MODEL CHECKING SECURITY PROTOCOLS:

A MULTIAGENT SYSTEM APPROACH

by

Ioana Cristina Boureanu

A thesis submitted to the Department of Computing

Imperial College London

in fulfillment of the requirements for the degree of

Doctor of Philosophy

Department of Computing

Imperial College London

2011

To my family: Ofelia, Aurica and Costachi

Familiei mele: Ofelia, Aurica şi Costachi

ABSTRACT

Security protocols specify the communication required to achieve security objectives, e.g., data-privacy. Such protocols are used in electronic media: e-commerce, e-banking, e-voting, etc. Formal verification is used to discover protocol-design flaws.

In this thesis, we use a multiagent systems approach built on temporal-epistemic logic to model and analyse a bounded number of concurrent sessions of authentication and key-establishment protocols executing in a Dolev-Yao environment. We increase the expressiveness of classical, trace-based frameworks by mapping each protocol requirement into a hierarchy of temporal-epistemic formulae.

To automate our methodology, we design and implement a tool called ***PD2IS***. From a high-level protocol description, `PD2IS` produces our protocol model and the temporal-epistemic specifications of the protocol's goals. This output is verified with the model checker `MCMAS`. We benchmark our methodology on various protocols drawn from standard repositories.

We extend our approach to formalise protocols described by equations of cryptographic primitives. The core of this extension is an indistinguishability relation to accommodate the underlying protocol equations. Based on this relation, we introduce a knowledge modality and an algorithm to model check multiagent systems against it. These techniques are applied to verify e-voting protocols.

Furthermore, we develop our methodology towards intrusion-detection techniques. We introduce the concept of *detectability*, i.e., the ability of protocol participants to detect jointly that the protocol is being attacked. We extend our formalisms and `PD2IS` to support detectability analysis. We model check several attack-prone protocols against their detectability specifications.

ACKNOWLEDGEMENTS

Firstly, I would like to bring my acknowledgements to my supervisor, Dr. Alessio Lomuscio, for his support and constant advice.

Secondly, I thank Dr. Mika Cohen for being an exceptional collegue, for sharing so many opinions about research and about life, in general.

Furthermore, I acknowledge my second supervisor, Dr. Michael Huth, for his availabity and systematic support.

And, I bring my special gratitude to Professor Ferucio Ţiplea who lead me to research and whose advice has always been of great value to me.

Also, I acknowledge Dr. Sorin Iftene and Dr. Liviu Ciortuz who inspired me during my undergraduate studies; that inspiration I have carried with me constantly.

I would also like to thank the Department of Computing of Imperial College London for the numerous opportunities I have enjoyed throughout my studies. I acknowledge the financial support of the Engineering and Physical Sciences Research Council, through their founding of the EP/E035655 project.

Then, I say "Thank You!" to Alex Summers, Dalal Alrajeh, Robert Craven, William Heaven, Jayshan Raghunandan, Jaspreet Shaheed, Tamas Suto, Simon Butler, Rueben Rowe, Francesco Russo, Juan Vaccari and Murilo Fernandes for making London feel more homely. And, I say "¡Gracias!" to Nicolas D'Ippolito for being a great friend over the last three years.

I would like to send my most sincere thanks to Tom Carlson for his understanding, love and care especially during the strenous moments of these studies and not only. Thank you, Tom!

Last, but certainly not least, I say "Mulţumesc!", a special thanks to my family and friends back home. This thesis is dedicated to my family.

DECLARATION

I hereby declare that I composed this thesis myself and that it describes my own research.

Contents

List of Figures

Chapter 1

Introduction

Motto: "The beginning is the most important part of the work."
(Plato)

In this chapter we present the motivations and contributions of the thesis, as well as a summary of its contents.

Our everyday life has become more and more reliant on sophisticated electronic devices: complicated surgery robots, iris readers, biometric passports, "chip-and-pin" or even contact-less payment cards. Human lives and their stability have come to depend on the correct functionality of automatic devices. Thus, it is vital that the software embedded in these devices is error-free. If this is not ensured, the users will be jeopardised and the production companies will have to face the costs of redeploying amended versions of their software. Therefore, *formal methods of verifying and validating* [176] the methodologies underlying safety-critical software products are indispensable. Many respected institutions value and invest in the verification of security-reliant systems, for example: Cisco Systems, Microsoft Corporation, IBM, VISA, MasterCard, RSA Security, Sun Microsystems, etc.

In particular, numerous safety-critical software products rely on *security protocols.* A security protocol is a "distributed algorithm defined by a sequence of steps precisely specifying the actions required of two or more entities to achieve a specific security objective." [145]. They are used in smart-cards, in electronic commerce, electronic banking, electronic voting, etc. Therefore, the formal verification of such protocols becomes of paramount importance. The *formal verification of security protocols* [149,112,41,95,5,182,34,144,3,153,138,72,138,163,79,181,2,23,19,9,44,8,175,171,102, 73,117,1,188,59,116] has contributed directly in showing that certain designs (e.g., variants [111] of the Kerberos protocol) were flawed, e.g., their privacy requirements could be violated by malevolent parties. Formal protocol verification inspects the existence of *protocol failures.* These occur when a protocol does not meet the goals for which it was intended, in "a manner whereby an adversary gains advantage not by breaking an underlying primitive such as an encryption algorithm directly, but by manipulating the protocol or mechanism itself" [145]. Being able to exhibit such failures on an early protocol design can prevent future financial losses and identity thefts. Moreover, security protocol verification has also revealed faults in protocols that had already been deployed (e.g., in the Secure Sockets Layer encryption protocol, used to secure communications on the Internet). This can put an end to unnecessary exposure to fraud. Through the abstract models used, security protocol verification has also benefited the design process, yielding clarity of requirements' specification and consequently contributing to new, more robust products. Examples of such beneficial impacts of security verification can be seen in several of the ISO security standards [82,85,84,83] available today.

Moreover, the *automatic* formal verification of security-reliant software has extended the practice of protocol testing, as lengthy manual proofs were replaced by seconds or milliseconds of running time on a PC. Thus, numerous systems and large spaces can be searched for errors in an instant. One of the successful techniques of automatic verification is *model checking* [51], which is today extensively used by many leading names in the software and hardware industry: JPL, NASA, Microsoft, etc. In particular, the *model checking of security protocols* [19,59,153,72] has also proven a resourceful technique in revealing modern, subtle security flaws (e.g., in the Single Sign On implementations of Google applications [10]).

In this thesis, we use formalisms drawn from logic and artificial intelligence to design *novel, systematic methodologies for model checking security protocols.* We take the seminal view of authentication logics: the security requirements of protocols revolve around certain facts being *known* or *unknown* to the protocol participants. For instance, when Alice receives a packet containing her one-time banking password, does Alice *know* that it has been indeed issued by the bank and that it indeed originates from the bank's server? Or, can Alice *know* that indeed this password has been recently generated in her current online banking session and that it has not been a mere replay from a malevolent party? In this thesis, this idea is taken further: can Alice *know*, e.g., by inspecting some of her different sessions, that one of her session is being hijacked?

This thesis introduces systematic multiagent system models (MAS) for several classes of security protocols executing in malevolent environments.

Additionally, we systematically map security requirements and properties related to them into hierarchies of temporal-epistemic formulae.

We automate the generation of our MAS-based models and the specifications to be verified.

Traditional and novel security aspects are verified by means of model checking these systems against temporal-epistemic specifications.

Theoretical and practical aspects of our AI-inspired approach are evaluated and discussed.

As a note, in this thesis we use "AI-inspired" with referrence to non-classical logics and multiagent systems, usually employed in the field of artificial intelligence (AI).

1.1 Motivations

In devising a knowledge-based approach to security verification, our **first motivation** is the formulation of protocol requirements. For instance, a goal of the NSPK (Needham-Schroeder Public Key) [155] protocol is formulated in the following way: "if the B-session completes, B is justified in *believing* that A holds K" ([66], page 3). Moreover, several ISO standards [82, 85, 84, 83] of security protocols place the knowledge of facts at the bottom of security requirements. It is precisely these ISO standards that lead Gollman[1] to found his systematisation [93] of security goals on logics of belief: e.g., $alice\ \texttt{believes}\ bob\ \texttt{believes}\ alice \overset{k_{ab}}{\leftrightarrow} bob$.

However, many formal methods approaches [44, 188, 175, 19, 73, 174] to protocol verification use a static notion of knowledge, its interpretation being often reduced to the possession of terms. In that

[1]For details, see Chapter 2, page 49.

sense, most of the formalisms for protocol verification express protocol requirements as *reachability properties*; the system satisfies, e.g., a key-establishment requirement if it reaches a state where a protocol participant possesses a certain value for the key K. In very restricted cases [59], some of the protocols' requirements are expressed not as reachability properties, but in linear temporal logic.

By contrast, we are motivated by lines of work where protocols' requirements are specified in more expressive languages, closer to their original formulations. Examples of the kind are Syverson's approaches [185, 184], rooted in logics of authentication [42, 95, 5, 191]. Therein, he argues that if a concept of participants' knowledge is to be used it needs to be a more expressive notion, one that goes beyond the possession of messages and that actually enquires the acknowledgement of facts. To motivate his view further, we exemplify with a requirement of the e-voting protocol in [158]. The privacy of *Alice*'s vote v is maintained if any attacker considers it possible that *Alice* votes another vote v' and that any other participant could have in fact voted v. Hence, the attacker *lacks the knowledge* of *Alice* having voted v. It is further argued [63] that certain security requirements (e.g., e-voting) indeed ought to be formulated in expressions other than reachability. Nevertheless, few protocol formalisms [2, 3, 177] support theoretically sound expressions of security properties beyond reachability. Moreover, practical verification tools rarely cater for the analysis of such advanced properties; when they do, they are semi-decidable [47].

Along the aforementioned lines, our **second motivation** is a supported [42,95,5,191,185,184,63] belief that several security requirements need to be formalised as expressive *state-properties* (i.e., properties interpreted over the entire unwound state-space) rather than as reachability, *trace-properties* (i.e., properties interpreted over a trace of the system). Moreover, we are motivated to provide sound methodologies and tools for the automatic verification of such advanced formulations of protocol goals.

A seminal effort in protocol verification is attributed to authentication logics [41,42,95,184,5,191] for early formalisation of protocol theories through logics of belief. The shortcomings of such logics (e.g., BAN [41], AT [5], GNY [95], VO [191]) were their poor systematisation and, most importantly, the lack of a well-founded semantics for the protocol theory employed. Fortunately, research [80, 55] into the computational soundness of cryptography has emerged in the more recent years. The modellings thereby used followed an $S5$ semantics. This led to several well-founded *theoretical* approaches employing epistemic logic in security settings [96, 99, 168]. Therefore, our **third motivation** is to revive the seminal approaches of authentication logics and take them further; we aim at specifying security protocols and their requirements as *well-founded*, expressive formulations based on epistemic logics.

Efficient tools for the automatic verification of non-classical and temporal-epistemic logics have been developed [127, 6, 110, 89] in recent years. Furthermore, these have been used successfully in the verification of a wide range of systems drawn from the distributed computing world, including [133, 130, 109].

These later developments underline our **fourth motivation**: at least in principle, it is now possible to develop toolkits based on temporal-epistemic formalisms and optimised for the symbolic model checking of security protocols against properties going beyond reachability specifications.

Some interest in model checking of epistemic expressions of security requirements has been recently shown [147, 130, 127]. However, these lines of work analysed specific protocols under ad-hoc modellings and no general, systematic methodology was put forward. Moreover, the consequent state space explosion was not studied there, thus making the approaches not viable for general, practical deployment. Closest to our line of work is the LDYIS formalism [125] (reviewed in Chapter 2, page 62). It puts forward a trace-based semantics for the NSPK protocol [155] using temporal-epistemic logic on top of interpreted systems [161]. LDYIS presents a basic bounded model checking algorithm for the verification of the model proposed. However, LDYIS does not give directions towards the systematisation and automation of model-generation, thereby limiting its possible impact.

The ad-hoc and pen-and-paper modellings recently used in epistemic verification of protocols and recalled above can be tedious and error-prone. Moreover, these limit the study of aspects like the scalability of the approaches taken. Therefore, our **fifth motivation** is to take incipient approaches of model checking of epistemic security requirements further into systematic, generalisable and automatable methodologies and toolkits. Furthemore, for distinct classes of security protocols we aim to create bespoke, systematic methodologies with the view of optimising the results of practical model checking.

To sum up, we are motivated in devising the theoretical and practical aspects of automatically obtainable, systematic multiagent system models for security protocols executing in a Dolev-Yao[2] environment. We aim at verifying traditional and novel security aspects by means of model checking these systems against systematically derived temporal-epistemic specifications.

1.2 Contributions

The main contributions of this thesis can be categorised in a fourfold way:

1. conceptual:
    - We distinguish two classes of security protocols: *receiver-transparent (RT)* and *receiver-opaque (RO)*. This is with the view of optimising the systematic generation of models for protocols and maximising the efficiency of the consequent model checking.
    - We introduce the concept of *detectability*, i.e., whether protocol parties are jointly able to detect protocol failures as soon as, after or whilst they occur.
2. methodological:
    - We introduce a novel, systematic methodology of modelling executions of receiver-transparent authentication and key-establishment protocols as multiagent systems.
    - We introduce a novel, systematic methodology of mapping each authentication and key-establishment goals into a taxonomy of temporal-epistemic formulae.
    - Using the multiagent system model aforementioned and temporal-epistemic logic, we introduce the first systematic methodology of specifying and verifying the detectability of protocol failures, i.e., whether groups of agents are theoretically able to detect protocol failures.
    - We introduce the first systematic methodology of modelling the executions of equationally-specified[3] protocols as multiagent systems. This also gives the first systematic methodology of modelling the executions of receiver-opaque protocols as MAS.

[2]The Dolev-Yao [71] is reviewed in Chapter 2, page 35.

[3]For details on equationally-specified protocols, see page 54.

3. algorithmic:

    - We introduce and implement algorithms for the automatic generation of multiagent system models for executions of `CAPSL`[4]-specified (receiver-transparent and receiver-opaque) security protocols.
    - We introduce a model checking algorithm for the verification of a correct notion of knowledge modulo cryptographic equational theories.

4. theoretical:

    - We prove the correctness of our AI-inspired protocol model with respect to traditional semantics for security protocols.
    - We prove the correctness of the interpretation of the cryptographic knowledge modality, introduced in the context of protocol expressed by equational theories.
    - We prove the correctness of the model checking algorithm of the knowledge modality introduced in the context of protocol expressed by equational theories.

By-products of these contributions will be outlined in the content of the chapters where they are presented.

To support the above description of our motivations and contributions, Chapter 2 contains a relevant literature review. After the complete enunciation of our results, in Chapter 8 we compare and contrast the related literature with the work of this thesis.

1.3 Research Output

Part of the research in this thesis was presented in the following papers:

- "A Compilation Method for the Verification of Temporal-Epistemic Properties of Cryptographic Protocols", by I. Boureanu and M. Cohen and A. Lomuscio, at the Joint International Workshop on Automated Reasoning for Security Protocols Analysis and Issues in the Theory of Security (ARSPA-WITS), 2009;

[4]For `CAPSL`, refer to [151, 66]. Or, for a review on `CAPSL`, see Chapter 2, page 54.

- "Model Checking Temporal Epistemic Specifications for Authentication Protocols Compiled into Interpreted Systems", by I. Boureanu and M. Cohen and A. Lomuscio, in the Journal of Applied Non-Clasical Logics, volume 19/4, pages 463–487, 2009;

- "Model Checking Detectability of Attacks in Multiagent Systems", by I. Boureanu and A. Lomuscio and M. Cohen, in the Proceedings of the 9th International Conference on Autonomous Agents and Multi-Agent systems (AAMAS), 2010;

1.4 Summary of Contents

In Chapter 2 we present the notions that constitute the background to this thesis and relate to the research hereby presented. In an attempt to make this thesis self-contained, we only selected those notions which we considered not necessarily commonplace.

In Chapter 3 we present the systematic design of a multiagent system model for the execution of receiver-transparent security protocols in a Dolev-Yao environment. It is based on the interpreted system formalism and the security requirements are expressed in temporal-epistemic logic.

In Chapter 4 we present an algorithm to produce the interpreted systems specifications aforementioned starting from a high-level `CAPSL` protocol description and the instantiation of the protocol parties involved. We then prove that the multiagent system models unwinding the specifications thus produced preserve the validation/refutation of standard security goals (i.e., standard `CAPSL` semantics validates these goals if and only the multiagent system models we produce do so too).

In Chapter 5 we present the toolkit `PD2IS` that, starting from a `CAPSL` protocol description, produces `ISPL` files describing multiagent system models of the kind introduced in Chapter 3. Several flavours of these models are presented. These differ in the level of optimisation for use with the `MCMAS` model checker. The files automatically generated by `PD2IS` also contain the systematic taxonomies of temporal-epistemic formulae corresponding to the protocol's security goals. Performance studies and other relevant metrics of the verification results are reported; comparisons between verifying the different flavours of these models, as well as comparisons between our methodology and other verification toolkits are presented.

In Chapter 6 we formalise the notion of *detectability*, the ability of groups of agents to acquire collective knowledge of protocol failures. This formalisation encompasses an MAS protocol model and a systematic taxonomy of temporal-epistemic formulae to describe the notion of detectability. We extend `PD2IS` to support the generation of these models and the corresponding formulae for detectability. We report on model checking MAS models for several well-known protocols against detectability specifications. We discuss the performance of the analysis process as well as the significance of the results.

In Chapter 7 we introduce *interrogative knowledge*, an epistemic modality modulo a convergent equational theory describing a protocol. We prove the correctness of the interpretation of this novel knowledge modality. We advance a model checking algorithm for interrogative knowledge. We use this novel methodology to analyse several automatically generated models for e-voting protocols expressed under convergent equational theories against security requirements specified in logics of time and interrogative knowledge.

The modelling and verification in Chapters 3–6 was applied to receiver-transparent protocols, i.e., protocols where any receiver can analyse messages down to atomic parts immediately upon their receipt. The introduction of a cryptographically adapt knowledge in Chapter 7 opens the way for a well-founded epistemic modality even for the case where the messages cannot be always analysed down to their atomic parts. The latter case is the case of *receiver-opaque protocols*. Therefore, the systematic model advanced in Chapter 7 applies to receiver-opaque protocols.

In Appendix C we give the details of a tool, similar in nature to `PD2IS`, but used for the systematic and automatic generation of IS-based models for receiver-opaque protocols. This tool is a compiler from a `CAPSL` protocol description to a `CIL`-like specification[5], to the `ISPL` encoding of the protocol's execution. Chapter 7 presents the automatic verification of receiver-opaque e-voting protocols. In Appendix C we also discuss experiments in generating and verifying several multiagent system models for other receiver-opaque protocols (e.g., advanced authentication protocols). Therefore, Appendix C completes the work in previous chapters. This renders our MAS methodology systematically and automatically applicable to both receiver-transparent and receiver-opaque protocols. A discussion on the suitability of our MAS approach for RTP or ROP will conclude Appendix C.

[5]For details on `CIL`, see page 57.

In Appendix A we present notions, proofs and implementation details adjacent to Chapter 3 and Chapter 5.

In Appendix B we give explanations, proofs and details additional to the content of Chapter 4. Most importantly, in Appendix B we show that our models are aligned to those in standard, trace-based semantics for `CAPSL`-described protocols by means other then the homomorphic relations presented in Chapter 4 (e.g., by the direct use of goal-satisfaction algorithms in [188] and, separately, by means of stuttering equivalence [37]).

In Chapter 8 we re-discuss our overall contributions, we compare them with existing related work and we draw the conclusions of the thesis.

In this chapter we have presented the motivations of this thesis and described its main contributions. We have concluded with the summary contents of the thesis.

Chapter 2

Literature Review

Motto: "What you are is what you have been, and what you will be is what you do now."
(The Buddha)

In order to highlight our motivations and contributions in protocol testing and AI-inspired verification, Chapter 1 mentioned a series of formalisms, tools and methodologies. In the current chapter, we detail their underlying notions. We focus only on those concepts which we consider not necessarily commonplace. We start with foundations of temporal-epistemic logics and computer-aided verification methods (e.g., model checking). We continue with notions of security protocol analysis. Lastly, we present the use of epistemic approaches in specifying and verifying cryptography-related aspects. After the main body of the thesis, in Chapter 8, the hereby presented related work will be compared and contrasted with our MAS approach to protocol verification.

2.1 Temporal and Temporal-Epistemic Logics

This thesis uses temporal and temporal-epistemic logics. We assume that the reader is familiar with the syntax and semantics of (propositional) modal logics [80, 104, 22, 45]. Therefore, we will only recapitulate notions which are inherent to temporal and temporal-epistemic logics. These include the syntax and semantics of temporal-epistemic logic, temporal properties of interest in the verification of distributed systems and model checking techniques for logics of time and knowledge.

2.1.1 Syntax

Branching time logic or *computational tree logic* (CTL) [20] is a fragment of the logics most addressed in this thesis.

CTL Syntax. Let PV be a set of *propositional variables.* Boolean formulae over PV are often called *atomic formulae.* The language of CTL is defined by:

$$\varphi ::= p \mid \neg p \mid \varphi \wedge \varphi \mid EX\varphi \mid EG\varphi \mid E[\varphi U \varphi], \tag{2.1}$$

where $p \in PV$.

We assume that the reader is familiar with general notions of transition systems. Thus, we report the respective readings of the above operators. $EX\varphi$ reads as "there exists a path such that at the next state on this path φ holds"; $EG\varphi$ reads as "there exists a path such that φ holds globally along this path"; $E[\varphi U\psi]$ reads as "there exists a path such that φ holds until ψ starts holding". The set $\{EX, EG, EU\}$ exhibited in 2.1 is the minimal or the canonical set of CTL operators. The CTL formulae generated by the grammar in 2.1 are in the so-called *existential normal form* (*ENF*). The dual of the existential *path operator* E quantifies paths universally and it is usually denoted A. Some of the equivalences over CTL satisfaction are: $EF\varphi \equiv E[\top U\varphi]$; $AX\varphi \equiv \neg EX\neg\varphi$; $AG\varphi \equiv \neg EF\neg\varphi$; $A[\varphi U\psi] \equiv \neg(E[\neg\psi U(\neg\varphi \wedge \neg\psi)] \wedge EG\neg\psi)$; $AF\varphi \equiv A[\top U\varphi] \equiv \neg EG\neg\varphi$. The interested reader is referred to [105] for details.

The logic CTLK contains the temporal operators of CTL and a modal operator for knowledge, denoted K. Equivalently, CTLK can be defined as the system obtained by the union of CTL and the modal system $S5$ [122] (with K as modal operator). In a multidimensional context, CTLK can be viewed as the union between CTL and the n-dimensional modal system $S5_n$ (roughly, $S5$ with n knowledge operators, usually denoted $K_1, \ldots, K_n$). These are formally expressed in the following.

CTLK Syntax. Let PV be a set of propositional variables. The language CTLK is given by the following grammar:

$$\varphi ::= p \mid \neg p \mid \varphi \wedge \varphi \mid EX\varphi \mid EG\varphi \mid E[\varphi U\varphi] \mid K\varphi, \tag{2.2}$$

where $p \in PV$, EX, EG, EU are CTL path operators and K is the *knowledge operator.*

We recall the axioms of the CTLK system (i.e., $S5$ with K denoting the the modal operator for knowledge):

- $K(p \to q) \to (Kp \to Kq)$ (rationality)
- $Kp \to KKp$ (positive introspection)
- $\neg Kp \to K\neg Kp$ (negative introspection)
- $Kp \to p$ (truthfulness)
- if $\vdash \varphi$ then $\vdash K\varphi$ (necessitation)

Let $Form$ be the set of CTLK formulae generated by the grammar in 2.2. We assume the implicit use of the *uniform substitution* (*US*) on $Form$, i.e., substitution of all occurrences of a variable with the same modal formula.

Extensions of CTLK exist [80, 148]. An example is the multidimensional system CTLK enriched with a modality for *distributed knowledge*, denoted D. It carries the meaning of cumulative or (implicitly) pooled together knowledge within the group of agents[1] Gr [80, 148]. Such a *group modality* is often informally assimilated to the concept of "wisdom of the crowds". When the group is not implicit, the distributed knowledge modality is indexed by name of the group, i.e., D_{Gr}. The formula $D_{Gr}\varphi$ denotes that the fact φ is *implicitly* known by the group Gr. The fact φ is made *explicit* if the agents in the group Gr collaborate actively. Such group modalities are often employed in the process of reasoning about multiagent systems [80]. Some of the fields where the use of such operators benefited the expressiveness of specifications are: game theory [68, 103], economic theory, puzzle solving [154, 70, 152].

As specification language, most chapters of this thesis use CTLK enriched with the distributed knowledge operator. Therefore, in the following by *temporal-epistemic logic* we intend this language, if not otherwise stated.

[1]The notion of "agents" will be formally introduced in the following. For now, informally assimilate the notion to people.

2.1.2 Semantics

The semantics for CTL is traditionally given by state-transition systems [50]. Also, a *de facto* semantics for modal logics is given in terms of Kripke models [114]. In this line of work, we primarily use the *interpreted systems* [161] formalism, an accepted semantics for temporal-epistemic logic. We mention that the interpreted systems (*IS*) formalism subsums the state-transition systems semantics for CTL. Furthermore, the IS semantics unwinds a Kripke model for knowledge.

Interpreted Systems. The interpreted systems formalism was introduced in the context of distributed systems [161]. It is extensively used in multiagent systems frameworks.

An interpreted system (IS) is defined upon the following elements:

- a finite set Ag of *agents*; agents are often indexed $1, \dots, n$;
- a finite set L_i of *local states* for agent i;
- a finite set Act_i of *local actions* for agent i;
- a function $P_i : L_i \to 2^{Act_i}$ for the *local protocol* of each agent i; P_i selects actions of agent i depending on its local state; if $a \in P_i(l)$, then the action a is said to be *enabled* at the local state l, for $a \in Act_i$ and $l \in L_i$;
- a special agent Env, the *Environment* of the system; this agent is described similarly to the other agents (i.e., it is associated with L_{Env}, Act_{Env}, P_{Env} as its local states, local actions and local protocol function);
- a set $G{=}L_1 \times L_2 \times \ldots \times L_n \times L_{Env}$ of *global states*;
- a function $l_i : G \to L_i$ that returns the local state of agent i from a given global state, for each $i \in Ag \cup \{Env\}$; an alternative notation to $l_i(g)$ is g_i, for $g \in G$ and $i \in Ag \cup \{Env\}$;
- a set $Act{=}Act_1 \times Act_2 \times \ldots \times Act_n \times Act_{Env}$ of *joint actions*;
- a function $E_j : L_j \times Act \to L_j$ for the *local evolution* of agent j, for each $j \in Ag \cup \{Env\}$; given a local state of the agent j and a global action, E_j defines the resulting local state of agent j;
- a function $E : G \times Act \to G$ for the *joint evolution*; for $g, g' \in G$, $a \in Act$, it is the case that $E(g, a) = g'$ if and only if $E_i(l_i(g), a) = l_i(g')$, for all $i \in Ag \cup \{Env\}$;
- a set $I \subseteq G$ of *initial global states*;

- a relation $V \subseteq G \times PV$ for a *valuation* relation.

An *interpreted system* $\mathcal{I}$ is then given by the tuple

$$\mathcal{I} = \langle (L_i, Act_i, P_i, t_i)_{i \in Ag}, (L_{Env}, Act_{Env}, P_{Env}, t_{Env}), I, V \rangle \, .$$

A *path* in an interpreted system $\mathcal{I}$ is an infinite sequence of global states described by the global evolution function: $\pi = (g_0, g_1, \ldots)$ such that $E(g_k, act_k) = g_{k+1}$, where $act_k \in Act$ is a joint action whose components are all enabled, for each $k \geq 0$. For a path $\pi = (g_0, g_1, \ldots)$, we write $\pi(k)$ to denote the state g_k along π. By $\Pi(g)$ we denote the *set of all the paths starting with* $g \in G$. Paths naturally give the set of *reachable* states. Naturally, it is possible to consider prefixes of paths which are finite; these are also referred to as *paths*.

Let $\mathcal{I}$ be an interpreted system, $g = (l_1, \ldots, l_n, l_E)$ be a global reachable state of $\mathcal{I}$ and φ, ψ be two CTL formulas. Then, satisfactions of these formulae on the interpreted system $\mathcal{I}$ at a reachable global state g is defined inductively as follows:

- $(\mathcal{I}, g) \models p$ iff $V(g, p) = true$
- $(\mathcal{I}, g) \models \neg\varphi$ iff it is not the case that $(\mathcal{I}, g) \models \varphi$
- $(\mathcal{I}, g) \models \varphi \wedge \psi$ iff $(\mathcal{I}, g) \models \varphi$ and $(\mathcal{I}, g) \models \psi$
- $(\mathcal{I}, g) \models EX\varphi$ iff there exists $\pi \in \Pi(g)$ such that $(\mathcal{I}, \pi(1)) \models \varphi$
- $(\mathcal{I}, g) \models EG\varphi$ iff there exists $\pi \in \Pi(g)$ and it is the case that $(\mathcal{I}, \pi(k)) \models \varphi$, for all $k \geq 0$
- $(\mathcal{I}, g) \models E(\varphi U \psi)$ iff there exists $\pi \in \Pi(g)$ such that $(\mathcal{I}, \pi(k)) \models \psi$ and $(\mathcal{I}, \pi(j)) \models \varphi$, for some $k \geq 0$ and for all j with $0 \leq j < k$.

Knowledge modalities are interpreted over an IS using equivalence relations over the states of the IS. Such equivalence relations are often referred to as *indistinguishability relations (over states)*.

State Indistinguishability. Two local states $l, l' \in L_i$ are *i-indistinguishable*, written $l \approx_i l'$, if and only if $\approx_i \subseteq L_i \times L_i$ is an equivalence relation over L_i, for $i \in Ag \cup \{Env\}$. In particular, this relation is often taken to be simply the equality relation (i.e., for some $i \in Ag \cup \{Env\}$, $l \in L_i$ is i-indistinguishable from $l' \in L_i$ if and only if $l = l'$).

Two global states $g, g' \in G$ are *i-indistinguishable*, written $g \sim_i g'$, if and only if the correspondent local states of agent i are *i-indistinguishable*, i.e., $l_i(g) \approx_i l_i(g)$, for $i \in Ag \cup \{Env\}$. We call the relation $\sim_i \subseteq G \times G$ the *indistinguishability relation*. Note that the indistinguishability relation can be lifted to a group of agents Gr, $\sim_{Gr} \subseteq G \times G$. A standard lift [80] is based on the intersection on the relations $\sim_i \subseteq G \times G$, for all $i \in Gr$, i.e., $\sim_{Gr} = \bigcap_{i \in Gr} \sim_i$.

We can now resume presenting the semantics of temporal-epistemic logic on interpreted systems, by recalling the interpretation of knowledge modalities. Let $\mathcal{I}$ be an interpreted system and g be a reachable state of $\mathcal{I}$. Then,

- $(\mathcal{I}, g) \models K_i\varphi$ iff $(\mathcal{I}, g') \models \varphi$ for all reachable states g' satisfying $g \sim_i g'$;
- $(\mathcal{I}, g) \models D_{Gr}\varphi$ iff $(\mathcal{I}, g') \models \varphi$ for all reachable states g' satisfying $g \sim_{Gr} g'$.

A system $\mathcal{I}$ *validates* a CTLK formula φ if the formula is satisfied at any reachable state g:

$$\mathcal{I} \models \varphi \text{ iff } (\mathcal{I}, g) \models \varphi, \text{ for any reachable state } g \text{ of } \mathcal{I}.$$

We refer to [80] for details on the interpreted system formalism.

We now recall some of the properties of interest modelled and studied in software verification.

Important Temporal Properties of State-Transition Systems.

• *Reachability* denotes the property that there is a computation path of the system leading to a particular state (i.e., a state with a certain property);

• *Safety* denotes the property that the system does not reach a state where a certain undesirable property holds.

• *Termination* or *finiteness* denotes the property that every computation of the system leads to a terminal state (hence, where no transition is possible).

• *Fairness* is the property that on every computation path of the system a certain property is satisfied infinitely many times.

• *Reactivity* denotes that a certain property eventually holds.

We will now report the CTL specifications of the properties above and their respective readings. We mention that there are no exact standards stipulating these CTL specifications and that we follow the lines in [16]. Reachability can be verified by using the following CTL specification: $AG(x)$ – x does hold at all points, on all paths. Hence, a state where $\neg x$ is reachable on some path. In the same approximating sense, we show how the above properties can be directly or indirectly verified as CTL specifications. Safety can be expressed in CTL as $\neg x\, AW\, y$: on any path, x does not occur before the first occurrence of y and y might never occur. Reactivity can be expressed in CTL as $AG(x \Rightarrow AFy)$: on any path, at any point it is the case that once x occurred, y will always occur eventually.

We now proceed to summarise some notions of computer-aided verification. In particular, we focus on the model checking of temporal-epistemic logics.

2.1.3 Model Checking of Temporal and Temporal-epistemic Logics

Given a general[2] model M for a system and a specification φ encoding one of the system's properties, the *model checking* [51] problem consists in establishing whether the model validates the formula, i.e., $M \models \varphi$[3]. In this thesis we will be employing model checking of interpreted systems against temporal-epistemic formulae.

Explicit Model Checking for CTLK. As its name says, *explicit model checking* [50, 51] is a type of model checking in which states and relations between states are encoded in an explicit manner. Explicit model checking for concurrent programs [115] against CTLK specifications is introduced in [129]. The underlying semantics used for the models follows closely the interpreted systems formalism. The procedure for the temporal fragment of the logic is in the style of the more traditional model checking of state-transition systems [37].

Let M be a concurrent-program model [129] (in the style of an IS) and φ be a CTLK formula. Let the notation $[\![\varphi]\!]$ denote the set $\{g \mid g \in G \wedge (M, g) \models \varphi\}$, i.e., the set of states which satisfy φ in M. The explicit model checking procedure as presented in [129] decides whether the set I of initial states is a subset of the states where φ is satisfied in M, i.e., $I \subseteq [\![\varphi]\!]$. For an agent i, model checking $K_i\varphi$ follows in [129] similar lines to the standard method applied for the temporal $EX\varphi$ operator [50,51] (i.e., constructing the closure of the set of states that satisfy $\neg\varphi$ under the pre-image of the $=_i$ indistinguishability relation). We assume that the reader is familiar with explicit model

[2]By "general" we mean that it does not need to be an interpreted system necessarily.

[3]In this secton, we assume that the computation of $\models$ is feasible and, in this general setting, ignore details, such as the set of initial/reachable states considered under $\models$, etc.

checking procedures; if needed, please refer to [50,51,129].

The complexity of deciding $M \models \varphi$ is of the order of $O(|\varphi| * (|G| + \tau))$, where $|\varphi|$ is the size of the formula φ, $|G|$ is the number of states in the model M and τ is the number of possible transitions in the model M.

OBDD-based Symbolic Model Checking of Temporal-Epistemic Specifications. Note that n propositional variables correspond via the valuation relation to 2^n states. This exponential ratio gives rise to the *state-explosion problem*. Given the complexity of the explicit model checking algorithm, the state-explosion problem makes its use inefficient or even impractical in real settings. In order to alleviate the state-explosion problem, *symbolic model checking* [143] uses efficient encodings of the sets of states and of the transition relation. Such encodings are referred to as *symbolic representations*.

There are different classes of data structures used to attain symbolic representations of states and transitions. One class of such data structures is called *reduced ordered binary decision diagrams* (`ROBDD`) [38]. In brief, a *(reduced) binary decision diagram* (*BDD*) is a compact representation of a binary tree. An *ordered BDD* (*OBDD*) is a BDD in which the boolean variables appear in the same order along each root-to-leaf path. These representations are at the basis of *OBDD-based symbolic model checking*. For a detailed review on BDDs, we refer the reader to [105].

The first step towards symbolic model checking is to express $[\![\varphi]\!]$ as OBDDs. Then, a symbolic model checking procedure represents operations over the OBDD-encoded sets as OBDDs. It uses the Shannon expansion [12] to model these encodings. Lastly, it simply follows the same steps as the explicit model checking algorithm. The interested reader is referred to [143,169].

`MCMAS` [127] is a tool for model checking multiagent systems. `MCMAS` is a symbolic model-checker for the verification of interpreted systems models against specifications given in temporal-epistemic and coalition logics. More precisely, `MCMAS` supports specifications based on CTL, epistemic logic (including operators of common [80] and distributed knowledge), *alternating time logic* (*ATL*) [7], and deontic modalities for correctness [131]. The symbolic representations used in `MCMAS` are ROBDDs. For the manipulation of BDDs, the implementation of `MCMAS` uses the `CUDD` library [179].

Interpreted systems models are described in `MCMAS` using `ISPL` (*Interpreted System Programming*

Language). The ISPL language has a syntax similar to SMV [46], as the next exemplifications will show. We proceed by giving the generic structure of an ISPL program:

1 - The *agents declarations* are defined by a sequence of agent-describing sections. These sections correspond respectively to the declarations of local states, local actions, local protocol and local evolution function. For instance, the local states are declared in terms of sets of variables, each ranging over a domain. Such domains are either ISPL standard types (e.g., bounded integer, boolean, etc.) or they can be defined by the user as enumeration types (e.g., $\{value_1, \ldots, value_n\}$). The local evolution is described in an "if-then" style, specifying the preconditions and postconditions for an action to take effect at a local state of an agent (i.e., $\langle post\_conds \rangle \ if \ \langle pre\_conds \rangle$).

```
Agent <agentID>
 <agent_body>
end Agent
```

2 - The *evaluation section* constitutes the ISPL specification of atomic formulae, i.e., predicates. These predicates are used as the building-blocks of the formulae to be checked.

```
Evaluation
 <proposition_declaration>
end Evaluation
```

3 - The *initial states* section declares a set of conditions on the variables of the agents, such that the set of possible initial global states is specified.

```
InitStates
 <condition_on_states>
end InitStates
```

4 - The *declaration of groups* is an optional section for explicitly specifying subsets of the set of agents. These subsets are used to index group-modalities within formulae.

```
Groups
 <groups_declaration>
end Groups
```

5 - The *formulae Section* stipulates the list of specifications to be checked.

```
Formulae
 <formulae_list>
end Formulae
```

Following the interpreted systems formalism, a special agent called Environment is declared in all ISPL files.

To give a flavour of the ISPL specifications, Example 2.1.1 cites from an ISPL example-program provided with the MCMAS distributions [127].

Example 2.1.1 (An Excerpt of the Bit-Transmition Protocol [80] modelled in ISPL)

```
   ...
Agent Environment
Vars:
state : {S,R,SR,none};
end Vars
Actions = {S,SR,R,none};
Protocol:
state=S : {S};

state=R : {R};
...
end Protocol
Evolution:
state=S if
(Action=S) or (Action=SR) or
(Action=R) or (Action=none);
...
end Evolution
end Agent
Agent Sender
Vars:
bit : { b0, b1};
ack : boolean;
end Vars
Actions = { sb0,sb1,nothing };
Protocol:
bit=b0 and ack=false : {sb0};
bit=b1 and ack=false : {sb1};
ack=true : {nothing};
end Protocol
Evolution:
(ack=true) if (ack=false) and
(Receiver.Action=sendack)
and (Environment.Action=SR) or
(Environment.Action=R));
end Evolution
end Agent
Agent Receiver
...
end Agent
Evaluation
recbit if ( (Receiver.state=r0) or ... );
recack if ( ( Sender.ack = true ) );
...
end Evaluation
InitStates
```

```
( (Sender.bit=b0) or (Sender.bit=b1) ) and
( Receiver.state=empty ) and
( Sender.ack=false) and
((Environment.state=none) or
(Environment.state=SR) or
(Environment.state=S) or
(Environment.state=R));
end InitStates
Fairness
envworks;
end Fairness
Formulae
E((bit0 or bit1) U (recack));
EF(EG(recbit and !recack));...
AG(recack -> K(Sender,(K(Receiver,bit0)
or K(Receiver,bit1))));
end Formulae
```

We presented the simple syntax of ISPL/MCMAS 0.7. Newer versions (hence, the current MCMAS 1.0 [126]) support more advanced features (e.g., variables in the *Environment* declaration can be qualified as *observable* to be globally accessed or, *local observable* to be accessed by particular agents, etc.). We refer the interested reader to [127, 126].

Other model checkers for temporal and temporal epistemic logics exist: e.g., MCK [89], Verics [110] and MOCHA [6]. For instance, MCK [89] is a model checker mostly used with specifications in logics of linear time and knowledge. Its underlying multiagent system semantics is somewhat different from that of MCMAS (e.g., in the case of perfect recall [146] semantics, etc.). An example where a comparison of performance between the two model checkers is drawn is [69].

In this section we have summarised the following notions: the syntax and semantics of temporal-epistemic logic, temporal properties of interest in the verification of distributed systems and model checking techniques for logics of time and knowledge. These constitute the main languages, semantics and verification methodologies that we will use in this thesis.

2.2 Verification of Security Protocols

In this section we summarise the main notions and methodologies employed in formal verification of security protocols. We start with a synopsis of security protocols. We continue with formalisms employed in formal methods verification of security protocols (i.e., *symbolic protocol verification*). It is generally accepted that there are three main approaches to the verification of abstract protocol models: logic-based, trace-based and those based on process algebra. For each approach, we present at least one well known formalism. For clarity of later sections, a distinct subsection is allocated to the formalisation of security requirements. The section concludes with a review of high-level specification languages for security protocols.

2.2.1 Security Protocols — an Overview

2.2.1.1 The Basics

Security protocols are succinctly depicted in *high-level protocol descriptions*. Two of the common methods to achieve this are the *Alice & Bob* notation [173] and UML sequence diagrams [40]. To illustrate, Example 2.2.1 shows the well known Needham Schroeder Public Key (NSPK) protocol [155] in the Alice & Bob notation:

Example 2.2.1 (The NSPK Protocol in the Alice & Bob Notation)

$$1.\ A \rightarrow B : \{A, N_A\}_{pub(B)}$$

$$2.\ B \rightarrow A : \{N_A, N_B\}_{pub(A)}$$

$$3.\ A \rightarrow B : \{N_B\}_{pub(B)}$$

The protocol description involves two *principals*: A and B. Each implies one *role*: the role of A (or, the A-*role*) and the role of B (or, the B-*role*). A concrete *participant* (e.g., *alice*) playing the role of A encrypts its name and its nonce[4] N_A with the public key $pub(B)$ of a B-*role* participant, and sends the resulting encryption $\{A, N_A\}_{pub(B)}$ to this B-*role* participant. The participant playing the role of A then waits for a message encrypted with its own public key $pub(A)$, containing its own nonce N_A and a new nonce N_B. Following this, the participant proceeds in his A-*role* by encrypting the received nonce N_B with the public key $pub(B)$ and then sends this encryption back to N_B's originator, the B-*role* participant. Dually to an A-*role*, the first action of a participant playing a B-*role* is awaiting for a message encrypted with its own public key $pub(B)$, etc.

As suggested from the above, a high-level description implicitly underlies one protocol *session*. In the case of the NSPK protocol, it is a session between an A-*role* participant, say *alice*, and a B-*role* participant, say *bob*. In other words, *alice* is an A-*role instance* and *bob* is a B-*role* instance

[4] "Nonce" stands for "number once used", i.e., random, unpredictable value.

in one concrete session. To consider multiple sessions, several instances of (at least) one role are instantiated. Such an instantiation is often referred to as a *protocol scenario*. In some scenarios *alice* can be the identity of several instances: e.g., two instances of the $A\text{-}role$ and one of the $B\text{-}role$. For example, in a fair-exchange protocol [13], *alice* can play the role of the ***B**uyer* in two of her running sessions and that of the ***A**uctionneer* in a third, simultaneous session. The multiple role instances and the non-determinism of their interleaving are some of the sources of the subtle protocol failures and of the great computational difficulty in finding these failures.

In the following, we will summarise some basic notions of security requirements and protocol attacks.

2.2.1.2 Mainstream Security Protocol Requirements

Following the lines in [145, 192], we will summarise the cryptographic meaning of security protocol requirements, also called *protocol goals*.

• *Confidentiality* or *secrecy* expresses the requirement of keeping certain information unavailable to unauthorised parties. For instance, the value of the nonce N_A in the NSPK scenario aforementioned should not leak to an intruder.

• *Privacy* denotes the requirement that the link between a certain information and a particular party is unknown to protocol observers. For instance, in electronic-voting protocols (e.g., [158]), the observer should not know that *alice* voted v.

• *Data integrity* is the requirement that alteration of data or data manipulation (insertion, deletion, and substitution) by unauthorised parties should not occur. A particular type of data integrity, often reminded, is *freshness*. It broadly means that the messages transmitted in a session were generated purposely for that session, e.g., are not replayed from older executions.

• *Authentication* is the requirement specifying that two or more protocol parties are mutually identifiable. It can view entities or information itself. This requirement usually goes beyond secrecy: e.g., the data might be known to remain secret, but two of the parties involved might desire a proof that their identities are indeed as claimed or that information indeed originated from one party, being aimed at the other. As such, authentication requirements are usually subdivided into two major classes: *entity authentication* and *data-origin authentication*.

• *Non-repudiation* is the requirement stipulating that entities should not be able to deny their previous actions. Trusted third parties (TTPs) are sometimes (e.g., [21]) employed as a means to resolve non-repudiation disputes.

If more specialised classes of security protocols are considered (e.g., contract-signing, electronic voting protocols), then more advanced security requirements are of interest.

• *Anonymity* is the requirement that a certain party cannot be linked to some data or an action that he/she has respectively transmitted or performed.

• *Eligibility* is the requirement that only eligible parties can engage in the protocol sessions and only once.

• *(Voting-)fairness* is the requirement that the actions/data of one party cannot be influenced by data already transmitted.

• *Individual (voting-)verifiability* stipulates that any voter engaged in the protocol is able to verify that his/her vote was counted.

• *Universal (voting-)verifiability* requires that any voter engaged in the protocol is able to verify that the published result is indeed the sum of the votes cast.

• *Receipt-freeness* demands that no voter party posseses or knows a receipt or proof of the way he/she voted.

• *Coercion-resistance* requires that no voter party can be coerced to cast a certain vote.

2.2.1.3 Security Protocol Attacks

In this section we describe the notion of protocol attack and the context in which it occurs.

The communication channels that support the deployment of most protocols are considered *unsecured* (unless specified otherwise). On such a channel unintended parties can reorder, alter, delete, insert, or read the data transmitted. In contrast, an adversary is not able to alter, redirect or read the data communicated on a *secured* channel. Such unintended parties are usually referred to as *attackers*, *adversaries* or *intruders.* An attacker can be *active* or *passive.* A passive attacker can only observe/read the information from an unsecured channel. In turn, an active adversary is an adversary who may transmit, alter, or delete information on an unsecured channel. The model for active attackers most often assumed in symbolic verification methods is the *Dolev-Yao thread* *model* [71]. In brief, a Dolev-Yao adversary:

- can copy every communication in the system;
- can block any message;
- can impersonate any honest participant;
- has unlimited computational power;
- can keep record of any public system event;
- cannot generate the secret data of honest participants;
- cannot break encryptions.

Also, it is to be noted that any group of Dolev-Yao intruders actively cooperating cannot cause more attacks than a single intruder acting alone [183].

An *attack* can be described as a protocol execution in which the adversaries manipulate the data and/or the participants in such a way that at least one of the protocol requirements is not fulfilled. Verification and cryptography research has lead to differentiate certain types of attacks. Following the lines of [145] and those of [48], we summarise the types of protocol attacks investigated in symbolic methods of cryptography analysis.

- *Known-key attack.* This attack is lead by the adversary having obtained some previously used keys and using this information to manipulate the protocol in his favour.
- *Replay attack.* In this attack the adversary records data from previously run sessions, manipulates it and inserts the results in current and later sessions. As opposed to the known-key attack, this attack is not based on the adversary necessarily knowing a leaked (long-term) key.
- *Impersonation attack.* In this attack the intruder engages in the communication by assuming partially or entirely the identity of one of the legitimate protocol parties.
- *Interleaving attack.* This type of attack usually employs impersonation techniques in an execution where multiple sessions are interleaved. It is sometimes referred to as a *man-in-the-middle attack*, as the intruder interposes in the communication of two honest parties, by impersonating one of them.
- *Type-flaw attack.* In this attack the adversary inserts a corrupted message with the purpose that the recipient would interpret part of it as what it is not, e.g., take a nonce to be a key, etc.
- *Binding attack.* In this attack the intruder succeeds in making his public key appear as the public key of an honest party, e.g., *alice* is misled into believing that $K_{intruder}$ is in fact K_{bob}.

In the following, we give the example of an attack against the Woo-Lam protocol [194]. This attack falls into several of the categories above. To begin with, Example 2.2.2 shows the Alice & Bob notation of the Woo-Lam protocol.

Example 2.2.2 (The Woo-Lam Protocol [194] in the Alice & Bob Notation)

$$\begin{array}{lll}
1. \; A \rightarrow B & : & A \\
2. \; B \rightarrow A & : & N_b \\
3. \; A \rightarrow B & : & \{A, B, N_b\}_{K_{AS}} \\
4. \; B \rightarrow S & : & \{A, B, \{A, B, N_b\}_{K_{AS}}\}_{K_{BS}} \\
5. \; S \rightarrow B & : & \{A, B, N_b\}_{K_{BS}}
\end{array}$$

The Woo-Lam protocol is an one-way authentication protocol, based on a trusted server. A initiates the communication by sending her identity to B. At its turn, B replies by sending back a nonce N_b. This nonce and the entities of the parties are wrapped by A in a packet encrypted with a shared-key between A and the server S. At step 3, A sends this packet to B. This principal cannot decrypt the message. In turn, he uses a key that he shares with the server S to encrypt a packet containing A and B's identities and the cyphertext previously received from A. Then, at step 4, B sends this lastly formed packet to the server. The server has all the due keys to decrypt the message and sends back to B a packet containing A, B, Nb, encrypted with their shared key. Upon receipt, if the message encrypted with K_{BS} contains the identity of A, B's identity and $B's$ original nonce N_b, then this should reassure B that A has a key with the server and indeed forwarded the correct nonce N_b in the packets. In other words, A authenticated herself to B (by using S as a TTP).

We now illustrate a known attack [194] on the Woo-Lam protocol:

Example 2.2.3 (An Attack on the Woo-Lam Protocol [194])

$$\begin{array}{lll}
1'. \; I_A \rightarrow B & : & A \\
2'. \; B \rightarrow I_A & : & N_b \\
3'. \; I_A \rightarrow B & : & N_b \\
4'. \; B \rightarrow I_S & : & \{A, B, N_b\}_{K_{BS}} \\
5'. \; I_S \rightarrow B & : & \{A, B, N_b\}_{K_{BS}}
\end{array}$$

In this attack, the intruder begins by impersonating the honest principal A, hence the notation I_A. He sends to B the identity of A. In the second step, B replies to the purported A with the freshly generated nonce N_b. But this message is intercepted by the intruder. In the third step, the intruder impersonates A again and replays B's own nonce N_b. Recall that, in a normal execution, the third-step packet $\{A, B, N_b\}_{K_{AS}}$ is un-decryptable by B. So, B can take the nonce N_b inside the intruder-originated packet to be in fact the bitstring for $\{A, B, N_b\}_{K_{AS}}$. This means that the intruder produces a type-flaw fraud. At step $4'$, B sends to the server his newly formed packet. This packet is however intercepted by the intruder. At the final step, the intruder impersonates the server S and replays to B the message sent at step $4'$. This message passes the tests that B would normally do at step 5, i.e., the message received by B in step $5'$ has the form that B would expect and it deludes B into thinking that he has been in a communication with A, agreeing with A upon N_b and being certified by a TTP.

Symbolic verification of cryptographic protocols addresses the type of attacks presented in this section (i.e., not those reliant on cryptanalysis and manipulation of cryptographic primitives). In the following, we summarise the symbolic approach to protocol analysis.

2.2.2 Formalisms for Specification and Verification

Security Verification Communities — a Synopsis. There are two main approaches to the verification of security protocols: *computational* verification methods and *symbolic* verification methods. Computational verification methods do not abstract much from the underlying cryptographic mechanisms: messages are bitstrings, cryptographic primitives are functions, the malevolent parties are non-deterministic probabilistic polynomial time Turing machines, etc. In turn, symbolic protocol verification (also, called *formal protocol verification*) makes certain assumptions that abstract the cryptographic mechanisms. For instance, a general assumption is that of *perfect cryptography* [71]: an encrypted message cannot be decrypted unless the appropriate decryption key is known. The attacker is also abstracted into the Dolev-Yao thread model. Such assumptions lead to convenient algebraic approximations of the protocol executions.

Problems of deciding whether security protocols meet even simple requirements (e.g., secrecy or confidentiality) are computationally expensive [73, 187, 171]. In particular, in the case of symbolic models, if no restrictions are considered, the problem of deciding secrecy is undecidable [172, 187]. Undecidability is specific to models where an unbounded number of parallel protocol sessions are considered or the number of sessions are bounded, but either infinitely many nonces or arbitrarily long messages are allowed. Nonetheless, if some of these aspects are restricted, the problem of deciding secrecy still remains of high complexity (e.g., deciding secrecy for the case of bounded number of sessions with finitely many nonces and bounded message-length is DEXPTIME-complete [188]). These results refer only to the difficulty of deciding simple security properties (e.g., secrecy) of cryptographic protocols; meanwhile, deciding more complicated properties dwelling on advanced protocol requirements (e.g., coercion-resistance) remains decidable only for (very) restricted cases [1] (e.g., passive intruders and specific classes of protocols).

During the late eighties and the nineties, formalisms and tools were developed in order to analyse security protocols by means of symbolic models. The great majority of these methods focused on authentication protocols. A comprehensive list of lines of work in this field, in the chronological order of their publication dates is [149, 112, 41, 95, 5, 182, 34, 144, 3, 153, 138, 72, 138, 163, 79, 181, 2, 23, 19, 9, 44, 8, 175, 171, 102, 73, 117, 1, 188, 59, 116]. There existed several logic-based approaches to security protocol verification (especially dedicated to authentication); we recall authentication logics like BAN [41, 42], GNY [95], SVO [184] and AT [5]. BAN was one of the first methodical approaches of protocol analysis and it became very popular at its time. However, the lack of axiomatisations and semantics soon raised criticisms [156]. Other logics (e.g., GNY, SVO, AT, etc.) were developed mostly as a response to BAN's shortcomings, but none of these enjoyed a long-lasting success. This was due to the lack of axiomatisations, to flaws discovered in the underlying methodologies or to the lack of computer-aided tools to analyse these logic systems. The protocol verification techniques emerging in later years clearly distinguish two mainstream directions: the trace-based formalisms and the process algebra-based ones. Amongst the trace-based prevalent semantics we recall those based on inductive approaches [162], rewriting logics [44, 67, 175], Horn clauses [24], strand spaces [79]. In turn, frameworks based on process algebra made possible verification lines like CSP/FDR [177, 72], spi-calculus [3] or the newer applied pi calculus [2]. In the following, we will summarise those aforementioned frameworks which are more closely related to this thesis.

2.2.2.1 Authentication Logics

BAN Logic. Introduced in [41] and revised in [42], BAN is a logic of belief, designed for the specification and analysis of authentication protocols. BAN also deals with key establishment protocols, but it assumes that the latter form part of the class of authentication protocols. In a nutshell, the BAN-analysis means looking at the beliefs of the honest participants after they have executed the protocol. Firstly, the protocol is *idealised* (e.g., parts of messages are replaced by formulae attaching a belief-related "meaning" to them). Secondly, assumptions about the initial state are made explicit. A procedure of manual annotation of the protocol follows: e.g., for each transmission rule $A \rightarrow B \,:\, M$, an assertion of type B *received* M annotates the rule. Finally, the logic is used to derive the beliefs held by protocol principals.

We now present some of BAN's syntax and inference rules. We use a notation closer to the natural language, in the style of the later authentication logics [5]. In the following, X is a message or a formula.

BAN syntax is given by assertions of the type explained below:

- A `believes` X — A may act as if X is $true$;
- A `received` X — A has received a message containing X, either in plaintext or in cyphertext;
- A `said` X — at some point in the past, A sent X in plaintext or in cryptotext and, at that point, A believed X and A acknowledged the sending of X;
- A `controls` X — A *has jurisdiction* on X, i.e., A should be trusted with respect to X;
- $fresh(X)$ — X has not been sent in any previous messages;
- $A \overset{k}{\longleftrightarrow} B$ — k will not be learned by any other parties, but A, B and (possibly) TTPs; the BAN-reading of $A \overset{k}{\longleftrightarrow} B$ is k is a *good key for* A *and* B;
- $\mathtt{PK}(A, k)$ — k is the public key of A;
- $\{X\}_k$ `from` A — A can recognise her own messages.

For instance, one of the *inference rules* of BAN is the *nonce verification* rule: $\dfrac{A\,believes\,fresh(X) \quad A\,believes\,B\,said\,X}{A\,believes\,B\,believes\,X}$. In light of the above, this rule is self-explanatory.

Whilst BAN has been successful in pen-and-paper proving certain properties of protocols, many criticisms have been brought to it (most arising mainly from the lack of a formal semantics). For instance, in [156], it is shown how using BAN logics one would prove that a certain protocol is safe, when in fact one of the protocol's keys is blatantly attack-prone. Other critiques refer to the fact that possession of keys could not be expressed in BAN and that universal quantification is implicit in BAN. The latter led to ambiguities of the following kind: e.g., $A\,\texttt{believes}\,S\,\texttt{controls}\,A \xleftrightarrow{k} B$ can imply either $A\,\texttt{believes}\,\forall k\,(S\,\texttt{controls}\,A \xleftrightarrow{k} B)$ or $A\,\texttt{believes}\,S\,\texttt{controls}\,\forall k\,(A \xleftrightarrow{k} B)$, etc.

BAN's Successors. As a consequence of the criticisms brought to BAN, several other authentication logics emerged in the early nineties. For instance, the *GNY* logic [95] refined subtle details in the BAN-idealisation of protocols. Also, the GNY logic supported the expression of key-possession, differentiated between inherently owned, locally generated messages and received messages. Through these amendments, GNY eliminates BAN's flaw exposed in [156]. However, BAN's lack of a formal semantics is inherited by GNY as well.

In 1991, Abadi and Tuttle created an authentication logic aimed at tackling the semantic inconsistencies discovered for BAN. This logic was called *AT* [5] and it exhibited a soundness proof using an underlying modal logic approach. This was a major step towards offering a sound logical ground to BAN-style approaches. However, the soundness proof in [5] was sketchy and its hypotheses were later found to be too strong.

To address an expressiveness deficiency of BAN's, the *VO* logic [191] was put forward. Unlike its predecessors, VO could formalise modern public-key authentication as it is used today in SSL and TLS protocols.

In 1994, Cervesato and Syverson created the *SVO* logic [182] (and, in 1996, re-affirmed [184] it). SVO is an authentication logic that was aimed at combining the facilities of all its BAN-like predecessors and at offering a sound semantics to the language used. While much in the style of BAN, SVO had an S5 [122] axiomatisation and an S5-based semantics.

The lack of semantics or tools to support computer-aided verification diminished the possible impact of these logic-based formalisms. They were replaced by other formal methods approaches to security verification, some of which are summarised in the sequel.

2.2.2.2 Rewriting-based Formalisms

A widely used rewriting-based formalism used in security protocol specification and verification is called *multiset rewriting* [44]. Concisely, the multiset rewriting formalism consists of:
- a *signature*, which specifies a set of *sorts* (e.g., keys, nonces, etc.) together with function and predicate symbols (each symbol having an associated, specific type over the sorts);
- a *set of variables*, each having an associated sort;
- *terms*, which are defined as usual over the signature;
- *atomic formulas*, which are constructs of the form $P\,(t_1, \ldots, t_n)$; P is a predicate symbol of type $s_1 \times \ldots \times s_n$ and t_i is a term of sort s_i, for any i;
- *facts*, which are atomic formulas $P\,(t_1, \ldots, t_n)$, where all terms t_i are ground terms;
- *states*, which are multisets of facts;
- *rules*, which are constructs of the form $F_1, \ldots, F_k \rightarrow \exists x_1 \ldots \exists x_j.G_1, \ldots, G_n$, where F, G are atomic formulae, $x_1, \ldots, x_j$ are variables; existential quantifiers in the right hand side of rules, simply called *existentials*, capture the generation of new elements (e.g., short-term keys and nonces).

The *multiset rewriting semantics* is based on state-transition systems, where the moves from one state to another are achieved by applying a transition/rewriting rule τ such as the one described above, e.g. $F_1, \ldots, F_k \rightarrow \exists x_1 \ldots \exists x_j.G_1, \ldots, G_n$. A resulting state has $\sigma F_1, \ldots, \sigma F_k$ removed, $\sigma G_1, \ldots, \sigma G_n$ added and $x_1, \ldots, x_j$ replaced with new symbols, where σ is a fully ground substitution. Free variables in such transition rules are considered universally quantified.

Most of the results obtained under multiset rewriting use the formalism in its *restricted form.* A *restricted role* is a set of constructions of the form: $A_i(\ldots), N_{R_j}(\ldots) \rightarrow \vec{\exists} \ldots A_k(\ldots), N_{S_l}(\ldots)$, where $A_1, \ldots, A_m$ is a finite list of predicates defining the role-states (i.e., states of each instantiated role), $N_{R_1}, \ldots, N_{R_n}$ and $N_{S_1}, \ldots, N_{S_n}$ are finite lists of network predicates, $i < k \leq m$ and $j < l \leq n$; generically, N_R and N_S a network predicates for receive and send actions, respectively.

The execution environment for a language denoting restricted form multiset rewriting is called *MSR* [181]. Therefore, the multiset rewriting formalism itself is often called *MSR*. The MSR execution environment runs on top of the well known rewrite engine called Maude [52].

Multiset rewriting has been extensively used in protocol verification. Adaptations and slight variations of the formalism have been developed [181]. In the state-of-the-art AVISPA project [9], the backends (i.e., the on-the-fly model checker OFMC [19], the constraint-solving based model checker AtSe [190], the SAT-based model checker SAT-MC [58], TA4SP [25]) rely mostly on (multiset) rewriting semantics for protocol executions. Other tools for protocol verification were also based on rewriting semantics, e.g., CASRUL [107]. Moreover, numerous theoretical results [175, 75] about protocol verification were obtained through the use of (multiset) rewriting semantics. To sum up, a large proportion of the trace-based semantics in protocol verification is based on (multiset) rewriting techniques.

2.2.2.3 The Bounded Protocol Model

The formalism in [188] is called the *bounded protocol model* and the protocol-execution semantics it introduces is inspired by [172]. A complete correspondence between the formalism in [188] and multiset rewriting [74, 73] is drawn in [188]. Due to the wide spread use of multiset rewriting based semantics and the correspondence mentioned above, in following chapters we will refer to [74,73,188] as standard, traditional and/or mainstream semantics for multi-session protocol execution. We summarise the main characteristics of the bounded protocol model formalism. We draw the reader's attention to this summary, as we will often refer to the bounded protocol model throughout the thesis, e.g., as part of a proof in Chapter 4.

- any protocol generic *rule* $A \to B : t$ is decomposed into two *actions*:

  — a *send action* $a = A!B\ :\ (M)t$;

  In the above, M is the set of all nonces and short-term keys generated by A in order to obtain the message t at the step where the action a is performed; hence, this set is also denoted $M(a)$ and the term t is denoted $t(a)$, i.e., the term of action a;

  — *receive action* $B?A\ :\ t$;

- a *protocol* is defined by a triple $\mathcal{P} = (\mathcal{S}, \mathcal{C}, \omega)$, where: *1)* $\mathcal{S}$ is a protocol signature consisting of a finite set $\mathcal{A}$ of principals, an at most countable set $\mathcal{K} = \mathcal{K}_0 \cup \mathcal{K}_1$ of keys and an at most countable set $\mathcal{N}$ of nonces; *2)* $\mathcal{C}$ is the set of protocol constants; *3)* ω is the sequence of actions describing $\mathcal{P}$ (also called the *body* of $\mathcal{P}$);

- a *role* is a sequence $\omega|_A$ of actions (where $A \in \mathcal{A}$), i.e., the sequence obtained from the body ω of the protocol $\mathcal{P}$, once all actions which are not executed by A are removed;
- a *substitution* is used to instantiate terms (over the signature); substitutions are homomorphically extended to actions, sequences of actions and, therefore, roles;
- a substitution σ is *suitable for an action* $a{=}AxB : t$ [188] if: *a)* $\sigma(A)$ is substituted to a legitimate participant (i.e., has an outset of keys, etc.); *b)* $\sigma(A) \neq \sigma(B)$; *c)* σ maps distinct nonces from $M(a)$ into distinct value-nonces, distinct short-term keys into distinct value-keys and σ has disjoint ranges for $M(a)$ and $Sub(t(a)) \setminus M(a)$;
- an *event* is a tuple (u, σ, i), denoting the i-th action of the role u instantiated under the suitable substitution σ;
- *analz* and *synth* are standard [163] rules for manipulation/derivation of terms in such symbolic protocol modelling: i.e., $analz(X)$ is the smallest set of terms containing the set X of terms, closed under decomposition rules (e.g., de-pairing, decryption, etc.), whilst $synth(X)$ is the least set of terms containing the set X of terms, closed under composition rules (e.g., pairing, encryptions etc.); then, $\overline{X}$ is used to denote $synth(analz(X))$;
- a *state* s is an indexed set of the form $s = (s_A \,|\, A \text{ principal})$.

This formalism employs a Dolev-Yao intruder model. Furthermore, the formalism operates under the following assumptions:

- every honest principal A is provided with some *secret information* $Secret_A \subseteq \mathcal{K}_0 \cup \mathcal{N}$, not known to the intruder; note that $Secret_A$ does not contain long-term keys;
- the intruder is provided with a set $\mathcal{N}_I \subseteq \mathcal{N}$ of nonces and a set $\mathcal{K}_{0,I} \subseteq \mathcal{K}_0$ of short-term keys. It is assumed that no element in $\mathcal{N}_I \cup \mathcal{K}_{0,I}$ can be generated by honest principals.

With these assumptions, the *initial states* in this formalism are given by $s_0 = (s_{0_A} | A \in \mathcal{A})$, where:

- $s_{0_A} = \mathcal{A} \cup \mathcal{C} \cup \mathcal{K}_A \cup Secret_A$, for any honest principal A;
- $s_{0_I} = \mathcal{A} \cup \mathcal{C} \cup \mathcal{K}_I \cup \mathcal{N}_I \cup \mathcal{K}_{0,I}$.

The *freshness check* denotes a test carried out to ensure that newly generated terms have not appeared previously in the protocols execution. Let s, s' be two states and let e be an event. A *computation step* at the state s resulting in state s', $s[e\rangle s'$, is described as follows:

- if the action of e is a send action, $A!B : (M)t$, then the *enabling conditions* are:
  1. $t \in \overline{s_A \cup M}$ (if *no freshness check* is considered)
  2. $t \in \overline{s_A \cup M}$ and $M \cap Sub(s) = \emptyset$ (if *freshness check* is considered)

  and the *computing rules* are:
  1. $s'_A = s_A \cup M \cup \{t\}$, $s'_I = s_I \cup \{t\}$, $s'_C = s_C$, for all $C \in \mathcal{A} - \{A, I\}$;
- if the action of e is a receive action, $A?B : t$, then the *enabling conditions* are:
  1. $t \in \overline{s_I}$

  and the *computing rules* are:
  1. $s'_A = s_A \cup \{t\}$ and $s'_C = s_C$, for all $C \in \mathcal{A} - \{A\}$

In [188], a *computation* or *run* of a protocol $\mathcal{P}$ is any sequence $s_0[e_1\rangle s_1[\cdots[e_k\rangle s_k$ of computation steps such that:

- $s_{i-1}[e_i\rangle s_i$, for any $1 \leq i \leq k$;
- $\{e_1, \ldots, e_{i-1}\} \supseteq {}^\bullet e_i$, for any $1 \leq i \leq k$, where ${}^\bullet e_i$ is the set of all the events which precede e_i in its underlying role. This means that actions within a role are performed in the order prescribed by the protocol description. For this, a *precedence* relation $\longrightarrow$ is defined and its transitive closure over the event e gives the set ${}^\bullet e$ of events preceding e in its underlying role.

A term $t \in \mathcal{K}_0 \cup \mathcal{N}$ is called *secret* at a state s if $t \in analz(s_A) \setminus analz(s_I)$, for some honest agent A. Secrecy is extended to runs: a run $\xi = e_1 \cdots e_k$ is *leaky* with respect to $T \subseteq \mathcal{K}_0 \cup \mathcal{N}$ if there exists $t \in T$ such that t is secret along some proper prefix of ξ but it is not secret along ξ. The *secrecy problem* (*SP*) is the decision problem of a protocol having leaky runs (with respect to atomic terms in T).

If the set T is uniformly replaced in the above definitions with the set $Secret_A$, then secrecy with respect to initial secrets is obtained. The difference is that the secret term t is not any atomic term, but an initial secret. This form of secrecy is denoted initial secrecy. The *initial secrecy problem* (*ISP*) is the decision problem of a protocol having leaky runs (with respect to initial secrets).

However, both SP and ISP can fall into two different categories: *(I)SP with freshness check* or *I(SP) without freshness check*. Considering or dismissing freshness checks yields respectively different complexity and decidability results of the *(I)SP* problem [188].

In order to relate this formalism with MSR, the *bounded protocol model* is introduced in [188]. Analysing secrecy in MSR under restricted form [73] coincides with verifying secrecy in the bounded protocol model (see pages 702–706 in [188]). A (T, k)-*bounded protocol* has all its messages built with terms drawn from T and all its messages have length at most k, where $T \subseteq \mathcal{T}_0$ is a finite set and $k \geq 1$. Certain parameters that usually lead to undecidability (e.g., number of instantiation, number of events) are found to be upper-bounded in [188] for the class of bounded protocols. For instance, the number of events of a (T, k)-bounded protocol $\mathcal{P}$ is of the order of $\mathcal{O}(2^{poly(size(\mathcal{P}))})$, where $size(\mathcal{P}) = |\omega| + k \log |T|$.

We do not present here the algorithms or the reductions used in [188] to obtain the decidability and complexity results for the (initial) secrecy problem. When needed, we will refer to the algorithms used for deciding (initial) secrecy of bounded protocols in more detail.

2.2.2.4 Applied Pi Calculus

The previous subsections reviewed the trace-based semantics for protocol executions (e.g., the MSR formalism and the bounded protocol model). In turn, in this section we summarise a formalism for protocol verification based on process algebra.

The *applied pi calculus* is a language for describing concurrent processes and their interactions. It was developed explicitly for modelling security protocols in [2] and it can handle more general cryptographic primitives than its similar predecessors (e.g., spi-calculus [3]). We will now summarise the main principles of the applied pi calculus formalism.

Terms on a finite signature are declared by the following grammar:

$$\begin{array}{rll} L, M, N, T, U, V ::= & a, b, c, k, m, n, s, t, r, \ldots & \text{— names} \\ & x, y, z, \ldots & \text{— variables} \\ & g(M1, \ldots, Ml), \ldots & \text{— functions} \end{array}$$

Equational theories are sets of equalities defined on terms, used to symbolise the properties of the cryptographic primitives underlying a protocol. For instance, the well-known properties of symmetric and asymmetric encryption is modelled by the following equational theory:

$$\begin{aligned} sdec(x, senc(x, y)) &= y \\ dec(x, enc(pk(x), y)) &= y \end{aligned}$$

Processes in applied pi are described by the following grammar:

$$\begin{aligned} P, Q, R := {} & 0 && \text{— null process} \\ & P|Q && \text{— parallel composition} \\ & !P && \text{— replication} \\ & \nu x.P && \text{— name restriction} \\ & u(x).P && \text{— message input} \\ & \overline{u}\langle M\rangle.P && \text{— message output} \\ & \text{if } M = N \text{ then } P \text{ else } Q && \text{— conditional} \end{aligned}$$

An *active substitution* $[M/_x]$ is a substitution that replaces x with M uniformly in a process. A process is *closed* when each variable is bound or defined by an active substitution. Processes are identified up to α-renaming [17]. A *context* is a process with a hole. An *evaluation context* is a context whose hole is not under a replication, a conditional, an input, or an output. A context $C[\,]$ *closes process* A when $C[A]$ is closed. The interested reader is referred to [2].

An *operational semantics of applied pi* describing a behaviour of processes that simulates the communication of (partly) substituted terms is presented in [2]. To exemplify, the rule of *NEW-C* is given like $\nu n.\nu w.P \equiv \nu w.\nu n.P$ and it denotes the commutativity of the "new" operator, i.e., ν. The *NEW-PAR* rule stipulates that a ν operator inside a parallel composition bounds over the whole parallel composition if its arguments are not free terms in the process under the composition, i.e., $A|(\nu nB) = \nu n.(A|B)$, where $n \notin freenames(A) \cup freevars(A)$. The *REWRITE* rule denotes that if M and N are equal modulo the equational theory E then substituting either x by M or by N has the same effect, i.e., $\{M/_x\} \equiv \{N/_x\}$, if $M =_E N$. For details on the semantics, refer to [2].

To define communication further, the operational semantics of applied pi introduces the so-called *labelled semantics*. The notation $A \xrightarrow{\alpha} B$ has the following particular instances:

- $A \xrightarrow{c(M)} B$ means that the process A performs an input of the term M from the environment on the channel c, and the resulting process is B;
- $A \xrightarrow{\bar{c}\langle u\rangle} B$ means that the process A outputs the free variable or channel name u and then A becomes ("behaves like") B;
- $A \xrightarrow{\nu u.\bar{c}\langle u\rangle} B$ means that A outputs the variable or channel name u that is restricted in A, and becomes free in B.

The labelled semantics subsumes *structural equivalence* and *internal reduction* [2].

The specifications formalised and studied in applied pi fall into three classes: reachability properties, *correspondence properties* and *equivalence properties*. We will summarise these specifications in the following section, when we explicitly address formalisations of security requirements.

The intruder model adopted in applied pi is usually a Dolev-Yao threat model. However, for some protocols (e.g., e-voting), the applied pi thread-model assumes only passive adversaries.

To carry out protocol verification in applied pi, the following methodology is employed: *1)* the equations to capture the cryptographic primitives are expressed; *2)* the protocol participants are divided into honest and dishonest; *3)* the honest parties are modelled as processes; *4)* each intended security property is specified as a reachability property, as a correspondence property or as an observational equivalence property; *5)* the model is then explored manually or by using `ProVerif`, where possible (i.e., for reachability properties).

`ProVerif` [23] is a tool that can verify applied pi specifications. The tool can analyse an unbounded number of protocol sessions, but it is therefore semi-decidable. That is, it might not terminate, but if and when it halts it gives the correct answer to the decision problem (e.g., secrecy). `ProVerif` can only analyse reachability properties. Hence, it can be used only for those properties specified in applied pi which are reducible to a reachability formula.

2.2.2.5 Other Formalisms and Tools

There are many other formalisms and tools for security protocol verification. In the following, we will briefly recall some of the important ones. `CSP` (*Communicating Sequential Processes*) [174] is one of the formalisms used in security verification and based on process algebra. CSP was extensively used by Lowe to prove protocol insecurity [139, 138, 72]. `Casper` [138] is a software that compiles a high-level protocol description into a `CSP` model of the protocol. `FDR` [72] is a model checker that can verify such a `CSP` model. Developed in the nineties, `Casper` and `FDR` were used successfully to analyse the so-called *Lowe's small systems* (i.e., models for small-size protocol scenarios). The underlying ideas of these methodologies followed Lowe's intuition that if protocol failures exist, they should be exhibited on a small-size instantiation of the protocol (i.e., on a low number of concurrent sessions); later and further practice [138, 9] suggests that this intuition is indeed correct.

Similar size systems were verified using the explicit model checker `Murphi` [153]. `Murphi` can handle a finite number of protocol sessions against reachability properties.

At the other end, `Hermes` [33] is a tool that can handle an unbounded number of protocol sessions, unbounded size of messages, unbounded number of participants, and unbounded number of nonces. It is not based on any of the techniques mentioned so far, but on symbolic *patterns*; these approximate the infinite set of safe messages [33]. Another tool that can handle infinite state-spaces by employing approximation and reduction techniques is `Athena` [180]. A more recent tool that expands on the ideas used in `Athena` is ***`Scyther`*** [61].

2.2.3 Specification of Security Requirements

There is no standard for specifying security requirements. The variations in specifications are due in part to the subtle differences between one protocol and another, even if they belong to the same class (e.g., the class of authentication protocols). Along these lines, we quote Syverson and Cervesato's opinion as expressed in [182]: "There is not a unique definition of authentication that all secure protocols satisfy".

In the first part of this thesis, we are interested in secrecy and authentication requirements. In the second part of the thesis, we focus on anonymity-originated requirements. We will now recall the view that the aforementioned formalisms took on security requirements and on the specification of security requirements.

2.2.3.1 Authentication Goals

Attempts of Standardisation. The only notable attempts to formalise authentication goals are those of Gollman [93] and, later, those of Lowe [137].

Gollman's authentication goals [93] are expressions of different flavours of entity authentication. They are denominated G1 to G3.

• The goal G1 stipulates that the protocol ought to establish a fresh session key, known only to the honest parties in that session and possibly to a trusted party.

• Additionally to G1, the goal G1' requires that the compromising of old sessions keys does not lead to the exposure of new session keys.

• The goal G2 specifies that a key associated with a principal B has to be used in a message received by another principal A in the protocol run, in a response to a challenge issued by A in the form of a nonce or a timestamp.

• The goal G3 enforces that a key associated with a principal B is used during the protocol run, in a response to a challenge issued by another principal A in the form of a nonce or a timestamp. However, A need not receive a message where this key was used.

The goals G1 and G1' refer only to one individual protocol session and do not consider possible interleaving of sessions. The goal G2 is probably the one closest to entity authentication [82] (or to what entity authentication is believed to require). The goal G3 is the weakest of the three.

Lowe's hierarchy for authentication is based on Lowe's observation [137] that all authentication properties are of the form: "A protocol P guarantees property X to initiator A with respect to another principal B if and only if whenever A completes a run of the protocol, apparently with responder B, then a certain requirement ϕ_X holds." In this fashion, by correlating the property X with different requirements ϕ_X, a taxonomy of authentication goals is presented in [137]. We summarise them from the weakest such requirement to the strongest.

• *Aliveness* of B as A's communication partner is a guarantee to A that B has been active at some point prior to the current moment of communication.

• *Weak agreement* of B with A requires that A had some evidence of B being actively engaged in the protocol (e.g., he has announced its public key). However, weak agreement does not require that B has been running the protocol with A, or that it had been A's communication partner.

• *Non-injective agreement* of B in front of A is formulated in [137] as: "B has previously been running the protocol, apparently with A, and B was acting as the responder in his run, and both agree on values of variables in a dataset ds". Note that for non-injective agreement to hold it is possible A and B have been actually engaged in different sessions (and it is only by chance that they have come to agree on information in the dataset ds).

• *Injective agreement* strengthens Lowe's non-injective agreement with the requirement that there is an one-to-one correspondence between the runs where the sharing of data is accomplished.

In [137], Lowe's hierarchy is best pictured by illustrating subtly different attacks, each mounted against one requirement or another.

Most often, the attempts of standardising the specifications for authentication like the ones summarised above had limited impact. Different formalisms have taken different views and approaches to specifying authentication properties. In the following, we summarise the specifications for authentication as they appear in the formalisms previously recalled in this section.

Authentication Properties Formalisations. BAN's approach to specifying authentication goals was protocol dependent. Whilst no standardisation was in place, some BAN authentication goals seem to be more frequent than others (e.g., key-establishment). Most of these goals related to the so-called belief in "goodness" of the keys. Some examples of BAN formulations of goals are: $A\,\texttt{believes}\,(A \overset{k}{\longleftrightarrow} B)$, $B\,\texttt{believes}\,(A \overset{k}{\longleftrightarrow} B)$, $A\,\texttt{believes}\,B\,\texttt{believes}\,(A \overset{k}{\longleftrightarrow} B)$, $B\,\texttt{believes}\,A\,\texttt{believes}\,(A \overset{k}{\longleftrightarrow} B)$.

The GNY, AT and SVO logics were concerned with correcting BAN's semantic flaws. As such, they did not increase the expressivity of BAN. Hence, to a large extent, the authentication goals remained the same in these successors of the BAN logic.

In turn, the VO logic did enrich the language of BAN and provided some more advanced authentication specifications. The approach to the VO specification of authentication goals distinguished the following formalisations [191]: *ping authentication*, *entity authentication*, *secure key* *establishment*, *mutual understanding of shared keys*, *key freshness*, *key confirmation.* For instance, ping authentication implied a notion of aliveness of the communication partner, by specifying that $A\,\mathtt{believes}\,B\,\mathtt{says}\,X$. Entity authentication was specified as a stronger property: B has previously generated a message-part Y and now B believes that A has used Y in one of her messages. This was denoted as $B\,\mathtt{believes}\,A\,\mathtt{says}\,F(X, Y_B) \wedge \mathtt{fresh}(Y_B)$ and it meant that A was authenticated to B as a legitimate holder of Y. The VO specification of mutual understanding carried an implicit nesting of beliefs: $B\,\mathtt{believes}\,A\,\mathtt{says}\,(A \overset{k-}{\longleftrightarrow} B)$. In VO, this was intended as: B believes that A has acquired k as a communication-key between A and B (whereas B might not have k).

In the multiset rewriting formalism, the authentication goals are expressed as reachability properties. MSR states are annotated with predicates and it is to be checked whether the execution terminates in states with certain tags (e.g., a tag $B\,\mathrm{TERMINATED}\,A\,Na$ means that a *B-role* has terminated its session with some values for A and Na).

In the bounded protocol model, secrecy is expressed in terms of existence of leaky runs.

In applied pi calculus, reachability and *correspondence properties* are used to analyse authentication properties. The idea of correspondence properties is that, by annotating processes with parametrised events $\overline{f}\langle M\rangle$, the relationships between the order of events and their parametrisation M can be studied. More formally, a correspondence property is a specification of the form $\overline{f}\langle M\rangle \rightsquigarrow \overline{g}\langle N\rangle$; it stipulates that if the event f has been executed then the event g must have been previously executed and any relationship between the event parameters must hold.

Having reviewed the formalisations of secrecy, authentication and key-establishment goals, we proceed to survey the formalisation of other types of security goals of interest to this thesis.

2.2.3.2 Anonymity Related Goals

Anonymity-related Formalisations. In applied pi calculus, *equivalence properties* are used to verify anonymity-like properties. Equivalence properties have been introduced in applied pi at a later stage to capture those properties that cannot be specified as reachability or correspondence formulae. Equivalence properties are based on the idea of indistinguishability between two processes. An example of such properties is the unlinkability-like requirement between a voter and its vote in e-voting environments. The notation $A \Downarrow c$ abbreviates that the process A can evolve to a process that is able to send a message on a channel c (i.e., when $A \rightarrow C[\overline{c}\langle M\rangle.P]$ for some term M and some evaluation context $C[\,]$ that does not bind c). In applied pi, observational equivalence is the largest symmetric relation R closed under extended processes with the same domain, such that ARB implies: *1)* if $A \Downarrow c$, then $B \Downarrow c$; *2)* if $A \rightarrow A'$ then, for some B', it is the case that $B \rightarrow^{*} B'$ and $A'RB'$; *3)* $C[A]RC[B]$ for all closing evaluation contexts $C[\,]$.

Other anonymity-related formalisations are those introduced in epistemic frameworks. The first to advance such formalisations are Halpern and O'Neill in [97]. They use a multiagent system formalism to reason about secrecy as a property of lack of knowledge. In [97], $\mathcal{I}$ is an interpreted system and $\tau(i, a)$ is used to mean that "agent i has performed action a or it will perform it in the future". Action a performed by agent i is *minimally anonymous* to agent j if $\mathcal{I} \models \neg K_j \tau(i, a)$ (hence, agent j cannot be sure that agent i performed the action a). Action a performed by agent i is *totally anonymous* to agent j if $\mathcal{I} \models \tau(i, a) \rightarrow \bigwedge_{i' \in Ag \setminus \{j\}} P_j \tau(i', a)$ (hence, agent j considers it possible that any agent could have done action a)[5]. Total anonymity can be restricted to *anonymity up to a set* $A \subsetneq Ag$ or to k*-anonymity*, anonymity between a group of k agents. These notions of anonymity are used in [97] in the context of security settings, to formalise secrecy properties.

Several approaches used the formalisation in [97] to specify anonymity-related properties. The notable ones are [90,78,142]. A taxonomy of formalisations referring to the anonymity or identifiability of agents, actions or agents performing actions is presented in [189]. To distinguish between each of these cases, new terminology was introduced in [189]; e.g., based on Halpern's seminal formalisation [97] of secrecy, [189] uses *onymity* to refer to knowledge of agents' names, *identity* to refer to knowledge of actions, *anonymity* to refer to the lack of knowledge about the agents' names and *privacy* to refer to the lack of knowledge about the actions performed. At the intersection of such schema, [189] places *role interchangeability* and *role non-interchangeability* specifications. Like in [97], this taxonomy classifies these anonymity-like properties into partial, total or up to some range. The hierarchy in [189] is best summarised by an illustration given in [189] and reported here in Figure 2.1.

In Section 2.2.3 we have reviewed the expression of security goals in various formalisms for protocol verification. We now proceed to summarise of the high-level languages used for describing

[5]The formula $P_j\varphi$ stands for $\neg K_j \neg\varphi$, for some $\varphi \in Form$ and j some agent.

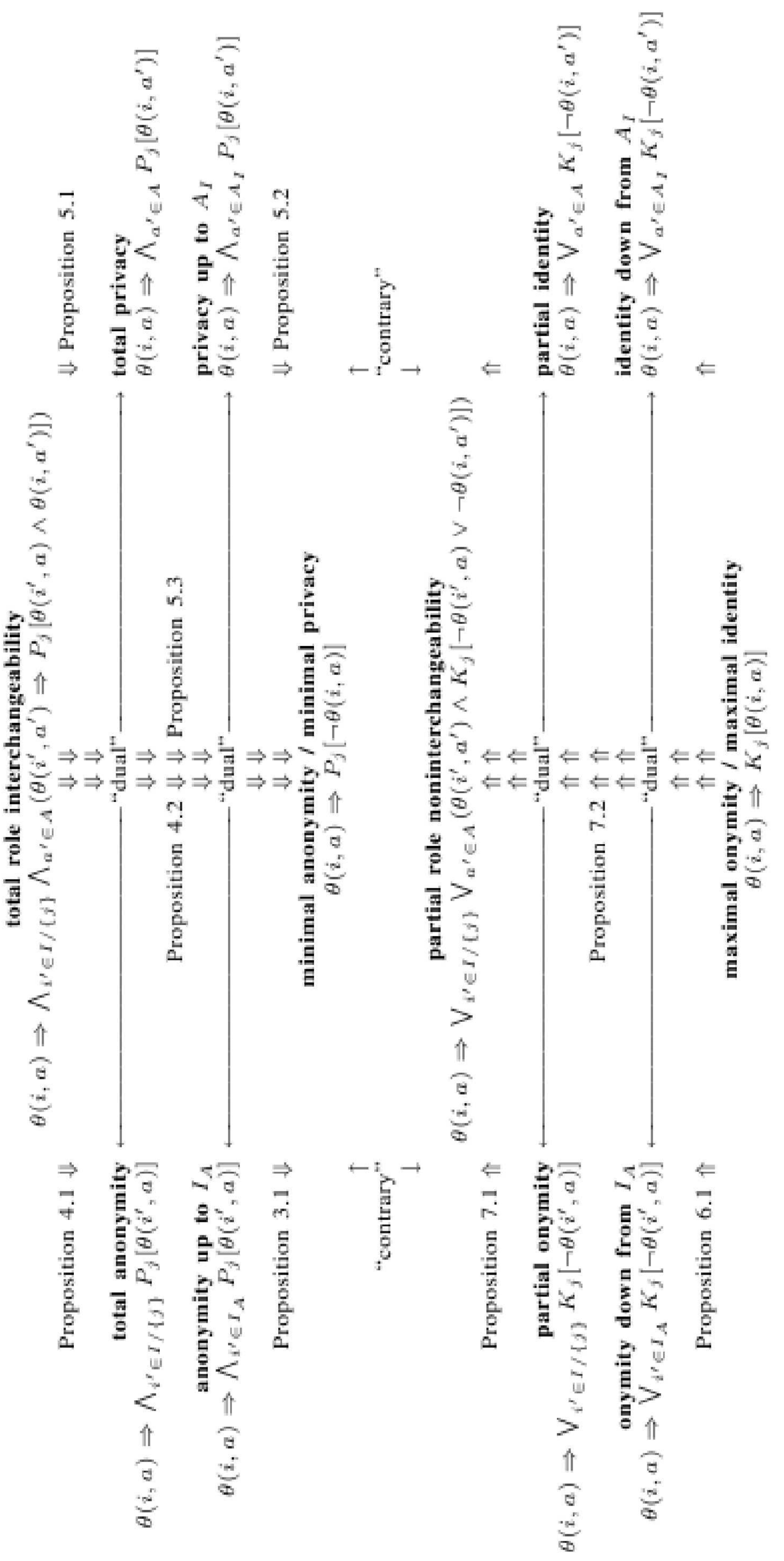

total role interchangeability
$\theta(i,a) \Rightarrow \bigwedge_{i' \in I/\{j\}} \bigwedge_{a' \in A}(\theta(i',a') \Rightarrow P_j[\theta(i',a) \wedge \theta(i,a')])$
Proposition 4.1 ⇓
⇓ Proposition 5.1
total anonymity
$\theta(i,a) \Rightarrow \bigwedge_{i' \in I/\{j\}} P_j[\theta(i',a)]$
"dual"
total privacy
$\theta(i,a) \Rightarrow \bigwedge_{a' \in A} P_j[\theta(i,a')]$
Proposition 4.2 ⇓ ⇓ Proposition 5.3
anonymity up to I_A
$\theta(i,a) \Rightarrow \bigwedge_{i' \in I_A} P_j[\theta(i',a)]$
"dual"
privacy up to A_I
$\theta(i,a) \Rightarrow \bigwedge_{a' \in A_I} P_j[\theta(i,a')]$
Proposition 3.1 ⇓
⇓ Proposition 5.2
minimal anonymity / minimal privacy
$\theta(i,a) \Rightarrow P_j[\neg\theta(i,a)]$
"contrary"
"contrary"
partial role noninterchangeability
$\theta(i,a) \Rightarrow \bigvee_{i' \in I/\{j\}} \bigvee_{a' \in A}(\theta(i',a') \wedge K_j[\neg\theta(i',a) \vee \neg\theta(i,a')])$
Proposition 7.1 ⇑
partial onymity
$\theta(i,a) \Rightarrow \bigvee_{i' \in I/\{j\}} K_j[\neg\theta(i',a)]$
"dual"
partial identity
$\theta(i,a) \Rightarrow \bigvee_{a' \in A} K_j[\neg\theta(i,a')]$
Proposition 7.2 ⇑
onymity down from I_A
$\theta(i,a) \Rightarrow \bigvee_{i' \in I_A} K_j[\neg\theta(i',a)]$
"dual"
identity down from A_I
$\theta(i,a) \Rightarrow \bigvee_{a' \in A_I} K_j[\neg\theta(i,a')]$
Proposition 6.1 ⇑
maximal onymity / maximal identity
$\theta(i,a) \Rightarrow K_j[\theta(i,a)]$

Figure 2.1 The Taxonomy in [189] of Epistemic Specifications of Identity and Action Privacy

security protocols.

2.2.4 Protocol Description Languages

In order to gain more expressivity than in the simple Alice&Bob notation, protocol description languages have been developed. With them, protocols can be described more accurately (e.g., details about the data-types to be used and the requirements intended at the design-time can be specified). Many compilers have been built to translate such high-level languages into low-level languages mirroring one semantics for protocol verification or another (e.g., `CASPER` is used to transform a high-level language into `CSP` [139], `HLPSL` [157] is compiled to `IF` in [167], `CAPSL` is compiled to `CIL` in [64], etc.). With these languages and their compilers, a move towards the software-engineering of protocol analysis is made.

In the following, we will survey two such high level protocol specification languages.

`CAPSL` (*Common Authentication Protocol Specification Language*) is a language developed by Millen et al., aimed at describing authentication and key-establishment protocols. The syntax of `CAPSL` and some explanations on the denotations implied are presented in [64]. For the interested reader, parts of these have been explained further in [66]. Another source for information about `CAPSL` is [151].

A *`CAPSL` protocol description* (usually) contains three distinct parts: a type specification (`typespec`), a protocol specification (`protocol`) and an environment specification (`environment`).

Declarations of abstract datatype used in the protocol are made in the type specification section. Commonplace datatypes (e.g., keys, nonces, etc.) are provided in a `CAPSL` standard package called *prelude*. Thus, the `typespec` section is mainly devoted to introducing user-defined types and signatures on them.

Furthermore, equations on these signatures can be defined (under the entry *axiom*). Equations are equalities on terms, as we recalled within the summary of the applied pi calculus formalism (i.e., page 45). They stipulate the equational theory underlying the protocol. A security protocol for which the description relies on a set of axioms formed with the underlying cryptographic primitives is denoted as an *equationally specified protocol* [178, 63]. We specified the equational theory underlying the Gong protocol [94] in `CAPSL`. We report our specification in Example 2.2.4.

Example 2.2.4 (The Type-Specification CAPSL Section for the Gong Protocol)

```
TYPESPEC GongSpec;
FUNCTIONS
    k(Skey): Skey;
    ha(Skey): Atom;
    hb(Skey): Atom, PRIVATE;
    recon(Skey,Atom,Atom): Skey;
VARIABLES
    Z: Skey;
AXIOMS
    recon(k(Z),hb(Z),ha(Z)) = Z;
END;
```

The Gong protocol uses one-way functions (denoted below hb, ha, k) and xor, but no encryptions or decryptions. The equation used to retrieve the message encoded is given in Example 2.2.4 under the entry `AXIOMS`.

The `CAPSL PRELUDE` also facilitates some denotations of properties and function symbols on these datatypes. In Example 2.2.4, the function hb has the modifier `PRIVATE`, hence its scope is restricted to principals that hold the key Z. Another example of a modifier to be used under the section of `AXIOMS` is `INVERT`. A term or a function is tagged with `INVERT` if it is *invertable*; for instance, a term $\{x\}_K$ is invertable in the sense that a protocol role can retrieve x if it holds the term $\{x\}_K$, plus the (inverse of the) key K. The `CAPSL PRELUDE` package also provides certain simplified notations for common cryptographic primitives (e.g., in the `CAPSL`, $\{\cdot\}$ is used to denote encryption).

In the `CAPSL PROTOCOL` specification section, further definition or specialisation of equations and terms is possible. As expected, this section describes the protocol, from the actual communication to its goals. In Example 2.2.5, we report the rest of our `CAPSL` description for the Gong protocol.

Example 2.2.5 (The Protocol CAPSL Section for the Gong Protocol)

```
PROTOCOL Gong;
IMPORTS GongSpec;
VARIABLES
     A, B: Client;
     S: Server;
     Na, Nb, Ns: Nonce;
     Pa, Pb: Skey;
     Kh, K: Skey;
DENOTES
     Pa = ssk(S,A): S;
     Pa = csk(A): A;
     Pb = ssk(S,B): S;
     Pb = csk(B): B;
     Kh = sha({Ns,Na,B,Pa}): A,S;
ASSUMPTIONS
     HOLDS A: B;
     HOLDS B: S;
MESSAGES
     1. A -> B: A, B, Na;
     2. B -> S: A, B, Na, Nb;
     3. S -> B: Ns, xor(sha({Ns,Nb,A,Pb}),Kh), sha({Kh,Pb});
           K = k(Kh);
     4. B -> A: Ns, hb(Kh);
           K = k(Kh);
     5. A -> B: ha(Kh);
GOALS
    SECRET K;
    PRECEDES A: B | Nb, K;
    PRECEDES B: A | Na, K;
```

```
END;
```

Under the section DENOTES, we defined terms. The role that holds these terms is also specified. In Example 2.2.5, the first line under DENOTES expresses that Pa is a shared key between the *A-role* and the *S-role*, but it is held by the *S-role*. Protocol goals (under the CAPSL section of GOALS) are described by assertions, each of which corresponds to some security requirement.

The CAPSL grammar for goals allows for the usual secrecy and agreement assertions, but also for knowledge, beliefs, proofs and assumption assertions. However, the GOALS grammar in CAPSL is tailored to key-establishment and authentication protocols. For that reason, much as it is possible to express some goal pertaining to knowledge, the CAPSL grammar does not allow the expression of more advanced requirements, e.g., lack of knowledge. In other words, negations cannot appear but as inequalities between terms. The exact CAPSL grammar for goals can be found at [151]. For instance, in the aforementioned simple assertions the most important keywords used are SECRET and PRECEDES. As Example 2.2.5 shows, the Gong protocol requires the secrecy of the key K, as well as the agreement between the *A-role* and the *B-role* participants upon the data Na, Nb and K. A temporal aspect, as in Lowe's hierarchy of agreement, is implied by the use of PRECEDES: e.g., B holds Nb before A does and eventually they agree on it. The assertions of knowledge and belief (expressed through keywords like KNOWS, BELIEVES, ASSUME) are used to support BAN-like goals. Nesting of different assertions is used in order to render requirements like chain-authentication, etc.

Most of the denotations associated to CAPSL keywords and descriptions are given in the context of backends for CAPSL (e.g., NRL [144], MAUDE-based verification [181], etc.) and are based on a rewriting semantics. The intermediate language for CAPSL is called CIL (CAPSL Intermediate Language) and it is introduced in [66]. A multiset rewriting semantics is implicitly attributed to CIL [66]. A translator for CAPSL to CIL is available at [151]. In the following, we give just a snapshot of the CIL file for the Gong protocol which we obtained by using the translator available at [151].

Example 2.2.6 (An Excerpt of the CIL file for the Gong Protocol)

```
rule (
     facts ( state ( roleA , 3 , terms ( A , B , Na , Ns , Pa ) ) ),
        ids ( ) ,
        facts ( state ( roleA , 4 , terms ( A , B , Na , Ns , Pa , sha
   ( cat ( Ns , cat ( Na , cat ( B , Pa ) ) ) ) ) ) ) )
     ) ,
      rule (
         facts ( state ( roleA , 4 , terms ( A , B , Na , Ns , Pa , Kh )
   ) ) ,
           ids ( ) ,
          facts ( state ( roleA , 5 , terms ( A , B , Na , Ns , Pa , Kh ,
   k ( Kh ) ) ) )
     ) ,
```

The snapshot of code in Example 2.2.6 shows that protocol roles in `CIL` are expressions of MSR restricted-roles (as described in Section 2.2.2.2). In particular, the excerpt in Example 2.2.6 depicts how, at step 4, the role A follows the protocol description/signature and generates $sha(cat(Ns, cat(Na, cat(B, Pa))))$ which is then rewritten to Kh in the next *rule*, etc.

`HLPSL` (*High-Level Protocol Specification Language*) [157] is a protocol description language similar to `CAPSL`. `HLPSL` was developed in the AVISPA project [9]. It also enjoys an intermediate language, which is called `IF` (*Intermediate Format*) and is based on a rewriting semantics. Each back-end of AVISPA (e.g., `OFMC` [19], `AtSe` [190], `SAT-MC` [58], `TA4SP` [25]) translates an `IF` file into a lower-level specification and carries out protocol verification. `HLPSL` provides default basic types for specifying protocols (e.g., agent, channel, boolean, integer, text, message, public key, symmetric keys), type constructors for user-defined datatypes, goal-specification syntax, etc. An `HLPSL` description is more detailed than a `CAPSL` one. An `HLPSL` protocol description is role-oriented and it depicts a protocol scenario (i.e., it includes role instantiation). For illustration purposes, we give below an excerpt of NSPK described in `HLSPL`:

Example 2.2.7 (An Excerpt of The `HLPSL` Description for the NSPK Protocol)

```
role alice (A, B: agent,
           Ka, Kb: public_key,
           SND, RCV: channel (dy))
```

```
played_by A def=

  local State : nat,
        Na, Nb: text
  init State := 0
  transition
    0.  State  = 0 /\ RCV(start) =|>
               State':= 2 /\ Na' := new() /\ SND({Na'.A}_Kb)
                   /\ secret(Na',na,{A,B})
                   /\ witness(A,B,bob_alice_na,Na')

    2.  State  = 2 /\ RCV({Na.Nb'}_Ka) =|>
               State':= 4 /\ SND({Nb'}_Kb)
                    /\ request(A,B,alice_bob_nb,Nb')
```

Specification languages which are not security-specialised have been sometimes succesfully used for describing cryptographic protocols. An example of this kind is CASL (*Common Algebraic Specification Language*) [14]. In these cases too, compilers to some lower level security semantics have been employed.

In Section 2.2.4 we summarised the characteristics of several high-level languages for security protocol description. Overall, in Section 2.2 we presented a synopsis of security protocols, a comprehensive list of formalisms employed in formal verification of cryptographic protocols, the formalisation of security requirements and a summary of high-level specifications of security protocols.

2.3 Epistemic Logic and Cryptography

In this section we will summarise the relations between the analysis of security protocols and epistemic logic. These include advances in the study of cryptographic indistinguishability, notions of computational soundness of formal cryptography analysis and the first steps towards the use of temporal-epistemic logic in automatic verification of security protocols.

Cryptographic Indistinguishability. We mentioned in Section 2.1 that the semantics of the knowledge operator K relies on some equivalence relation over the local states of the agents, called the indistinguishability relation. It is easy to see that the simple equality relation is not always adequate for modelling cryptography (e.g., two encrypted messages of an agent cannot be symbolically distinguished if the agent does not possess the respective decryption keys). A correct notion of indistinguishability is important in modelling security protocols, especially in the context of certain security requirements. As such, vote-privacy requires that every vote remains private; by observing all runs of the voting process, the attacker cannot single out a run in which he can tell apart the fact that *alice* voting a from *bob* voting b, even if he applied the cryptographic equations of the protocol.

The perfect cryptography assumption[6] implies that there is no cryptanalysis encoded in formal or symbolic models of protocols (e.g, agent i does not store or compare the lengths of the messages, etc.).

In AT [5], two local states are indistinguishable if their content cannot be differentiated via the decryptions possible at those states. This only referred to symmetric decryption. Thus, a local state s_i containing only the key k and the term $senc(a, k)$ for the symmetric encryption of the vote a with the key k is i-distinguishable in AT from a state s'_i containing only the key k and the term $senc(b, k)$, i.e., $senc(a, k)$ and $senc(b, k)$ both represent encrypted bitstrings, opaque to agent i.

In later years, the notion of AT-like indistinguishability has been further expanded to be applied to asymmetric cryptography as well. The SVO logic (recalled in Section 2.2) was one of the formalisms that supported indistinguishability for asymmetric encryption [184]. The latter form of indistinguishability was modelled in [60, 193] as well.

In applied pi calculus [2], reviewed in Section 2.2, the notion of indistinguishability is intuitively modelled by the notion of *static equivalence*. A *frame* σ is an extended applied pi calculus process built up from 0 and active substitutions of the form $[M/_x]$ by parallel composition and restriction. The domain $dom(\sigma)$ of a frame σ is the set of variables that it exports (those variables x for which the frame contains a substitution $[M/_x]$ not under the restriction of x). Intuitively, the frame σ_A corresponding to a process A accounts for the "static knowledge" exhibited by A to the environment (i.e., a simpler notion of knowledge in which the operational semantics of the process A is abstracted).

[6]See [71] or page 37 for a summary of the perfect cryptography assumption.

Two terms M and N are *equal in the frame* σ, $(M =_E N)\sigma$, if and only if there exists n and a substitution θ such that σ is $\nu n.\theta$, $\theta(M) = \theta(N)$, and $n \cap (fn(M) \cup fn(N)) = \emptyset$. Two frames are *statically equivalent* if they have the same domains and they exhibit the same equalities of terms modulo the underlying equational theory.

In [164] Pucella et al. propose a different kind of indistinguishability. In a nutshell, two states $g, g' \in G$ are i-indistinguishable if the following holds: when agent i applies an algorithm A_i to each of the states, the results respectively obtained are the same, i.e., $g \sim_i g'$ if and only if $A_i(g) = A_i(g')$. In a model extending the interpreted system formalism, a knowledge modality based on this indistinguishability relation and called *deductive algorithmic knowledge* is put forward [164].

Other indistinguishability relations and consequent knowledge modalities (partly) related to cryptographic modelling exist. Agent i is *aware* of ϕ [134] if the formula ϕ is within a dedicated set of facts. In [134], agent i *explicitly knows* ϕ if i knows it in the standard sense and i is aware of ϕ. Explicit modality can therefore be viewed as a simplified version of deductive algorithmic knowledge. A procedure computing such an underlying indistinguishability relation of explicit knowledge is implemented in the `MCMAS-X` [130] branch of the `MCMAS` model checker.

Computational Soundness of Cryptographic Formalisation. Computational soundness results for the formalisation of cryptography in symbolic models started with the work of Abadi and Rogaway in [4]. A symbolic verification methodology is computationally sound if it enjoys a proof that the guarantees it offers hold in a corresponding computational approach as well (i.e., if some security guarantee holds for abstract-term formalisations, then it also holds for the bitstring formalisation). The work of Abadi and Rogaway only treated the case of symmetric encryption and that of passive adversaries. Later, this was extended for asymmetric encryption and for stronger adversary models. Recently, in [57], Comon-Lundh et al. have proven computational soundness for observational equivalence in applied pi calculus.

Working towards the computational soundness of epistemic-based cryptographic modellings, [54] draws the attention to the *local omniscience problem* [160]. This problem is inherent in any Kripke semantics (hence, it is invariant with the indistinguishability relation). In a nutshell, the logical omniscience problem refers to the fact that agents know all properties of their local states. For instance, the validity $\models (\{m\}_k \; contains \; m)$ implies that $\models K_i(\{m\}_k \; contains \; m)$, for some agent i in the system. To counteract this issue, Cohen underlines the difference between considering the indistinguishability relation in a *de dicto* versus in a *de re* sense [54]. Let the local states be understood *de re* [36]. In this case, if $A \; receives \, enc(m, k)$ then (by $S5$) she knows *what* she received, i.e., $K_A A \; receives \, enc(m, k)$. Now, let the local states adhere to a *de dicto* interpretation. Then, the latter knowledge of A is not the case (i.e., the implication $A \; receives \, enc(m, k) \rightarrow K_A A \; receives \, enc(m, k)$ cannot be valid for all ground terms of the form $enc(m, k)$). Following these ideas, Cohen proves completeness of a BAN-like logic for cryptography in a semantics that differentiates the interpretation de dicto from the interpretation de re (i.e., he uses a counterpart semantics [123] framework).

Towards Temporal-Epistemic Verification of Security. A recent methodology of verifying cryptographic protocols against temporal-epistemic formulations of security requirements is presented in [125]. The authors use a bounded model checking framework to verify security protocols specified as interpreted systems specialised with security-related features. The framework is called *LDYIS* (*Lazy Dolev-Yao Interpreted Systems*).

We concisely give the syntax and the semantics of an LDYIS as per [125]. In an LDYIS, a collection of sets is introduced to define the model:

- a finite set Ag of agents, containing a special agent ι representing the intruder;
- an ordered set $\mathcal{N}^f$ of fresh nonces, a set $\mathcal{N}^o$ of nonces already used, a set $\mathcal{N} = \mathcal{N}^f \bigcup \mathcal{N}^o$ of nonces;
- an indexed set $\mathcal{K} = (K_i \,|\, i \in Ag)$ of keys;
- a set $\mathcal{K}_i$ of keys known to agent i, a set $\mathcal{N}_i^f$ of nonces to be freshly generated by agent i.

To enrich the specification, other mathematical objects are introduced, e.g., $@_i$ denoting the address of some agent i and id_i the number of parallel sessions that agent i carries simultaneously. A message msg of the protocol is defined by the following grammar:

$$i \,|\, I \,|\, n \,|\, N \,|\, k \,|\, K \,|\, \{msg\}_k \,|\, \{msg\}_K \,|\, msg \cdot msg.$$

In order to depict the actual communication of messages, [125] introduces the concept of letter; a *letter* lt is given by the tuple $lt = ((@_s, @_r), msg)$ and it denotes that a message msg is sent by agent s to agent r. A local state l_i of agent i is given by the tuple $l_i = \left(Ag_i, \mathcal{N}_i^o, \mathcal{N}_i^f, \mathcal{K}_i, id_i, lt_i\right)$, where all components follow the explanations above and lt_i is defined such that $lt_i \subset (lt, id)^+$. The notation $(lt, id)^+$ denotes the recording of the sequence of the letters that an agent has received or sent, together with the identifier of the session in which he did so.

The *LDYIS* model for a security protocol Pr is the tuple $M_{Pr} = (G, g^0, \Pi, \sim_1, \sim_2, \ldots, \sim_n, V)$, where $g^0 \in \prod_{i=1}^n L_i$ is an initial global state of the system, G is the set of global states reachable from g^0, $\sim_i \subseteq G \times G$ is an epistemic relation for agent i with $g \sim_i g'$ if and only if $l_i(g) = l_i(g')$, $V : G \times PV \rightarrow \{true, false\}$ is a valuation-function of propositional variables over states and $\Pi = \bigcup_{g \in G}, \Pi(g)$ is the set of all paths starting at g which are compliant with the *Lazy Dolev-Yao conditions* reported below.

In [125], the intruder follows a Dolev-Yao thread model. However, to diminish the state-space implied by this, the *lazy Dolev-Yao conditions* are introduced. These follow a *matched send-receive* idea: the intruder honestly or maliciously plays each step of the protocol as a sender, matched upon the corresponding step of the receiver. Consequently, the LDYIS intruder theory is described by the following set of rules: *honest-send-i-A* $\rightarrow B$, $fake$-$send$-i-$\iota(A) \rightarrow B$[7] and ι-$forward$. It is to be noted that the lazy Dolev-Yao conditions in LDYIS yield a semantics different from the *lazy intruders* in [18, 19], i.e., likened to the concept of laziness in functional programming.

The LDYIS formalism models a type of secure-channels (i.e., the matched send-received semantics and the use of the sender's address in letters does not allow spoofing [81]). In [125], the lazy Dolev-Yao rules are depicted at the level of desired behaviour of the agents, i.e., the systematic integration with a protocol theory is left open. The methodology is applied to a manually analysed case-study of the NSPK protocol, i.e., the generalisation and automation of the principles employed are left open.

Other epistemic-based approaches to security protocol verification are present in [78, 90]. An electronic voting protocol is modelled in an ad-hoc manner using dynamic epistemic logic [148]. The indistinguishability relation used is the simple equality relation. Details about the modelling or the performance of the analysis are scarce.

The formalisation in [90] is dedicated to several types of protocols (i.e., onion routing [92],

[7]In the LDYIS semantics, $\iota(A)$ designates the impersonation of agent A by the intruder.

crowds [192]). The logic language in [90] is based on epistemic logic, whereas the temporal aspect is embedded in the protocol theory. The semantics adopted in [90] is that of distributed programs [80]. The actual analysis of the protocol is manual, but it is not presented in detail.

In this section we summarised the relations between the analysis of security protocols and epistemic logic, e.g., notions of cryptographic indistinguishability and computational soundness of formal cryptography analysis, as well as the first steps towards the use of temporal-epistemic logic in verification of security protocols.

In this chapter we have summarised aspects of temporal-epistemic logics, security protocol verification and the use of logics of knowledge in the analysis of cryptography. All these comprise essentially the notions that will be used in the following sections. Also, our review insisted on the formalisms and methodologies that relate to the work in this thesis. In the last chapter, these relations between existing state-of-the-art work and this thesis will been explored in depth.

Chapter 3

Multiagent System Models for Receiver-Transparent Security Protocols

Motto: "All our work, our whole life is a matter of semantics..."
(Felix Frankfurter)

In this chapter we present a multiagent system formalism for multi-session executions of authentication and key-establishment protocols. The presentation is oriented towards a systematic methodology of generating `ISPL` multiagent system specifications from `CAPSL` high-level protocol descriptions. Aspects of this formalism have been presented in [27, 26]. In later chapters we will prove that the proposed multiagent system semantics is homomorphic with a standard semantics for security protocols.

Section 3.1 formalises `CAPSL` protocol descriptions through many-sorted signatures and it uses this formalisation to introduce the notions of *receiver-transparent* and *receiver-opaque* protocols. Protocol execution scenarios are captured through substitutions of terms defined on the signatures. Section 3.2 lays the foundations for the formalisation of `CAPSL` goals and it introduces the notions of *atomic* and *complex* goals. Section 3.3 presents a full MAS model of multi-session executions of `CAPSL`-described receiver-transparent protocols. A taxonomy of temporal-epistemic formulae is systematically given to express each atomic and/or complex `CAPSL` goal in the model. The chapter concludes with an evaluation of the methodology.

3.1 A Formalisation of `CAPSL`-Described Security Protocols

In Section 2.2.1 we summarised notions of high-level protocol descriptions (e.g., `CAPSL` descriptions). We also showed (in Section 2.2.4, Section 2.2.2.2 and Section 2.2.2.3) that many approaches formalise such a high-level description through a protocol signature. We are going to give our formalisation of a `CAPSL` protocol description also through a (many-sorted) signature. This signature is of the kind previously summarised, used in [188, 175, 73]. Whilst closest to [188], the signature we introduce models precisely a `CAPSL` description (i.e., not a description expressed in a generic high-level language). The literature on `CAPSL` [66, 151, 64] does not provide any explicit formalisation of a `CAPSL` description. We follow closely the `CAPSL` abstract grammar in advancing our formalisation of the `CAPSL` descriptions.

Definition 3.1.1 (Protocol Signature) *A* protocol signature *is a triple $\mathcal{S} = (\mathcal{A}, \mathcal{K}, \mathcal{N})$, where $\mathcal{A}$ is a finite set of principals, $\mathcal{K}$ is a countable set of keys and $\mathcal{N}$ is a countable set of nonces.*

We further describe S as follows:

- $\mathcal{A} = Ho \cup \{I\}$, where Ho is the set of honest principals and I symbolises the intruder;
- $\mathcal{K} = \mathcal{K}_0 \cup \mathcal{K}_1$, where $\mathcal{K}_0$ represents a set of short-term keys and $\mathcal{K}_1$ a set of long-term keys. Long-term keys are those held/shared by participants prior to engaging in the protocol runs. Every honest principal A is associated with a set of long-term keys, denoted $\mathcal{K}_A$. More precisely, long term-keys can be asymmetric encryption/decryption keys and symmetric keys, i.e., $\mathcal{K}_1 = \bigcup_{A \in \mathcal{A}} \{(K_A^e, K_A^d)\} \cup \{K_{AB} | A \in \mathcal{A}, B \in \mathcal{A}\}$. The intruder is also associated with such sets, given that he can engage in the protocol as an honest principal.

The set $\mathcal{T}_0 = \mathcal{A} \cup \mathcal{N} \cup \mathcal{K}$ denotes the *atomic terms*. On $\mathcal{S}$, we define a function symbol *Sort* to return the actual sort of an atomic term, i.e., one of elements in $\{\mathcal{A}, \mathcal{N}, \mathcal{K}\}$.

Over the set of atomic terms $\mathcal{T}_0$, the set $\mathcal{T}$ of (non-atomic) *terms* is defined inductively using function symbols in $\mathcal{S}$. We consider symbols for pairing and encryption. Other cryptographic primitives could be modelled in a signature for `CAPSL` (as following chapters will show). We allow only encryption with atomic keys[1]. The notation $Sub(t)$ denotes the set of subterms of a given term t and it is inductively defined on the structure of t, as usual. We define more specialised classes of subterms as follows.

The set $ESub(t) = \{t' \in Sub(t) \,|\, (\exists t'' \in \mathcal{T})(\exists K \in \mathcal{K})(t' = \{t''\}_K)\}$ is the set of encrypted subterms of t. By $OSub(t)$ we denote the ordered multiset of atomic subterms of t, i.e., the list of atomic terms that are subterms of t, taken in the order in which they appear.

We assume that the encryption key is the last atomic subterm of a term, i.e., encrypted messages are always of the form $\{t\}_k$, for some $t \in \mathcal{T}$, $k \in \mathcal{K}$. Such an assumption is reasonable, as any form of encryption can be symbolically rewritten to follow this pattern [102]. We also introduce the notation $OSub(t)_i$ to denote the i-th element in $OSub(t)$ (i.e., if $OSub(t) = \{t_1, \ldots, t_n\}$ then $OSub(t)_i = t_i$, for any $i = \overline{1,n}$). By $last(t)$ we denote the last atomic subterm of t, i.e., the last to appear in $OSub(t)$. Then, the encryption key within some symbolic encrypted term t is denoted as $OSub(t)_{last(t)}$.

The *length* of a term t, $|t|$, is inductively expressed by: $\begin{cases} |t| = 1, \text{ for } t \in \mathcal{T}_0; \\ |(t_1, t_2)| = |t_1| + |t_2| + 1, \text{ for any } t_1,\, t_2 \in \mathcal{T}; \\ |\{t\}_k| = |t| + 2, \text{ for any } t \in \mathcal{T} \text{ and } k \in \mathcal{K}. \end{cases}$

We can now outline how the protocol signature above partially formalises a `CAPSL` description:

— atomic terms $\mathcal{A}$, $\mathcal{K}$, $\mathcal{N}$ correspond to the `VARIABLES` section; $\mathcal{K}_A$ together with $\mathcal{K}_{AB}$ for some $A, B \in \mathcal{A}$ corresponds to part of the `DENOTES` section of a `CAPSL` description;

— $\mathcal{T}$, $Sub(\mathcal{T})$, $OSub(\mathcal{T})$ and $ESub(\mathcal{T})$ describe the `MESSAGES` section of a `CAPSL` description.

We further formalise a `CAPSL` description by adding some more sorts and function symbols to the signature $\mathcal{S}$; additional terms and sets of terms will emerge. These are described as follows:

• *Steps* – a sort denoting the protocol steps (as per the `CAPSL` description);

• *Rules* – an additional set of terms, also called *rule-terms*; such a rule-term r is of the form $i.A \rightarrow B : t$, where $i \in Steps$, $A, B \in Ho$ and $t \in \mathcal{T}$ (hence, it corresponds to a `CAPSL` communication rule).

[1]An atomic key is an element of $\mathcal{K}$. In turn, a complex key is a non-atomic term (i.e., an element from $\mathcal{T} \setminus \mathcal{T}_0$).

We introduce two function symbols on the set of rule-terms, *step* and t:

- $step(r) = i$, for $r \in Rules$ and $r{=}i.A \to B : t$ (hence, $A, B \in Ho, t \in \mathcal{T}, i \in Steps$);
- $t(r) = t$, for $r \in Rules$ and $r{=}i.A \to B : t$ (hence, $A, B \in Ho, t \in \mathcal{T}, i \in Steps$).

In a denotation, the function *step* returns the step of a rule-term (i.e., the step of a corresponding `CAPSL` rule), whilst $t(r)$ returns the message in a rule-term r (i.e., the message in a corresponding `CAPSL` protocol rule).

Further, we select the subset of terms on the signature $\mathcal{S}$ which precisely denotes the messages exchanged:

- $Msg = \bigcup\limits_{r \in Rules} \{t' \in \mathcal{T} \,|\, t' = t(r)\}$.

On the set of messages we then define two more notions. We use a function symbol *composites* which when applied to a message gives all its encrypted subterms; it is hence, an alias for $ESub$ when referring to messages precisely (and not to terms in general). The set $Composites$ denotes all (proper or improper) encrypted parts of the messages in the entire `CAPSL` protocol description. Formally, on the signature $\mathcal{S}$ as considered, we define:

- for $t \in Msg$, $composites(t){=}ESub(t)$;
- $Composites = \bigcup\limits_{t \in Msg} composites(t)$.

With these notions, the `MESSAGES` section of a `CAPSL` description is now fully described.

For a fixed $A \in Ho$, we formalise the *role of A* described by a `CAPSL` file as the set of rule-terms restricted to that principal A. Alternatively, we call this *an A-role* (under the signature $\mathcal{S}$):

- $Rules^A = \{i.A \to B : t,\ j.B \to A : t \mid B \in Ho,\ t \in Msg,\ i, j \in Steps\}$.

We define other objects related to the *A-role*, e.g., the steps where A either sends or receives and the messages either sent or received particularly by A:

- $Steps^A = \{i \in Steps \,|\, (i.B \to A : t) \text{ or } (i.A \to B : t), B \in Ho, t \in Msg\}$;
- $Msg^A = \bigcup\limits_{r \in Rules^A} \{t' \in \mathcal{T} | t' = t(r)\}$.

Over 2^{Steps^A}, 2^{Steps^A} and $2^{Steps^A} \times Steps^A$ we respectively define the function symbols $first$, *last* and *next*. We assume that, in a denotation, the sort *Steps* ranges over a totally ordered and countable set. Thus, $first(Steps^A)$ returns the first step in which an *A-role* is engaged. In the same way, *last* returns the last step of the *A-role*. For some $j \in Steps^A$, $next(Steps^A, j)$ returns the step immediately following j in which the A-*role* will be engaged (according to the `CAPSL` protocol description). In addition, sometimes we will refer to the set $SentMsg^A$ to denote all the messages sent by the A-*role* (as the `CAPSL` description implies). Hence, $SentMsg^A = \bigcup_{r \in Rules^A} \{t' \in \mathcal{T} | t' = t(r),\, r = i.A \to B : t\}$. Similarly, the set $ReceivedMsg^A = Msg^A \setminus SentMsg^A$ denotes the messages received by the A-*role* (as per the `CAPSL` description). So, the messages received by the A-*role* is given by $ReceivedMsg^A = \bigcup_{r \in Rules^A} \{t' \in \mathcal{T} \,|\, t' = t(r),\, r = i.B \to A : t\}$.

Further specialisations, which are relative to a generic A-role, are introduced below:

- for any $A \in Ho$ in the signature $\mathcal{S}$, $Atoms^A = \bigcup_{r \in Rules^A} \{t' \in Sub(t'') \cap \mathcal{T}_0 \mid t'' = t(r)\} \bigcup \bigcup_{r \in Rules^A} \{A, B, K^e_B, K_{AB} | \, r = i.A \to B : t\} \bigcup \{K^d_A \in \mathcal{K}_1\}$;

  The set $Atoms^A$ contains all atomic terms given and/or communicated to A throughout her role. By the second and the third terms of the union, $Atoms^A$ includes the atoms that A knew before having proceeded with her first action. These atoms correspond to the variables appearing in the `CAPSL` assertions of type `HOLDS A: VAR` and `DENOTES` of a protocol description. By the first term of the union, $Atoms^A$ contains the atoms that A receives in her role. Importantly, note that some of these atoms might be within an encryption that A cannot decrypt, i.e., some atoms might be unintelligible to A. We will now formalise the proper subsets of $Atoms^A$ suggested above.

- for any $A \in Ho$ in the signature $\mathcal{S}$ as considered,
  $OwnedAtoms^A = \Big\{ t \in Atoms^A \cap Sub(t') \,|\, (Sort(t) \in \{\mathcal{N}, \mathcal{K}_0\})$ and $((i.A \to B : t') \in Rules^A)$ and $(\nexists j \in Steps^A, j < i$ such that $j.C \to A : t''$ with $t \in Sub(t''))\Big\} \bigcup \Big\{ K^d_A \in \mathcal{K}_1 \Big\}$;

  Thus, a nonce or a short-term key t is in $OwnedAtoms^A$ if the `CAPSL` description stipulates the following: A ought to send t as part of message t' at step i and there is no smaller step j where some C could have made t known to A. Hence, t is inherent to A and A is the first to send it through the network, at some step i.

  Then, most of the atoms in $OwnedAtoms^A$ correspond to variables designated under `HOLDS A: variables` in the `ASSUMPTION` section of the `CAPSL` descriptions. The rest of them are specified under the `DENOTES` section of the `CAPSL` description.

- for any $A \in Ho$, in the signature $\mathcal{S}$ as considered
  $\mathit{LearnedAtoms}^A = \{t \in \mathit{Atoms}^A \setminus \mathit{OwnedAtoms}^A \mid \mathit{Sort}(t) \in \{\mathcal{N}, \mathcal{K}_0\}\}$;

  From the meaning of $\mathit{OwnedAtom}^A$, it follows that $\mathit{LearnedAtoms}^A$ denotes the set of nonces and short-term keys attained by A throughout her role. Some of these atoms might not be intelligible to A, as they might be part of composites she receives but cannot decrypt.

- for any $A \in Ho$, in the signature $\mathcal{S}$ as considered, for each $i \in \mathit{Steps}^A$:
  $i\text{-}\mathit{LearnedAtoms}^A = \bigcup_{\substack{r \in \mathit{Rules}^A, \\ j = \mathit{step}(r), j \leq i}} \{\mathit{OSub}(t') \mid t' = t(r), r = j.B \to A : t\} \cap \mathit{LearnedAtoms}$

  Therefore, $i\text{-}\mathit{LearnedAtoms}^A$ contains all the atomic-terms, not owned by A, but sent to A by step i of her role. Again, some of these atoms might not be intelligible to A, as they might be part of composites she receives but cannot decrypt.

- $\mathit{PublicData} = \mathit{Atoms}^A \setminus \{\mathit{OwnedAtoms}^A \cup \mathit{LearnedAtoms}^A\}$. The *public data* consists of principals (e.g., $\mathcal{A}$) and long-term keys (e.g., $\mathcal{K}_1$).

Remark 3.1.2 *For simplicity of explanations, we introduced only the cryptographic sorts $\mathcal{A}$, $\mathcal{K}$, $\mathcal{N}$ to denote principals, keys and nonces, respectively. The signature can be naturally extended to contain other sorts relevant to security protocols, e.g., timestamps, fields, etc. The set of terms introduced would consequently be modified. For instance, in a more general signature, $\mathit{LearnedAtoms}^A$ would be $\{t \in \mathit{Atoms}^A \setminus \mathit{OwnedAtoms}^A \mid \mathit{Sort}(t) \notin \{\mathcal{A}, \mathcal{K}_1\}\}$, etc. Other primitives, except for pairing and encryption, can also be included, e.g., hashing, digital signatures, etc. The results that we will present in this section can be extended in the context of such additional sorts and/or terms, as Chapter 5 will show.*

To sum up, we have presented a formalisation of `CAPSL` descriptions through a (many-sorted) signature $\mathcal{S}$ and a term-algebra on $\mathcal{S}$. We will now use this to partition the class of `CAPSL`-described protocols in two: *receiver-transparent protocols* (*RTP*) and *receiver-opaque protocols* (*ROP*). As `CAPSL` is a language for describing authentication and key-establishment protocols, the dichotomy between RTP and ROP refers to authentication and key-establishment. However, the notions implied are general and could refer to any other class of protocols, e.g. e-voting, contract signing, etc.

Definition 3.1.3 (Receiver-Transparent Protocols) *Let Pr be a protocol and $\mathcal{S}$ be its signature as above. The protocol Pr is called* receiver-transparent *(*RT*) if for all principals $A \in Ho$, for all rules $r \in Rules^A$ of the form $r = i.B \rightarrow A : t$, for all $t' \in composites(t)$ and $t_0 \in OSub(t')_{last(t')}$, it is implied that $t_0 \in i\text{-}LearnedAtoms^A \cup OwnedAtoms^A$.*

In an RT protocol, all keys t_0 of any encrypted part of a message t received by an A-*role* at step i are required to be either in $i\text{-}LearnedAtoms^A$ or in $OwnedAtoms^A$. Hence, in receiver-transparent protocols the receivers are always able to decrypt any composite of a message down to its atomic parts as soon as the message is received.

Note that the class of RT protocols is not equal to the class of protocols with one single level of encryption in any message sent or received. In RT protocols, there can be many layers of encryption in the messages transmitted, but the receivers can decrypt all involved composites of these messages. Thus, the class of protocols with only one level of encryption is a proper subclass of the class of RT protocols. A large number of authentication protocols fall into the class of RT protocols (see authentication protocol libraries and repositories [48, 49, 119]). An example of an RT protocol is the Needham Schroeder Public Key (NSPK) protocol, the description of which we recalled in Example 2.2.1.

By contrast, receiver-opaque protocols are protocols where (at least) one receiver is unable to decrypt fully (at least) one of the received message strictly upon its arrival. Namely, there is a composite in a received message for which the receiver does not hold yet the decryption key. It might be the case that the receiver is never able to decrypt that composite, i.e., not even after the completion of the protocol. The latter is the case of several chain-authentication protocols. In Example 3.1.4, we illustrate the RO, chain-authentication protocol due to Otway and Bull [39].

As we can see in Example 3.1.4, in the Otway-Bull authentication protocol an honest B party cannot decrypt/analyse the message he receives from A at step 1 (i.e., it is encrypted/hashed with the symmetric key K_a that only A posseses); nor can C decrypt/analyse the message she receives from B at step 2, as it is encrypted/hashed with the symmetric key K_b unknown to C. We can also see that, for instance at step 4, C cannot decrypt the last three proper composites of the message received (i.e., $\{|K_{bc}, C, N_b|\}_{K_b}$, $\{|K_{ba}, A, N_b|\}_{K_b}$, $\{|K_{ba}, B, N_a|\}_{K_a}$). However, there are composites of message 4 which C can decrypt/analyse, i.e., the first two proper composites, $\{|K_{cs}, S, N_c|\}_{K_c}$

and $\{|K_{bc},\, B,\, N_c|\}_{K_c}$, as each of these is encrypted/hashed with the keys known to C.

Example 3.1.4 *The Otway-Bull [39] Protocol: A Receiver-Opaque Protocol*

$$
\begin{aligned}
1.\, A \to B :& Hash_{K_a}\{|A,\, B,\, N_a,\, -|\}\\
2.\, B \to C :& Hash_{K_b}\{|B,\, C,\, N_b,\, Hash_{K_a}\{|A,\, B,\, N_a,\, -|\}|\}\\
3.\, C \to S :& Hash_{K_c}\{|C,\, S,\, N_c,\, Hash_{K_b}\{|B,\, C,\, N_b,\, ,Hash_{K_a}\{|A,\, B,\, N_a,\, -|\}|\}|\}\\
4.\, S \to C :& \{|K_{cs},\, S,\, N_c|\}_{K_c},\, \{|K_{bc},\, B,\, N_c|\}_{K_c},\\
& \{|K_{bc},\, C,\, N_b|\}_{K_b},\, \{|K_{ba},\, A,\, N_b|\}_{K_b},\\
& \{|K_{ba},\, B,\, N_a|\}_{K_a}\\
5.\, C \to B :& \{|K_{bc},\, C,\, N_b|\}_{K_b},\, \{|K_{ba},\, A,\, N_b|\}_{K_b},\\
& \{|K_{ba},\, B,\, N_a|\}_{K_a}\\
6.\, B \to A :& \{|K_{ba},\, B,\, N_a|\}_{K_a}
\end{aligned}
$$

Formally, a receiver-opaque protocol is defined as follows.

Definition 3.1.5 (Receiver-Opaque Protocols) *Let Pr be a protocol and $\mathcal{S}$ be its signature as above. The protocol Pr is called* receiver-opaque *(*RO*) if there exists at least one principal $A \in Ho$, for which there exists (at least) a rule $r \in Rules^A$ of the form $r = i.B \to A : t$ such that there exists $t' \in composites(t)$ with $t'' = OSub(t')_{last(t')}$ and $t'' \notin i\text{-}LearnedAtoms^A \cup OwnedAtoms^A$.*

The above informally says the following. If for some A-*role*, for a step i of the A-*role* and t a message sent to A at step i, there exists at least one composite of t whose encryption key t'' is still unknown to A at the current step i, then the protocol Pr is an RO protocol.

In the last two pages, we differentiated two distinct classes of protocols (ROP and RTP). We outline such a distinction for optimisation purposes; as next chapters will show, we will model each class systematically, but through bespoke MAS formalisations. In the following, we insist on how protocol executions will be formalised in our model.

Protocol Instantiations. Recall from Chapter 2 that *substitutions* are used (in [171,188,44,11] etc.) to model protocol executions; given a protocol description/signature, substitutions are viewed as functions that map terms to values: principals to names of protocol participants, short-term key to short-term key values, nonces to arbitrary values.

In our formalisation, we extend substitutions to map terms of sort *Step* over natural numbers. Substitutions are applied uniformly over terms and sets of terms, e.g., $\sigma(K_{AB}) = K_{\sigma(A)\sigma(B)}$ or $\sigma(K^e_A) = K^e_{\sigma(A)}$. At the denotational level we override the names of the function symbols, i.e., we write $OSub(x)$, even for the ground value $x \in \mathcal{R}_t$ and, as in the signature, we intend that it returns the values within x in a component-wise manner.

We emphasise the following distinction between symbolic names and concrete parties engaged in the protocol executions:

- a *principal* is a symbolic term of sort $\mathcal{A}$;
- a *participant* is a substituted/ground term of sort $\mathcal{A}$.

Hence, $A \in \mathcal{A}$ is a principal, whereas the name $\sigma(A) = alice$ denotes a participant.

Substitutions homomorphically apply to rules, sequence of rules and, therefore, roles. So, one substitution can uniformly map a high-level description of a protocol into a representation of one running protocol session. Then, to give such a substitution for the entire protocol, we need then to give a substitution that operates on all of its atomic terms.

Definition 3.1.6 (Protocol Instantiation) *A* protocol instantiation *is a substitution that maps each atomic term on the signature* $\mathcal{S}$ *into a value from a corresponding range.*

We recall the Alice&Bob description of NSPK protocol already presented in Example 2.2.1:

$$\begin{aligned}
&1.\ A \to B : \{A, N_A\}_{pub(B)}\\
&2.\ B \to A : \{N_A, N_B\}_{pub(A)}\\
&3.\ A \to B : \{N_B\}_{pub(B)}.
\end{aligned}$$

An example of a NSPK instantiation is the substitution σ_1 given in Example 3.1.7. Additionally to the terms shown in Example 3.1.7, the substitution σ_1 is also uniformly applied to keys.

Example 3.1.7 (A Substitution for the NSPK Protocol)

$$\sigma_1(A) = alice,\ \sigma_1(B) = intruder,\ \sigma_1(N_A) = m,\ \sigma_1(N_B) = n.$$

Protocol instantiations can also be given role by role. For instance, instead of giving one substitution for the entire $\mathcal{T}_0$, we can give several substitutions each applying to the atoms in one role only. Thus, σ_1 in Example 3.1.7 can be separated into $\sigma_{11}(A\text{-}role)$ with σ_{11} applied to $A, B, N_A \in OwnedAtoms^A$ (extended uniformly on K_B^e, K_A^d), plus $\sigma_{12}(B\text{-}role)$ with σ_{12} applied to $A, B, N_B \in OwnedAtoms^B$ (extended uniformly on K_A^e, K_B^d). Operating such restrictions relies on the fact that atoms in $\mathcal{T}_0$ can be partitioned by roles, i.e., by $OwnedAtoms^X$, $X \in Ho$. For example, in the above, N_A belongs to the *A-role* and N_B belongs to the *B-role*. In the following sections we will formalise this procedure of partitioning a protocol instantiation into role-substitutions (respecting certain criteria). In the following, we consider that the executions of a protocol are rendered by a set of role-substitutions.

Let Pr be a `CAPSL`-described protocol and Σ_{Pr} be a set of role-substitutions for Pr. The set Σ_{Pr} may contain more than one substitution for the same *X-role*, $X \in Ho$. We write $\sigma(X\text{-}role)$ to denote one arbitrary $\sigma \in \Sigma_{Pr}$ that applies to *X-role*, $X \in Ho$. To abridge notations, sometimes we also write $\sigma(X\text{-}role) \in \Sigma_{Pr}$ if $\sigma \in \Sigma_{Pr}$ that applies to *X-role*, $X \in Ho$.

We introduce an explicit notion of *range*. Formally, we assume that each sort X (e.g., $\mathcal{A}$, $\mathcal{K}$, $\mathcal{N}$) has a finite value-range $\mathcal{R}_X$. A role-substitution $\sigma(A\text{-}role) \in \Sigma_{Pr}$ takes any atomic term $t \in OwnedAtoms^A$ of sort X to a value in $\mathcal{R}_X$, where $A \in Ho$. The range of a non-atomic term is defined inductively on its structure as the Cartesian product of the ranges of its subterms taken in the order of their appearance: for all $t \in \mathcal{T} \setminus \mathcal{T}_0$, $\mathcal{R}_t = \prod_{t' \in OSub(t)} R_{t'}$. The explanations above are formalised as follows.

Definition 3.1.8 (Range of a Term under $\mathcal{S}$) *The* range of a term *under a signature $\mathcal{S}$ and a given set of substitutions is given by:*
$$\begin{cases} \mathcal{R}_t = \mathcal{R}_X, & t \in X, X \in \{\mathcal{A}, \mathcal{K}, \mathcal{N}\}; \\ \mathcal{R}_t = \prod_{t' \in OSub(t)} R_{t'}, & t \in \mathcal{T} \setminus \mathcal{T}_0. \end{cases}$$

Remark 3.1.9 *For $X \in \{\mathcal{N}, \mathcal{K}_0\}$, we consider the range $\mathcal{R}_X$ large enough for:*

- *any role-substitution in Σ_{Pr} to map different nonces/short-term keys in $OwnedAtoms^A$ into different values (e.g., for some $A \in Ho$, for every $\sigma(A\text{-}role) \in \Sigma_{Pr}$, for $t, t' \in OwnedAtoms^A$ of sort X, $t \neq t'$, there exist $x, y \in \mathcal{R}_X$, $x \neq y$ such that $\sigma(t) = x$ and $\sigma(t') = y$);*

- *different role-substitutions in Σ_{Pr} to map the same atom in $OwnedAtoms^A$ differently (e.g., for some $A \in Ho$, for some $\sigma(A\text{-}role), \sigma'(A\text{-}role) \in \Sigma_{Pr}$, $\sigma \neq \sigma'$, for $t \in OwnedAtoms^A$ of sort X, there exist $x, y \in \mathcal{R}_X$, $x \neq y$ such that $\sigma(t) = x$ and $\sigma'(t) = y$);*

- *atoms of the same sort X, from $OwnedAtoms^A$ and $OwnedAtoms^B$, to be mapped distinctively under the role-substitutions in Σ_{Pr} (e.g., for some $A \in Ho$, $B \in Ho$, $A \neq B$, for role-substitutions $\sigma(A\text{-}role)$, $\sigma(B\text{-}role) \in \Sigma_{Pr}$, $t \in OwnedAtoms^A$, $t' \in OwnedAtoms^B$ of sort X, there exist $x, y \in \mathcal{R}_X$, $x \neq y$ such that $\sigma(t) = x$ and $\sigma'(t') = y$);*

- *any role-substitution in Σ_{Pr} to map atoms in $OwnedAtoms^A$ distinctively and not in a biunivocal correspondence. In other words, for $A \in Ho$, for some $\sigma(A\text{-}role) \in \Sigma_{Pr}$, $t \in OwnedAtoms^A$ of sort X, if σ maps the term t to some value $y \in \mathcal{R}_X$ and this is not known to an observer, the observer should not be able to infer $t \in OwnedAtoms^A$ such that $\sigma(t) = y$ when the signature $\mathcal{S}$, the name of the instantiated-roles as per $\Sigma_{Pr}(\mathcal{A})$, other mappings of Σ_{Pr} over $\mathcal{R}_X$ and $y \in \mathcal{R}_X$ are given.*

We consider that the explanations in Remark 3.1.9 are sufficient for the content of this chapter. We will formalise them further in Chapter 4 (e.g., Definition 4.1.6).

Role-substitutions and their ranges are used to map roles into agents and create the basis of our MAS modelling for security protocol executions, as the following sections will show.

In this section we have formalised the `TYPESPEC` and the `PROTOCOL` sections of a `CAPSL` description through a signature $\mathcal{S}$. We proceed to formalise the `GOALS` section of a `CAPSL` description.

3.2 A Formalisation of `CAPSL` Security Requirements

In Section 2.2.4 we summarised the security requirements expressible in the `GOALS` section of a `CAPSL` protocol description. In this line of work, we employ a significantly large fragment of the `CAPSL` language for security goals (i.e., we only dismiss the `CAPSL` assertions `ASSUME` and `PROVE`). Over this fragment of the `CAPSL` language for security requirements, we specialise `CAPSL` goals into *atomic* and *complex*. The details of this dichotomy is presented below.

The *atomic `CAPSL` goals* or *atomic `CAPSL` facts* are defined as follows:

$$\alpha ::= \texttt{AGREE}\,\texttt{A},\texttt{B} : \overline{\texttt{Var}} \quad | \quad \texttt{HOLDS}\,\texttt{A} : \overline{\texttt{Var}} \quad | \quad \texttt{PRECEDES}\,\texttt{A},\texttt{B} : \overline{\texttt{Var}} \quad | \quad \texttt{SECRET} : \overline{\texttt{Var}},$$

where `A` and `B` are `CAPSL` declarations of type `Principal` (or `Node`) and $\overline{\texttt{Var}}$ is a list of `CAPSL` variables (i.e., declared under the `CAPSL` section `VARIABLES`). On the signature $\mathcal{S}$, $A, B \in Ho$ and

$Vars \subset \mathcal{T}_0$ is a set of atomic terms respectively corresponding to the elements in the list $\overline{\texttt{Var}}$ of `CAPSL` variables.

Informally, $\texttt{AGREE}\,\texttt{A},\texttt{B} : \overline{\texttt{Var}}$ expresses that `A` and `B` have the same values for variables in $\overline{\texttt{Var}}$, i.e., protocol participants of *A-role* and *B-role* have the same values for the atoms in $Vars$. The atomic goal $\texttt{HOLDS}\,\texttt{A} : \overline{\texttt{Var}}$ states that any instantiated *A-role* has concrete values for variables in $\overline{\texttt{Var}}$. The atomic goal $\texttt{PRECEDES}\,\texttt{A},\texttt{B} : \overline{\texttt{Var}}$ stipulates that any instantiated *A-role* has access to the variables in $\overline{\texttt{Var}}$ prior to any participant of a `B`-*role*, and that when the protocol run has ended these participants should agree on the data in $\overline{\texttt{Var}}$. Finally, the atomic goal $\texttt{SECRET} : \overline{\texttt{Var}}$ requires that values of the variables in $\overline{\texttt{Var}}$ are not in possession of any unintended party.

Complex `CAPSL` goals are formed with belief and knowledge assertions, upon the following grammar:

$$\alpha ::= \alpha \quad | \quad \texttt{KNOWS}\,A : \alpha \quad | \quad \texttt{BELIEVES}\,A : \alpha,$$

where α is an atomic fact as above. Informally, the goal $\texttt{KNOWS}\,A : \alpha$ expresses that `A` acknowledges the fact α, and the goal $\texttt{BELIEVES}\,A : \alpha$ states that `A` is justified in considering that the fact α is the case. The `CAPSL` documentation does not provide a full description of the precise meaning of the belief and knowledge assertions. However, the following examples and their respective explanations may serve as a guideline.

Example 3.2.1 (Doxastic goal)

$$\textit{BELIEVES}\,B : \textit{HOLDS}\,A : \textit{K} \tag{3.1}$$

The goal stated in Example 3.2.1 is interpreted in [66], page 3, as "if the B-session completes, B is justified in believing that A holds K. The belief assertion means that $\texttt{HOLDS}\,A : \texttt{K}$ is interpreted in the context of B's values for A and K".

Example 3.2.2 (Acknowledged authentication)

$$\textit{KNOWS}\,A : \textit{KNOWS}\,B : \textit{AGREE}\,B, A : \textit{Ma} \tag{3.2}$$

The `CAPSL` goal in Example 3.2.2 is a goal required by KSL [111], a repeated authentication protocol. Kehne-Schönwälder-Langendörfer (KSL) is a variant of the Kerberos protocol and its ISO standard requires the acknowledgement of the other party's knowledge. This is specified in `CAPSL` through a goal with two levels of nesting for the knowledge assertions, as in Example 3.2.2; this expresses that if `A` terminates its run of the protocol, then it knows that `B` knows that `A` and `B` have the same value for `Ma`.

The `CAPSL` language supports BAN logic specifications with several levels of nested knowledge and belief assertions. Take the example of the NSPK protocol. Its BAN analysis in [43] formalises that if the protocol parties are honest, then it attains that `B` believes that `A` believes that `B` believes that the nonce `Nb` is secret. In `CAPSL` this is expressed by means of the following goal:

Example 3.2.3 (BAN goal)

$$\mathit{BELIEVES}\,B : \mathit{BELIEVES}\,A : \mathit{BELIEVES}\,B : \mathit{SECRET} : \mathit{Nb}$$

In Section 3.2, we have classified `CAPSL` goals as atomic, if they contain no belief or knowledge assertions, and as complex, otherwise. The first step towards a deeper formalisation of the `CAPSL` goals with respect to the signature $\mathcal{S}$ is to consider a logically-extended signature $\mathcal{S}_L$. Such a signature $\mathcal{S}_L$ will be obtained from $\mathcal{S}$ by adding goal-predicate symbols (operating on terms). To give the exact denotation of these predicates we need a precise model of execution. We will therefore give these formalisations in the later sections once we have introduced our MAS model for multi-session protocol execution. As a preamble, the evaluation (i.e., semantics) of the specifications derived from these predicates would reside in interpreting CTLK formulae over the MAS model.

Overall, in Section 3 we have formalised the `TYPESPEC`, the `PROTOCOL` and the `GOALS` section of a `CAPSL` description through a signature $\mathcal{S}$. Protocol execution scenarios have been formalised through role-substitutions of the variables and terms on the signature. The next section will give a MAS-based model for the actual multi-session execution of security protocols.

3.3 Υ_{IS}: Interpreted Systems for CAPSL-described RT Security Protocols

In this section we present an *IS-based operational semantics* for the class $\mathcal{P}$ of receiver-transparent (authentication and key-establishment) protocols described in CAPSL. We use $\Upsilon_{IS}^{\mathcal{P}}$ to denote the resulting class of IS-based formalisations of these protocols. We use Υ_{IS}^{Pr} to denote such a formalism for a given RT protocol Pr. When either the protocol Pr or the class of protocols $\mathcal{P}$ is implicit, we simply write Υ_{IS}. We use M_{IS} to denote the model resulted from the unwinding of the Υ_{IS} IS-based formalism.

We introduce some function and predicate symbols on the signature $\mathcal{S}$. In the model to be given, these are helpers in describing each agent and its local states, local actions, local evolution function, etc. We hereby introduce these symbols by using generic prototypes and general denotations, i.e., not exhibiting aspects of the protocol execution semantics. In later sections, we will use specific arguments (e.g., specific local states, protocol-biased conditions) to specialise the denotations of these symbols within a MAS semantics for protocol executions.

3.3.1 Function Symbols on the Signature $\mathcal{S}$ for Υ_{IS}

Let $\mathcal{S}$ be a signature for an RT protocol Pr, $\mathcal{T}_0$ be the set of basic terms , $\mathcal{T}$ be the set of terms, **Ind** be a set of indices (i.e., a subset of the natural numbers) and Σ be a set of instantiations. Also, consider the ranges of terms under $\mathcal{S}$ and Σ, as previously defined. Let a special range containing the boolean values "true" and "false" be called *Boolean* and a special range containing the unique value $\perp$ be called $\mathcal{R}_\perp$.

On the above, we define the following functional symbols:

- *in_match* with the domain $\mathcal{T} \times (\mathbf{Ind} \times \mathbf{Ind})$ and the codomain *Boolean*
- *out_match* with the domain $2^{\mathcal{T}_0 \times (\mathcal{R}_{\mathcal{T}_0} \cup \mathcal{R}_\perp)} \times \mathcal{T}$ and the codomain *Boolean*;
- *decryptable* with the domain $2^{\mathcal{T}_0 \times (\mathcal{R}_{\mathcal{T}_0} \cup \mathcal{R}_\perp)} \times \mathcal{T}$ and the codomain *Boolean*;
- *set* with the domain $2^{\mathcal{T}_0 \times (\mathcal{R}_{\mathcal{T}_0} \cup \mathcal{R}_\perp)} \times \mathcal{T}$ and the codomain $2^{\mathcal{T}_0 \times (\mathcal{R}_{\mathcal{T}_0} \cup \mathcal{R}_\perp)}$;

- *setMsg* with the domain $((\, 2^{(Msg \times Roles \times \mathcal{R}_{Ho}) \times (\mathcal{R}_{Msg} \cup \mathcal{R}_{\perp})}\,) \times Msg)$ and the codomain $(\, 2^{(Msg \times Roles \times \mathcal{R}_{Ho}) \times (\mathcal{R}_{Msg} \cup \mathcal{R}_{\perp})}\,)$;

- *setAt* with the domain $(\,(2^{(\mathcal{T}_0 \times Msg \times Roles \times \mathcal{R}_{Ho}) \times (\mathcal{R}_{\mathcal{T}_0} \cup \mathcal{R}_{\perp})}\,) \times T_0 \times Msg)$ and the codomain $(\, 2^{(\mathcal{T}_0 \times Msg \times Roles \times \mathcal{R}_{Ho}) \times (\mathcal{R}_{\mathcal{T}_0} \cup \mathcal{R}_{\perp})}\,)$;

- *recordAt* with the domain $(\,(2^{T_0 \times \mathcal{R}_{\mathcal{T}_0}}) \times T_0 \times Msg)$ and the codomain $(\, 2^{\mathcal{R}_{\mathcal{T}_0}}\,)$;

- *recordPart* with the domain $(\,(2^{T \times \mathcal{R}_{\mathcal{T}}}) \times T \times Msg)$ and the codomain $(\, 2^{\mathcal{R}_{\mathcal{T}}}\,)$;

- *analz* with the domain $(2^{T_0 \times \mathcal{R}_{\mathcal{T}_0}} \times 2^{T \times \mathcal{R}_{\mathcal{T}}} \times Msg)$ and with the codomain *Boolean*;

- *consistent* with the domain $\mathcal{T} \times 2^{\mathcal{T}_0 \times (\mathcal{R}_{\mathcal{T}_0} \cup \mathcal{R}_{\perp})}$ and the codomain *Boolean*;

- *construct* with the domain $\mathcal{T} \times 2^{\mathcal{T}_0 \times (\mathcal{R}_{\mathcal{T}_0} \cup \mathcal{R}_{\perp})} \times 2^{\mathcal{T}_0 \times (\mathcal{R}_{\mathcal{T}_0} \cup \mathcal{R}_{\perp})}$ and the codomain $\mathcal{T} \times (\mathcal{R}_{\mathcal{T}} \cup \mathcal{R}_{\perp})$;

- *compose* with the domain $\mathcal{T} \times 2^{\mathcal{T}_0 \times (\mathcal{R}_{\mathcal{T}_0} \cup \mathcal{R}_{\perp})}$ and the codomain $\mathcal{T} \times (\mathcal{R}_{\mathcal{T}} \cup \mathcal{R}_{\perp})$;

- *synth* with the domain $\mathcal{T} \times 2^{\mathcal{T} \times \mathcal{R}_{\mathcal{T}}}$ and the codomain $\mathcal{T} \times (\mathcal{R}_{\mathcal{T}} \cup \mathcal{R}_{\perp})$.

Let $\sigma \in \Sigma$ be an arbitrary substitution. In the following, we give the *denotational interpretation under* σ, $\mathbb{I}^{\sigma}$, of these function symbols.

- for any $t \in \mathcal{T}$, any $(i,j) \in (\mathbf{Ind} \times \mathbf{Ind})$,
$$in\_match^{\mathbb{I}^{\sigma}}(t,i,j) = \begin{cases} true, & \text{if } (\,OSub(t)_i = OSub(t)_j\,) \Rightarrow (\,OSub(\sigma(t))_i = OSub(\sigma(t))_j\,) \\ false, & \text{if } (\,OSub(t)_i = OSub(t)_j\,) \wedge (\,OSub(\sigma(t))_i \neq OSub(\sigma(t))_j\,) \end{cases}$$

Let the notation $[|\varphi|]$ denote the usual truth-valuation of φ, where φ is a simple boolean expression (i.e., conjunction, implication, etc.).

To shorten the description, an alternative way to give the interpretation of *in_match* is:

$in\_match^{\mathbb{I}^{\sigma}}(t,i,j) \equiv [|(\,OSub(t)_i = OSub(t)_j\,) \Rightarrow (\,OSub(\sigma(t))_i = OSub(\sigma(t))_j\,)|]$.

The denotation of $in\_match(t,i,j)$ says that if the atomic terms at position i and j within t are the same, then under a substitution they should be mapped onto the same values (i.e., the substituted values are an *inner-match* with respect to the pattern of t).

- for any $V \subseteq \mathcal{T}_0 \times (\mathcal{R}_{\mathcal{T}_0} \cup \mathcal{R}_\perp)$, any $t \in \mathcal{T}$,

$$out\_match^{\mathbb{I}^\sigma}(V,t) = \begin{cases} true, & \text{if } v = \sigma(t''), \quad \text{for } \forall (t',v) \in V \text{ with } v \neq \perp, \\ & \qquad\qquad\qquad \forall t'' \in Sub(t) \cap \mathcal{T}_0 \text{ and } t' = t'' \\ false, & \text{if } \exists (t',v) \in V \text{ with } v \neq \perp, \exists t'' \in Sub(t) \\ & \text{such that } t' = t'' \text{ and } v \neq \sigma(t'') \end{cases}$$

Using the $[|\cdot|]$ notation,

$out\_match^{\mathbb{I}^\sigma}(V,t) = [|(\,\forall (t',v) \in V \, with \, v \neq \perp, t' \in Sub(t) \cap \mathcal{T}_0\,) \Rightarrow (\,v = \sigma(t')\,)|]$.

The denotation of $out\_match(V,t)$ under some substitution σ says the following. As long as V has entries t' for atomic subterms t and these are mapped to something else but $\perp$, then the *outer* substitution σ *matches* V if it maps each t' to the same value as V recorded for t'.

- for any $V \subseteq \mathcal{T}_0 \times (\mathcal{R}_{\mathcal{T}_0} \cup \mathcal{R}_\perp)$, any $t \in \mathcal{T}$,

$$decryptable^{\mathbb{I}^\sigma}(V,t) = \begin{cases} true, & \text{if } (\,\forall t' \in ESub(t),\, last(t') = i, \sigma(t') = x'\,) \Rightarrow (\,OSub(x')_i \in V\,) \\ & \\ false, & \text{if } (t'',\perp) \in V \quad \text{for some } t' \in ESub(t) \text{ and } t'' = OSub(t')_i. \end{cases}$$

Using the $[|\cdot|]$ notation,

$[|(\,\forall t' \in ESub(t), last(t') = i\,) \Rightarrow (\,(OSub(t')_i, v) \in V \wedge v \neq \perp \wedge v = OSub(\sigma(t'))_i\,)|]$.

The denotation of $decryptable(V,t)$ under some substitution σ is that $\sigma(t)$ is decryptable if the values which correspond to keys in $\sigma(t)$ are all to be found in V.

- for any $V \subseteq \mathcal{T}_0 \times (\mathcal{R}_{\mathcal{T}_0} \cup \mathcal{R}_\perp)$, any $t \in \mathcal{T}$,

$set^{\mathbb{I}^\sigma}(V,t) = \{(t',\sigma(t'')) \cup (V \setminus (t',\perp)) \mid (t',\perp) \in V, t'' \in Sub(t) \cap \mathcal{T}_0,\ t' = t''.\}$

Let $\sigma[v/t]$ be a substitution that uniformly maps only the atomic term t to the value v leaving all the other terms in an underlying structure unchanged. The denotation of the *set* symbol can also be given as: $set^{\mathbb{I}^\sigma}(V,t) = \sigma[n/t'](V)$, for all $(t',\perp) \in V$, $t' \in Sub(t) \cap \mathcal{T}_0$ and $\sigma(t') = n$.

The denotation of $set(V,t)$ says that "holes" in V (i.e., entries where atomic subterms of t are assigned to $\perp$) are "filled with" (i.e., *set to*) actual values according to what $\sigma(t)$ dictates.

- for any $V \subseteq (Msg \times Roles \times \mathcal{R}_{Ho}) \times (\mathcal{R}_{Msg} \cup \mathcal{R}_\perp)$, any $t \in Msg$,

$setMsg^{\mathbb{I}^\sigma}(V,t) = \{((\,t, A, \sigma(A)\,), \sigma(t)) \cup (V \setminus ((t, A, \sigma(A)), \perp)) \mid (\,t, A, \sigma(A)\,), \perp) \in V\}$.

Let $\sigma[v/t]$ be a substitution extended to non-atomic terms that uniformly maps only the non-atomic term t to the value v leaving all the other terms in an underlying structure unchanged. Then, for $A \in Ho$, $n \in \mathcal{R}_t$ and $alice \in \mathcal{R}_{Ho}$, an example of applying $setMsg$ is: $setMsg^{\mathbb{I}^\sigma}(V,t) = \sigma[(n, A, alice)/(t, A, alice)](V)$, where $((t, A, alice), \perp) \in V$, $\sigma(t) = n$ and $\sigma(A) = alice$.

The denotation of $setMsg(V,t)$ under a substitution σ says that "holes" in V (i.e., values $\perp$) at entry $(\,t, A, \sigma(A)\,)$ are "filled" with the value $\sigma(t)$; hence, the value for entry $(\,t, A, \sigma(A)\,)$ is *set to* $\sigma(t)$.

- for any $V \subseteq (\mathcal{T}_0 \times Msg \times Roles \times \mathcal{R}_{Ho}) \times (\mathcal{R}_{\mathcal{T}_0} \cup \mathcal{R}_\perp)$, for any $m \in Msg$, for $t' \in OSub(m)$, $setAt^{\mathbb{I}^\sigma}(V,t',m)$=$\{((t', m, A, \sigma(A)), \sigma(t')) \cup (V \setminus ((t', m, A, \sigma(A)), \perp)) \mid ((t', m, A, \sigma(A)), \perp) \in V\}$. Let $\sigma[v/t]$ be a substitution that uniformly maps only the atomic term t to the value v leaving all the other terms in the underlying structure unchanged. Then, for $A \in Ho$, $alice \in \mathcal{R}_{Ho}$ and $n \in \mathcal{R}_t$, an alternative definition for the denotation of the symbol $setAt$ is: $setAt^{\mathbb{I}^\sigma}(V,t,m) = \sigma[(n, m, A, alice)/(t, m, A, alice)](V)$, where $((t, m, A, alice), \perp) \in V$ and $m \in Msg$, $t \in OSub(m)$ and $\sigma(t) = n$ and $\sigma(A) = alice$.

  The denotation of $setAt(V,t,m)$ under a substitution σ is similar to the one of the previous symbol (i.e., $setMsg(V,t)$). Unlike in the case of $setMsg$, the structure V now has "holes" also for atoms corresponding to atomic subterms of t; these are "filled"/*set in* V according to $\sigma(t)$.

- for any $V \subseteq T_0 \times \mathcal{R}_{T_0}$, any $m \in Msg$, for $t' \in OSub(m)$, $recordAt^{\mathbb{I}^\sigma}(V,t',m)$ is the set-union $\{(t', \sigma(t'))\} \cup V$.

  The denotation of $recordAt(V,t',m)$ under σ is that atom-value pairs given by m and $\sigma(m)$ are added to V (i.e., *recording* the values of atoms within m into V).

- for any $V \subseteq \mathcal{T} \times \mathcal{R}_{\mathcal{T}}$, any $m \in Msg$, for $t' \in Sub(m)$, $recordPart^{\mathbb{I}^\sigma}(V,t',m)$ is the set-union $\{(t', \sigma(t'))\} \cup V$.

  The denotation of $recordPart(V,t',m)$ under σ is that non-atomic subterms t' of the substituted term m are added to V (i.e., *recording* into V the values of non-atomic parts within m).

- for any $m \in \mathcal{T}$,

$$analz^{\mathbb{I}^\sigma}(V_1, V_2, m) = \begin{cases} true & \\ and\ recordPart^{\mathbb{I}^\sigma}(V_2, t', m) & \\ and\ recordAt^{\mathbb{I}^\sigma}(V_1, t_0, m), & \\ \text{ with } t_0 \in OSub(Sub(t') \setminus ESub(t')) & \text{if } ((\, t' \in ESub(m), last(t') = i\,) \\ & \Rightarrow (\, OSub(\sigma(t'))_i \in V_1\,)) \\ & \\ false, & \text{if } ((t' \in ESub(m), last(t') = i) \\ & \text{ and } \ (\, OSub(\sigma(t'))_i \notin V_2\,)) \end{cases}$$

The denotation of $analz(V_1, V_2, m)$ under σ is as follows. If the values of keys for $\sigma(t')$ (i.e., $OSub(\sigma(t'))_i$) are found in V_2, then it means that the composite $\sigma(t')$ in m can be fully analysed (i.e., decrypted); the composite t' is recorded into V_2 and the atoms found un-encrypted, "in-plain" within $\sigma(t')$ (i.e., $t_0 \in OSub(Sub(t') \setminus ESub(t'))$) are added to the set V_1 (i.e, according to the denotations of *recordAt* and *recordPart*, respectively).

- for any $t \in \mathcal{T}$,

$$consistent^{\mathbb{I}^{\sigma[v/t]}}(t, V) = \begin{cases} true, & \text{if } v' \neq v'', \text{ for all } t', t'' \in Sub(t) \cap \mathcal{N},\ t' \neq t'' \text{ such that} \\ & \qquad (t', v'),\ (t'', v'') \in V \text{ and } \sigma[v'/t'],\ \sigma[v''/t''], \\ false, & \text{if } \exists t', t'' \in Sub(t) \cap \mathcal{N},\ t' \neq t'' \text{ for which } v' = v'' \\ & \text{when } (t', v'),\ (t'', v'') \in V \text{ and } \sigma[v'/t'], \sigma[v''/t''], \end{cases}$$

where $\sigma[v/t]$ denotes a substitution that uniformly maps atomic terms in t to values.

Using the $[|\cdot|]$ notation, $consistent^{\mathbb{I}^{\sigma[v/t]}}(t, V)$=
$[|(\forall t', t'' \in Sub(t) \cap \mathcal{N}, t' \neq t''\,) \Rightarrow (\,(t', \sigma(t')),\ (t'', \sigma(t'')) \in V \wedge \sigma(t') \neq \sigma(t'')\,)|]$.

The denotation of $consistent^{\mathbb{I}^\sigma}(t, V)$ stipulates that σ is a substitution which maps different nonces in t into different values drawn from V. If $consistent^{\mathbb{I}^\sigma}(t, V)$=*true*, the substitution σ is called *consistent*.

- for any $t \in \mathcal{T}$, for any $V \in \mathcal{T}_0 \times (\mathcal{R}_{\mathcal{T}_0} \cup \mathcal{R}_\perp)$, $V' \subseteq V$,

$$construct^{\mathbb{I}}(t,V,V') = \begin{cases} (t,\gamma[v/t_0](t)), & \text{if } (\forall t_0 \in Sub(t)\cap\mathcal{T}_0\,) \Rightarrow \\ & \quad (\,(t_0,v)\in V,\ v\neq\bot \text{ and } consistent^{\mathbb{I}^{\gamma}}(t,V') = true\,) \\ & \\ (t,\bot), & \text{if } (\ \exists t_0\in Sub(t)\cap\mathcal{T}_0, (t_0,\bot)\in V\ ) \\ & \quad \text{or} \\ & \text{if } (\ \forall t_0\in Sub(t)\cap\mathcal{T}_0, (t_0,v_0), v_0\neq\bot \\ & \quad \Rightarrow consistent^{\mathbb{I}^{\gamma[v_0,t_0]}}(t,V') = false\ ) \end{cases}$$

The denotation of *construct*(t,V,V') is as follows. If the map V has concrete values (i.e., not $\bot$) for all the atomic terms of t and these values uniformly constitute a consistent substitution γ for t (i.e., *consistent*$^{I^{\gamma}}(t,V')$=*true*), then the pair $(t,\gamma(t))$ is returned. Otherwise, the pair $(t,\bot)$ is returned.

- for any $t\in\mathcal{T}$, for any $V\in\mathcal{T}_0\times(\mathcal{R}_{\mathcal{T}_0}\cup\mathcal{R}_{\bot})$,

$$compose^{\mathbb{I}}(t,V) = \begin{cases} (t,\gamma[v/t_0](t)), & \text{if } (\ \forall t_0\in Sub(t)\cap\mathcal{T}_0\ ) \Rightarrow (\ (t_0,v)\in V\ ) \\ & \\ (t,\bot), & \text{if } (\exists t_0\in Sub(t)\cap\mathcal{T}_0) \text{ such that } (t_0,\bot)\in V \end{cases}$$

Like *construct*, *compose* denotes the composition of terms out of an existing map of atom-values. However, the composition required in *compose* is less strict than in the one stipulated by the symbol *construct* (i.e., the requirement of σ to be consistent is hereby dropped).

- for any $t\subseteq\mathcal{T}$, for any $V\subset(\mathcal{T}\times\mathcal{R}_{\mathcal{T}})$,

$$synth^{\mathbb{I}}(t,V) = \begin{cases} (t,\sigma[v/t'](t)), & \text{if } (\ \forall t'\in Sub(t))\ \Rightarrow \\ & (\ (t',v)\in V \text{ or } compose^{\mathbb{I}}(t',V|_{T_0}) = (t',v)\ ) \\ & \\ (t,\bot), & \text{if } \exists t'\in Sub(t) \text{ such that } (t',\bot)\in V \\ & \ \text{and}\, compose(t',V|_{T_0}) = (t',\bot) \end{cases}$$

The denotation of $synth(t,V)$ means that a value $\sigma(t)$ for t can be constructed from the map V if: either $\sigma(t)$ is composable part by part out of atoms found in V (i.e., *compose* interpreted for part t' and $V|_{T_0}$ returns *true*) or the values for such parts t' are already to be found in the map V (i.e., $(t', v) \in V$). Otherwise, the interpretation of the symbol returns $(t, \perp)$ to mean that t is not synthesisable from the map V. If $synth^{\mathbb{I}}(t, V) = (t, v)$ with $v \in \mathcal{R}_t$, $v \neq \perp$, then we say that the $[v/t]$ is *synthesisable from* V.

The interpretations $\mathbb{I}^{\sigma}$ and $\mathbb{I}$ of the symbols above will be made more intuitive in the context of specific maps V and an actual semantics for protocol executions. This will take place in following sections, as soon as the IS-based operational semantics of protocol executions is introduced. As a preamble, the term-value maps within the symbols will be given by parts of the local state of agents; the composition, setting, matching and analysis of values will be used to formalise messages being sent and received over the network under a Dolev-Yao thread [2].

3.3.2 Agents on the Signature $\mathcal{S}$ for $\varUpsilon_{IS}$

In this section we describe the agents in our multiagent system formalisation of security protocol executions.

Let Pr be a receiver-transparent (authentication) protocol, D be its `CAPSL` description, $\mathcal{S}$ be the sorted signature described in Section 3.1 underlying D and consider the set of terms, function and predicate symbols defined on $\mathcal{S}$. Let Σ_{Pr} be a set of role-substitutions for Pr, $X \in Ho$ and $\sigma(X\text{-}role) \in \Sigma_{Pr}$.

Definition 3.3.1 **(A^{σ}-agents)** *For a RT protocol Pr, for any $A \in Ho$, for any $\sigma(A\text{-}role) \in \Sigma_{Pr}$, an A^{σ}-agent corresponds to the homormorphic instantiation of the A-role under σ.*

Thus, an A^{σ}-agent designates a participant $\sigma(A)$ together with all the homomorphically instantiated objects associated to the *A-role* (e.g., its atoms, its A-rules etc.). Note that two distinct substitutions $\sigma_1(A\text{-}role), \sigma_2(A\text{-}role) \in \Sigma_{Pr}$ respectively correspond to two distinct agents: A^{σ_1}*-agent* and A^{σ_2}*-agent*. When σ is implicit, the notation *A-agent* replaces the notation A^{σ}*-agent*.

We write ag^{σ}_{A} to name a particular A^{σ}*-agent*, where $\sigma \in \Sigma_{Pr}$ is a fixed instantiation of an *A-role*, for $A \in Ho$. When the role-substitution σ is implicit, we simply write ag_A. The notation $\sigma(A\text{-}role) \mapsto ag_A$ is a diagrammatic way to denote the mapping of protocol roles into agents.

[2]We have recalled the Dolev-Yao thread model in Chapter 2, page 35. The interested reader is referred to [71].

The set Ag=$\{ag_1, \ldots, ag_n\}$ is the *set of agents* obtained as above (i.e., for each $i \in \{1, \ldots, n\}$, there exists $A \in Ho$ and $\sigma(A\text{-}role) \in \Sigma_{Pr}$ such that $\sigma(A\text{-}role) \mapsto ag_i$).

To depict these A-agents, we introduce some notions as follows.

Definition 3.3.2 (A-Stores) *For an RT protocol Pr, for any $A \in Ho$, an A-Store is the ordered list of typed atoms in $Atoms^A$ (i.e., the ordered multiset of typed atomic terms in the A-role).*

A unique order of this list for A can be derived, i.e., by sorting the list $Rules^A$ of rules by $Steps^A$, iterating the sorted list over and returning the atoms in the order that they appear in each message. An example for the *A-Store* built on (the signature for) the NSPK description is $A\text{-}Store_{NSPK}$, $A\text{-}Store_{NSPK}$=$(A\colon \mathcal{A}, B\colon \mathcal{A}, K_A^d\colon \mathcal{K}_1, K_B^e\colon \mathcal{K}_1, N_A\colon \mathcal{N}, N_B\colon \mathcal{N})$.

Remember that in Section 3.1 we introduced the notion of range. Below, we specialise this notion to formalise the values for a term t that an A-*agent* "considers" possible.

Definition 3.3.3 (Restricted Ranges for Atomic Terms) *Let ag_A be the agent corresponding to the instantiation of an A-role under some arbitrary $\sigma(A\text{-}role) \in \Sigma_{Pr}$, i.e., $\sigma(A\text{-}role) \mapsto ag_A$. For $t \in Atoms^A$, we write $\mathcal{R}_t^{ag_A}$ to denote* the range of atom t for ag_A*, as below:*

$$\begin{cases} \mathcal{R}_t^{ag_A} = \sigma(t), & \text{if } t \in OwnedAtoms^A \\ \mathcal{R}_t^{ag_A} = \mathcal{R}_t, & \text{if } t \in PublicData \cup LearnedAtoms^A \end{cases}$$

An alternative notation for $\mathcal{R}_t^{ag_A}$ is $\mathcal{R}_t^\sigma$, where $\sigma(A\text{-}role) \mapsto ag_A$. The reading of the notation $\mathcal{R}_t^\sigma$ is the *σ-restricted range of atom t.*

The definition of $\mathcal{R}^{ag_A}$ suggests that, in a protocol session, ag_A^σ will accept the values of the nonces and short-term keys in $OwnedAtoms^A$ only if they are compliant to the role-substitution σ; this is due to the fact that ag_A will have originated the values of this data in the first place, according to $\mathcal{S}$ and to the substitution σ. However, ag_A will accept the network-originated data (i.e., $t \in PublicData \cup LearnedAtoms^A$) irrespective of the role-substitution σ.

The set $\mathcal{R}^\sigma_{Atoms^A}$ denotes the set of all values used to map the atoms of an *A-role* under a substitution $\sigma \in \Sigma_{Pr}$. Formally, $\mathcal{R}^\sigma_{Atoms^A} = \bigcup\limits_{t \in Atoms^A} \mathcal{R}_t^\sigma$.

We use the above to give the definition of a *view* of an A^σ-*agent*.

Definition 3.3.4 (A$^\sigma$-view) *Let $A \in Ho$ and $\sigma(A\text{-}role) \in \Sigma_{Pr}$. A* view of an *$A^\sigma$-agent or,* an *$A^\sigma$-view, is a relation $A^\sigma\text{-}view \subseteq Atoms^A \times (\mathcal{R}^\sigma_{Atoms^A} \cup \mathcal{R}_\perp)$, $A^\sigma\text{-}view = \{(at, \sigma(at)), (at', v'), (at'', v'') \mid at \in OwnedAtoms^A, at' \in LearnedAtoms^A, at'' \in PublicData, v' \in \{\mathcal{R}_{at'} \cup \mathcal{R}_\perp\}, v'' \in \mathcal{R}_{at''}\}$.*

Whenever the substitution σ is implicit, we simply write *A-view* instead of *A^σ-view*.

Note that an *A^σ-view* can also be seen as an instantiation of an *A-store*, respecting a particular *A-role* substitution σ (i.e., σ maps atoms in $OwnedAtoms^A$ to precise values, learned terms at' can be set to any value in $\mathcal{R}_{at'} \cup \{\perp\}$ and long-term keys and principals can be set to any in their unrestricted range). For instance, ag_A could engage with any *B-role* party (i.e., $B \in PublicData$, so $(B, x) \in A^\sigma\text{-}view$, for any $x \in \mathcal{R}_A$), but ag_A's view is fixed as far as his own data is concerned (i.e., $\exists! z \in \mathcal{R}_{N_A}$, $z = \sigma(N_A)$ such that $(N_A, z) \in A^\sigma\text{-}view$).

Definition 3.3.5 (Possible A$^\sigma$-views) *Let $A \in Ho$, $\sigma(A\text{-}role) \in \Sigma_{Pr}$ and ag_A denote an A^σ-agent, i.e., $\sigma(A\text{-}role) \mapsto ag_A$. The* set $Views^{ag_A}$ *of all possible A^σ-views or the* set $Views^{ag_A}$ of all possible views for the agent ag_A *is given by:* $Views^{ag_A} = \bigcup_{v' \in \{\mathcal{R}_{at'} \cup \mathcal{R}_\perp\}, v'' \in \{\mathcal{R}_{at''}\}} \Big\{(at, \sigma(at)), (at', v'), (at'', v'') \mid at \in OwnedAtoms^A, at' \in LearnedAtoms^A, at'' \in PublicData\Big\}$.

Equivalently, $Views^{ag_A}$ represents all the possible instantiated *A-stores* respecting the application of a particular σ, $\sigma(A\text{-}role) \mapsto ag_A$.

Definition 3.3.6 (Possible A-Agents' Views) *The* set $Views^A$ of all possible views for *A-agents is given by:* $Views^A = \bigcup_{\sigma \in \Sigma_{Pr}, \sigma(A-role)} Views^{ag^\sigma_A}$*, where $A \in Ho$, $\sigma(A-role) \in \Sigma_{Pr}$ and $\sigma(A\text{-}role) \mapsto ag^\sigma_A$.*

The possible views of the *A-agents* show all the possible ways in which *A-stores* could be instantiated under all the possible role-substitutions $\sigma \in \Sigma_{Pr}$, given some protocol execution model. In other words, it shows all the possible ways in which the *A-role* could evolve under the set Σ_{Pr} of role-substitutions (considering that any *A-role* obeys the protocol, but executes in a malevolent environment).

We define *A-view*$_0$ as a special possible view of an instantiated *A-role* to denote the initialisation phase of this *A-role*. In an *A-view*$_0$ public data could be mapped to any value in its respective range; allowing any value for $ag^{\sigma}_{A}.view_0(B)$ symbolises that ag^{σ}_{A} does not have a pre-elected *B-role* partner of communication, as we do not assume authentication-of-origin [93]. All the atoms in $OwnedAtoms^A$ will be assigned to concrete values according to substitution σ, whereas atoms in $LearnedAtoms^A$ are mapped to $\perp$.

Remark 3.3.7 *In an actual implementation of the MAS model, in the initial states of ag^{σ}_{A} a public datum will not be assigned to a concrete value, the atoms in $OwnedAtoms^A$ will be assigned according to σ and the atoms in $Learned^A$ will be assigned to some specific values denoting null-data (as the symbol $\perp$ does in the above).*

Definition 3.3.8 ($\mathbf{A^{\sigma}}$-view$_\mathbf{0}$) *For an arbitrary $A \in Ho$, for $\sigma(A\text{-}role) \in \Sigma_{Pr}$, an* initial view of *an A^{σ}-agent is a relation $A^{\sigma}\text{-}view_0 \subseteq Atoms^A \times (\mathcal{R}^{\sigma}_{Atoms^A} \cup \mathcal{R}_{\perp})$, $A^{\sigma}\text{-}view_0 = \{(at, \sigma(at)), (at', \perp) \mid at \in OwnedAtoms^A, at' \in LearnedAtoms^A\}$.*

The sets of all possible initial views for an A^{σ}-agent and, in general, for *A-agents* can be given analogously to the case of possible non-initial views, shown in Definitions 3.3.5 and 3.3.6.

Let $A \in Ho$, $\sigma(A\text{-}role) \in \Sigma_{Pr}$ and $\sigma(A\text{-}role) \mapsto ag_A$. To complete the description of ag_A, we introduce the set $Steps^{ag_A}$ of *execution steps of agent ag_A*: $Steps^{ag_A} = Steps^A \cup \{last(Steps^A) + 1\}$.

3.3.2.1 Agents' Local States

We define the local states of agents using the notions of views and steps introduced above.

Definition 3.3.9 (Possible Local States for Agents) *Let $A \in Ho$, $\sigma(A\text{-}role) \in \Sigma_{Pr}$ and ag_A denote an A^{σ}-agent, i.e., $\sigma(A\text{-}role) \mapsto ag_A$. The set $L_{ag_A} \subseteq Steps^{ag_A} \times Views^{ag_A}$ is the* set of possible local states of ag_A.

Aligned with mainstream protocol approaches, a specific behaviour of every ag_A agent will be modelled in our formalisation. For instance, at each receiving step, several of ag_A's atoms in *PublicData* and in $LearnedAtoms^A$ will be assigned to concrete values, thus diminishing the number of ag_A's possible views. Therefore, the reachable local states of ag_A is, in general, a relatively small subset of the set $Steps^{ag_A} \times Views^{ag_A}$.

Let $i \in Steps^{ag_A}$ and $l \in L_{ag_A}$ such that $l = \langle i, view\rangle$, where $view \in Views^{ag_A}$ is possible at step i. We introduce the notation $l_{ag_A}@i$ to denote an arbitrary view of ag_A from those possible at the protocol step i, e.g., here, $l_{ag_A}@i = view$.

Definition 3.3.10 (Initial Local States of the Agents) *Let $A \in Ho$, $\sigma(A\text{-}role) \in \Sigma_{Pr}$, $\sigma(A\text{-}role) \mapsto ag_A$ and $view_0^{ag_A}$ be a possible initial view of ag_A. The* initial states of ag_A *are given by: $\langle i, view_0^{ag_A}\rangle = \langle i, (\,(at, \sigma(at)), (at', \perp)\,)\rangle$, where $i = first(Steps^{ag_A}), at \in OwnedAtoms^A$ and $at' \in LearnedAtoms^A$.*

For the NSPK protocol, one of the *possible* initial local states of the *A-role* agent ag_A is $i\_l = \langle 1, (\,(A, alice), (B, bob), (k_A, pvk_{alice}), (k_B, pbk_{bob}), (n_A, r_1), (n_B, \perp)\,)\rangle$. The state i_l shows one of the possibilities of ag_A "choosing" a communication partner, namely *bob*, from all B-role participants. Assume that the value $r_7 \in \mathcal{R}_{\mathcal{N}}$ for n_B is received by ag_A at step 3. Therefore, a *possible* local state of ag_A is l=$\langle 3, (\,(A, alice), (B, bob), (k_A, pvk_{alice}), (k_B, pbk_{bob}), (n_A, r_1), (n_B, r_7)\,)\rangle$.

Now, we introduce the local states of the Environment agent.

3.3.2.2 Environment's Local States

We express a Dolev-Yao intruder through the Environment agent. To model the local states of this agent, we first introduce the following maps and sets:

- a map *msg_log* from $SentMsg^A \times \{A^\sigma\text{-}agent \,|\sigma(A\text{-}role) \in \Sigma_{Pr}, A \in Ho\}$ to $\cup_{t \in SentMsg^A}\mathcal{R}_t \,\cup\, \mathcal{R}_\perp$;
- a map *atoms_log* from $Atoms^A \times SentMsg^A \times \{A^\sigma\text{-}agent \,|\sigma(A\text{-}role) \in \Sigma_{Pr}, A \in Ho\}$ to $\cup_{t' \in \{Sub(t)\cap\mathcal{T}_0 | t \in SentMsg^A\}}\mathcal{R}_{t'} \,\cup\, \mathcal{R}_\perp$.

Additionally we consider:

- a map *insider_vars* from $\cup_{A \in Ho}Atoms^A$ to $\cup_{t \in \mathcal{T}_0}\mathcal{R}_t$;
- a relation *agent_names* $\subseteq Ho \times \cup_{A \in Ho}\{\sigma(A) | \sigma \in \Sigma_{Pr}\}$ between principals and participants;
- for $V \subseteq \{\mathcal{N} \cup \mathcal{K}_0\}$, a map *values_log* $\subset (V \times \mathcal{R}_t)$ from atomic terms to values;

- for $V \subseteq \mathcal{T}$, a map $analz\_log \subset V \times \mathcal{R}_{\mathcal{T}}$ from terms to values;
- a map $synth\_log$ from Msg to $2^{\mathcal{R}_{Msg}}$;
- variables do_forge, do_analz, $stop\_analz$ of type $boolean$;
- constants $\sharp max\_SubstMsg$, $\sharp max\_SubstComposites$ of type *bounded integer*;
- variables $flag\_analz_1, \ldots, flag\_analz_{\sharp max\_SubstComposites}$ of type $boolean$; (to refer to $flag\_analz_i$, $i = \overline{1, \ldots, \sharp max\_SubstComposites}$, we sometimes simply use $flag\_analz$);
- variables $already\_analz_1, \ldots, already\_analz_{\sharp max\_SubstComposites}$ of type $boolean$; (to refer to $already\_analz_i$, $i = \overline{1, \ldots, \sharp max\_SubstComposites}$, we sometimes simply use $already\_analz$);
- variables $count_1$, $count_2$ of type *bounded integer*.

The map msg_log denotes that the intruder will keep track of each value of each message that an instance of an *A-role* (i.e., an A^{σ}-*agent*) sends, for all $A \in Ho$, for all $\sigma(A\text{-}role) \in \Sigma_{Pr}$. This is a formalisation of the Dolev-Yao interception of all messages.

The map $atoms\_log$ encodes a history of the communication (i.e., it stores the value of every atom in each message that any agent transmits).

The structure $insider\_vars$ formalises the fact that the intruder could be an insider acting as any (honest) role in the protocol *Pr* (i.e., through this structure the intruder is provided with at least one value for each atom in the protocol signature)[3].

The structure $agent\_names$ denotes the fact that the intruder keeps a record of all participants in the protocol. The structure captures the possibility that, in multi-session executions, there can be more than one instance of an *A-role* (i.e., for some $A \in Ho$, if $\sigma_1, \sigma_2 \in \Sigma_{Pr}$ with $\sigma_1(A) = alice$, $\sigma_2(A) = alice$ then both $(A, \sigma_1(A))$ and $(A, \sigma_2(A))$ appear in $agent\_names$).

The map $values\_log$ denotes the fact that the intruder will recall the values of the atomic data that he has learned throughout his analysis.

[3]Under a Dolev-Yao model, one atom is enough for the intruder to generate potentially the whole message space: e.g., $n_{\mathbf{I}}$ can generate $\{n_{\mathbf{I}}\}_{K_{\mathbf{I}}}$, $\{\{n_{\mathbf{I}}\}_{K_{\mathbf{I}}}\}_{K_{\mathbf{I}}}$, …, etc. However, for sake of clarity, we model an intruder with local variables for all atoms in the protocol (roles).

The map *analz_log* denotes the fact that the intruder will recall the parts of messages (i.e., composites) that he was able to decrypt/decompose throughout his analysis of learned data.

The map *synth_log* denotes the fact that the intruder will recall, for each symbolic message, the value-strings that he has synthesised throughout his execution.

The variable do_forge is a control variable used in order to reduce the number of times that the actions of Dolev-Yao compositions are executed. The variable *do_analz* is a control variable used in order to reduce the number of times that the actions of Dolev-Yao analyses of terms trigger. We will later show that this reduction is done in a non-restrictive way (i.e., the intruder agent still analyses and synthesises all the terms possibly attainable at a certain protocol stage, under a given bounded instantiation). Variables *stop_analz*, $flag\_analz$ and *already_analz* are auxiliary variables also used in the encoding of the Dolev-Yao analysis.

The variable $count_1$ acts in conjunction with variable *do_analz*. The variable $count_1$ records the number of trials of decomposing terms executed by the intruder at some reached state. The variable $count_1$ will be increasing until it reaches the value of the constant $\sharp max\_SubstComposites$. In Section 3.3.1 we introduced the *analz* operator. The counter $count_1$ will be used in a simulation of the fixpoint of the *analz* operator applied to a certain local state of the Environment.

The variable $count_2$ acts in conjunction with variable do_forge. The variable $count_2$ will be increasing until it reaches the value of the constant $\sharp max\_SubstMsg$. In Section 3.3.1 we introduced the *synth* operator. The counter $count_2$ will be used in the modelling of the intruder trying to derive all $[x/t]$ for a message $t \in Msg$ and a value $x \in \mathcal{R}_t$ by applying a *synth* operator to a certain local state of his.

Through the definitions of *msg_log* and *atoms_log* we note that, unlike in the case of the honest agents, the ranges of terms for the intruder are not restricted to particular substitutions. This models the Dolev-Yao behaviour, in that the intruder can try to synthesise messages over the entire permitted ranges and it does not expect to intercept or analyse any particular values.

Definition 3.3.11 (Possible Local States for the Environment) *A* possible local state of the Environment l_{Env} *is a tuple* l_{Env}=($msg\_log, atoms\_log, insider\_vars, agent\_names, values\_log,$ $analz\_log, synth\_log, do\_analz, stop\_analz, flag\_analz, already\_analz, , do\_forge, count_1, count_2,$

$\sharp max\_SubstMsg, \sharp max\_SubstComposite$), where each component is as defined above.

The *set L_{Env} of all possible local states of the Environment* is given by the union of all possible local states l_{Env} as above.

Definition 3.3.12 (Initial States of the Environment) *An* initial state of the Environment *$l_{0_{Env}}$ is given by the tuple $l_{0_{Env}} = (msg\_log^0, atoms\_log^0, insider\_vars^0, agent\_names^0, values\_log^0, analz\_log^0, synth\_log^0)$, where:*

- *$msg\_log^0(t, ag_A) = \perp$, for all $A \in Ho$, for all $t \in SentMsg^A$, for $\sigma(A\text{-}role) \in \Sigma_{Pr}$ and ag_A an A^σ-agent;*
- *$atoms\_log^0(t, m, ag_A) = \perp$, for all $t \in Sub(m) \cap \mathcal{T}_0$, for all $m \in SentMsg^A$, for all $A \in Ho$, for $\sigma(A\text{-}role) \in \Sigma_{Pr}$ and ag_A an A^σ-agent;*
- *$insider\_vars^0(t) = v$, for all $t \in OwnedAtoms^A$, for all $A \in Ho$, for some $v \in \mathcal{R}_t$;*
- *$agent\_names(A) = \bigcup_{\sigma(A-role)\in\Sigma_{Pr}} \{\sigma(A)\}$, for each $A \in Ho$;*
- *$values\_log^0 = insider\_vars^0$;*
- *$do\_forge := true$, $do\_analz := false$, $stop\_analz = true$, $flag\_analz = false$, $already\_analz = false$, $count_1 = 0$, $count_2 = 0$, $\sharp max\_SubstComposites$, $\sharp max\_SubstMsg$–as explained in Subsection 3.3.2.4.*

Table 3.1 shows an initial state of the intruder/Environment, under the NSPK instantiation given in Example 3.1.7. In Table 3.1, the values v_1, v_2, k_{greg} are arbitrarily chosen from their respective ranges to denote the initial data for N_A, N_B and $K_{\mathbf{I}}^d$ assigned to the Dolev-Yao insider. In the style of Remark 3.1.9, the range for the agent-names is large such that (groups of) honest agents cannot trivially infer that *greg*, the name assigned to the *id* of the intruder, is a culprit. The use of $\perp$ denotes, as before, that a concrete value for a certain symbolic term has not yet been acquired through the communication.

3.3.2.3 Agents' Local Actions

Recall that, in Section 3.3.2.1, we defined the range $\mathcal{R}_t^{ag_A}$ of values that ag_A considers "acceptable" for an atom $t \in Atoms^A$. Furthermore, in Section 3.1, the range $\mathcal{R}_t$ of a non-atomic term (e.g., a

• msg_log^0 :=	• $atoms\_log^0$:=	• $insider\_vars^0$:=	
$\langle\ \{A, Na\}_{Kb},\ ag_A^1\ \rangle\ :\ \bot$	$\langle\ A,\ \{A, Na\}_{Kb},\ ag_A^1\ \rangle\ :\ \bot$	$Na:\ v_1$	
$\langle\ \{A, Na\}_{Kb},\ ag_A^2\ \rangle\ :\ \bot$	$\langle\ Na,\ \{A, Na\}_{Kb},\ ag_A^1\ \rangle\ :\ \bot$	$Nb:\ v_2$	
$\langle\ \{Na, Nb\}_{Kb},\ ag_B^1\ \rangle\ :\ \bot$	$\langle\ A,\ \{A, Na\}_{Kb},\ ag_A^2\ \rangle\ :\ \bot$	$Ka:\ k_{greg}$	
$\langle\ \{Nb\}_{Ka},\ ag_A^1\ \rangle\ :\ \bot$	$\langle\ Na,\ \{A, Na\}_{Kb},\ ag_A^2\ \rangle\ :\ \bot$	$Kb:\ k_{greg}$	
$\langle\ \{Nb\}_{Ka},\ ag_A^2\ \rangle\ :\ \bot$	$\langle\ Na,\ \{Na, Nb\}_{Kb},\ ag_B^1\ \rangle\ :\ \bot$		
	$\langle\ Nb,\ \{Na, Nb\}_{Kb},\ ag_B^1\ \rangle\ :\ \bot$		
	$\langle\ Nb,\ \{Nb\}_{Ka},\ ag_A^1\ \rangle\ :\ \bot$		
	$\langle\ Nb,\ \{Nb\}_{Ka},\ ag_A^2\ \rangle\ :\ \bot$		
• $agent\_names$:=	• $values\_log^0$:=	• $analz\_log^0$	• $synth\_log^0$
$A\ :\ \sigma_1(A),\ A\ :\ \sigma_2(A)$	$\{v_1, v_2\}$	$\emptyset$	$\emptyset$
$B\ :\ \sigma_3(B)$			

Table 3.1 An Initial State For the Intruder in an NSPK Execution

message, a composite) was introduced as $\mathcal{R}_t = \prod_{t' \in OSub(t)} \mathcal{R}_{t'}$, for $t \in \mathcal{T} \setminus \mathcal{T}_0$. Now, we lift $\mathcal{R}_t^{ag_A}$ from atomic terms to messages, i.e., the range of values for a message t which are "acceptable" to ag_A.

Definition 3.3.13 (Restricted Ranges for Non-Atomic Terms) *Let $A \in Ho$, $\sigma(A\text{-}role) \in \Sigma_{Pr}$ and ag_A be an A^σ-agent, i.e., $\sigma(A\text{-}role) \mapsto ag_A$. Let $t \in \mathcal{T}$. The* range $\mathcal{R}_t^{ag_A}$ of the term t for ag_A *is the Cartesian product of the σ-restricted ranges of the ordered atoms t' in the term t:*
$\mathcal{R}_t^{ag_A} = \prod_{t' \in OSub(t)} \mathcal{R}_{t'}^{ag_A}$.

Definition 3.3.13 formalises the following fact: if the ranges of atoms are σ-restricted for an A^σ-*agent*, then the ranges of non-atomic terms for that A^σ-*agent* will be (homomorphically) restricted under σ too. For $t \in \mathcal{T}$ and $\sigma(A\text{-}role) \in \Sigma_{Pr}$, an alternative notation for $\mathcal{R}_t^{ag_A}$ is $\mathcal{R}_t^\sigma$; the reading for $\mathcal{R}_t^\sigma$ is *σ-restricted range of the term t*.

Let $A \in Ho$, $\sigma(A\text{-}role) \in \Sigma_{Pr}$ and ag_A be the A^σ-*agent* given by $\sigma(A\text{-}role) \mapsto ag_A$. We now proceed to introduce the actions of the agents in the model.

For each rule-term $r \in Rules^A$ of the form $r = i.A \to B : t$ (for some $B \in Ho$), we introduce the following set of actions for ag_A: $\{send(t, x) \mid t = t(r), x \in \mathcal{R}_t^{ag_A}\}$. It expresses the fact that the

A^σ-*agent* ag_A could potentially send any value for the message t within the σ-restricted range for t.

For the union of all rule-terms $r \in Rules^A$ of the form $r = i.B \to A : t$ (for some $B \in Ho$), we introduce a single action for ag_A called *receive*.

We also add the λ action for ag_A denoting the "empty" action and the *wait* action which is used for synchronisation purposes.

Definition 3.3.14 (Possible Local Actions for Agents) *Let $A \in Ho$, $\sigma(A\text{-}role) \in \Sigma_{Pr}$ and ag_A denote an A^σ-agent, i.e., $\sigma(A\text{-}role) \mapsto ag_A$. The* set Act_{ag_A} of all possible local actions of ag_A *is given by:*

$$\left\{ \bigcup_{t \in SentMsg^A} \{ \bigcup_{x \in \mathcal{R}_t^{ag_A}} send(t,x) \} \right\} \cup \left\{ receive \right\} \cup \left\{ wait \right\} \cup \left\{ \lambda \right\}.$$

Recall that Ag is the set of A^σ-*agents* for all $A \in Ho$, for all $\sigma(A\text{-}role) \in \Sigma_{Pr}$. The set $LAct = \bigcup_{ag \in Ag} Act_{ag_A}$ denotes as the *possible local actions of agents* (i.e., the possible local actions of A^σ-*agents*, for all $A \in Ho$, for all $\sigma(A\text{-}role) \in \Sigma_{Pr}$).

3.3.2.4 Environment's Local Actions

Now, we proceed with the Dolev-Yao actions in the present model.

The set $\cup_{t \in Msg}\{forge(t,v) | v \in \mathcal{R}_t\}$ is the set of *forging* actions of the Environment. For some $t \in Msg$ and $v \in \mathcal{R}_t$ arbitrary, an action of the form $forge(t,v)$ suggests that the intruder could potentially compose all value-strings v in $\mathcal{R}_t$ for message t and send them through the network.

The set $\cup_{t \in Composites}\{analz(t,v) | v \in \mathcal{R}_t\}$ is the set of *analysis* actions of the Environment. For some $t \in Composites$ and $v \in \mathcal{R}_t$ arbitrary, an action of the form $analz(t,v)$ suggests that the intruder will potentially try to analyse any value-string v for any composite t. The domain unfolded by the Dolev-Yao analysis is reasonable, since the intruder cannot anticipate what data the honest agents possess.

The set $\cup_{A \in Ho}\{intercept\_from(ag) | \sigma(A\text{-}role) \mapsto ag, \sigma(A\text{-}role) \in \Sigma_{Pr}\}$ is the set of *intercepting* actions as of the Enviroment. This means that the intruder will intercept all messages coming from any role-instance in the protocol execution.

The set $\{transmit\_to(ag,t,x)\}$ denotes actions of *transmitting* a message to agent ag, where

$A \in Ho$, $\sigma(A\text{-}role) \mapsto ag$, $\sigma(A\text{-}role) \in \Sigma_{Pr}$, $t \in ReceivedMsg^A$ and $x \in \mathcal{R}_t$. This set shows that the intruder is ready to insert any value-message to any role-instance.

Action λ of the Environment denotes that the intruder could potentially have no active involvement at some stage of the execution.

Definition 3.3.15 (Possible Local Actions for the Environment) *The* set Act_{Env} of all possible local actions for the *Environment is given as:*

$$\left\{ \bigcup_{t \in Msg} \{ \bigcup_{v \in \mathcal{R}_t} forge(t,v)\} \right\} \cup \left\{ \bigcup_{t \in Composites} \{ \bigcup_{v \in \mathcal{R}_t} analz(t,v)\} \right\} \cup$$
$$\left\{ \bigcup_{A \in Ho} \bigcup_{\sigma(A-role) \in \Sigma_{Pr}} \{intercept\_from(ag^{\sigma}_{A})\} \right\} \cup$$
$$\left\{ \bigcup_{A \in Ho} \bigcup_{\sigma(A-role) \in \Sigma_{Pr}} \{ \bigcup_{t \in ReceivedMsg^A} \{ \bigcup_{x \in \mathcal{R}_t} transmit\_to(ag^{\sigma}_{A}, t, x)\}\} \right\} \cup \left\{ \lambda \right\}.$$

3.3.2.5 Joint Actions

Recall that, at page 85, we introduced the set Ag=$\{ag_1, \ldots, ag_n\}$ of agents in this model (i.e., for each $i \in \{1, \ldots, n\}$, there exist $A \in Ho$ and $\sigma(A\text{-}role) \in \Sigma_{Pr}$ such that $\sigma(A\text{-}role) \mapsto ag_i$).

Definition 3.3.16 (Possible Joint Actions) *A* possible joint action $\overline{a}$ *is a tuple* $\overline{a}$=$(a_{ag_1}, \ldots a_{ag_n}, a_{Env})$*, where* $a_{ag_i} \in Act_{ag_i}$*,* $i = \overline{1,n}$ *and* $a_{Env} \in Act_{Env}$.

The *set of all possible joint actions* is denoted *Act*.

For a joint action $\overline{a} \in Act$ and for $ag_i \in Ag$, the notation $\overline{a}_{ag_i}$ denotes the local action of ag_i within the joint action $\overline{a}$, selected in a component-wise manner, i.e., $\overline{a}_{ag_i} = a_{ag_i}$ if $\overline{a} = (a_{ag_1}, \ldots a_{ag_n}, a_{Env})$ and $i = \overline{1,n}$.

We now proceed with the definition of the local protocol for the agents. Let $A \in Ho$, $\sigma \in \Sigma_{Pr}$ and ag_A be an agent such that $\sigma(A\text{-}role) \mapsto ag_A$.

3.3.2.6 The Local Protocol of Honest Agents

Let $l = \langle i, view \rangle$ be an arbitrary possible local state of ag_A and $view_0$ denote ag_A's initial view.

We define the local protocol of ag_A in the following way: $P_{ag_A}(l) = P_{ag_A}(\langle i, view\rangle) =$

$$\begin{cases} \text{for each } x \in \mathcal{R}_t^{ag_A} & \\ send(t, x), & \text{if } i = step(r) \text{ where there exists uniquely } r \in Rules^A, \\ & r = i.A \rightarrow B : t \text{ for some } B \in Ho \text{ and } construct^{\mathbb{I}}(t, view, view_0) = (t, x) \ \ (*) \\ \{receive,\ wait\}, & \text{if } i = step(r) \text{ where there exists uniquely } r \in Rules^A, \\ & r = i.B \rightarrow A : t, \text{ for some } B \in Ho \\ \lambda, & \text{if } i = last(Steps^A) + 1 \end{cases}$$

Let us develop $(*)$ above. Recall the denotational semantics, given in Section 3.3.1, for symbol *construct*. The agent ag_A can perform $send(t, x)$ whenever she can compose the value-string x for t from values in her current view. This means that ag_A "uses" x_i for t_i in her local view to "build" a substitution for t and this substitution maps nonces which ag_A owns into different values. Formally, the denotation $construct^{\mathbb{I}}$ returns *true* if it is possible to find a substitution $[x/t]$ out of $(t_i, x_i) \in view$ such that $consistent^{\mathbb{I}[x/t]}(t, view_0)$, i.e., the nonces in $OwnedAtoms^A$ and assigned in $view_0$ to concrete values are indeed mapped distinctively under $[x/t]$.
Let $t \in SentMsg^A$ and $\overline{t_1 t_2 \ldots t_n}$ be a representation of t as the ordered sequence of its atomic subterms (i.e., $t_i = OSub(t)_i$). According to the Definition 3.3.3 of $\mathcal{R}_t^{ag^\sigma_A}$, if the term t is sent out by ag^σ_A as $x \in \mathcal{R}_t^{ag^\sigma_A}$, then it follows that: *a).* $x_i = OSub(x)_i$ is $\sigma(t_i)$, for each $t_i \in OwnedAtoms^A$; *b).* x_i can take any value in $\mathcal{R}_{t_i}$, for each $t_i \in LearnedAtoms^A$. The agent ag_A could receive any value $x_i \in \mathcal{R}_{t_i}$ for $t_i \in LearnedAtoms^A$. For that, ag_A is equipped with all possible send actions determined by x_i ranging over the entire range $\mathcal{R}_{t_i}$, for $t_i \in LearnedAtoms^A$.

For one $j \in \{1, \ldots, n\}$, assume that $t_j \in LearnedAtoms^A$ and for all $i = \overline{1, n}, i \neq j$, assume that $t_i \in OwnedAtoms^A$. Moreover, assume that $\sigma(t_i) = a_i$. Then, $(*)$ expands to:

$$\begin{cases} \mathit{send}(\overline{t_1 t_2 \ldots t_{j-1} t_j t_{j+1} \ldots t_n}, \overline{a_1 a_2 \ldots a_{j-1} \mathbf{x_j^1} a_{j+1} \ldots a_n}), & \text{if } i = \mathit{step}(r) \text{ and } (t_1 = a_1 \text{ and} \\ & \ldots \text{ and } t_{j-1} = a_{j-1} \text{ and } t_j = x_j^1 \\ & \text{and } t_{j+1} = a_{j+1} \text{ and } t_n = a_n) \\ \mathit{send}(\overline{t_1 t_2 \ldots t_{j-1} t_j t_{j+1} \ldots t_n}, \overline{a_1 a_2 \ldots a_{j-1} \mathbf{x_j^2} a_{j+1} \ldots a_n}), & \text{if } i = \mathit{step}(r) \text{ and } (t_1 = a_1 \text{ and} \\ & \ldots \text{ and } t_{j-1} = a_{j-1} \text{ and } t_j = x_j^2 \\ & \text{and } t_{j+1} = a_{j+1} \text{ and } t_n = a_n) \\ \vdots & \\ \mathit{send}(\overline{t_1 t_2 \ldots t_{j-1} t_j t_{j+1} \ldots t_n}, \overline{a_1 a_2 \ldots a_{j-1} \mathbf{x_j^{|\mathcal{R}_{t_j}|}} a_{j+1} \ldots a_n}), & \text{if } i = \mathit{step}(r) \text{ and } (t_1 = a_1 \text{ and} \\ & \ldots \text{ and } t_{j-1} = a_{j-1} \text{ and } t_j = x_j^{|\mathcal{R}_{t_j}|} \\ & \text{and } t_{j+1} = a_{j+1} \text{ and } t_n = a_n) \end{cases}$$

Therefore, if only one atomic subterm t_j of t is bound to iterate all of the values in $\mathcal{R}_{t_j}$, then ag_A is equipped with $|\mathcal{R}_{t_j}|$ possible send actions at the local state $\langle i, \mathit{view}\rangle$ for some i such that $i.A \to B : t \in \mathit{Rules}^A$. It is easy to see that, in general, the problem of finding $[x/t]$ is akin to picking one ordered n-tuple over the generalised Cartesian product $\underset{i=1,\ldots,n}{\Pi} \mathcal{R}_{t_i}^{ag_A}$. Further, some constraints are to be applied to these n-tuples, i.e., for $[x/t]$ to be a consistent substitution. So, the entire space of $\underset{i=1,\ldots,n}{\Pi} \mathcal{R}_{t_i}^{ag_A}$ need not be unwound. However, in general the number of possibilities of forming a value-string for t out of the possible values in a view $l_{ag}@i$ of ag_A is of the order of $\underset{j=1,\ldots,n}{\Pi} |\mathcal{R}_{t_j}^{ag_A}|$. Note that for $t_i \in \mathit{OwnedAtoms}^A$, $|\mathcal{R}_{t_j}^{ag_A}| = 1$. Whilst this is promising, the unconstrained ranges $\mathcal{R}_{t_j}$ can be large.

The possible send actions of ag_A shown in expression $(*)$ can be given in an operational way as $O(\underset{j=1,\ldots,n}{\Pi} |\mathcal{R}_{t_j}^{ag_A}|)$ rules for sending out $t = \overline{t_1, \ldots, t_n}$ as the value-string x, composed from the current view. This alternative is written as follows:

$$\left\{ \frac{ag_A.t_1 = x_1 \wedge ag_A.t_2 = x_2 \wedge \ldots \wedge ag_A.t_n = x_n \wedge ag_A.i = \mathit{dueStep}}{\mathit{send}(\overline{t_1 t_2 \ldots t_n}, \overline{x_1 x_2 \ldots x_n})} (\mathit{rule}_j \quad) \right. \qquad (**),$$

where $j \in \{1, \ldots, O(\underset{j=1,\ldots,n}{\Pi} |\mathcal{R}_{t_j}^{ag_A}|)\}$, $i \in \mathit{Steps}^A$, $r \in \mathit{Rules}^A$, $r = i.A \to B : t$, $x_k \in \mathcal{R}_{t_k}^{ag^A}$, $x_k \in l_{ag_A}@i$ and $k \in \{1, \ldots, n\}$.

In relation with $(*)$ above, we prove:

Lemma 3.3.17 *Let $A \in Ho$, $\sigma(A\text{-}role) \in \Sigma_{Pr}$ and $ag_A \in Ag$ correspond to $\sigma(A\text{-}role)$. Let $r \in Rules^A$, $r = i.A \to B : t$ (for some $B \in Ho$), $l \in L_{ag_A}$ with $l = \langle i, view\rangle$ and $view_0$ be the initial view of ag_A. Then, there exists $v \in \mathcal{R}_t^{ag_A}$ such that $construct^{\mathbb{I}^\sigma}(t, view, view_0) = (t, v)$.*

Lemma 3.3.17 stipulates the following fact. Whenever an agent ag_A has reached the step where the composition of t is due, the agent ag_A can perform the composition of t as some $v \in \mathcal{R}_t^{ag_A}$. In other words, at least one of the actions in $\cup_{x \in \mathcal{R}_t^{ag_A}} \{send(t, x) | (r = i.A \to B : t) \in Rules^A\}$ of agent ag_A will be enabled at state $l \in L_{ag_A}$, where $A \in Ho$, $\sigma \in \Sigma_{Pr}$, $\sigma(A\text{-}role) \mapsto ag_A$, ag_A in state $l = \langle i, view\rangle$, $view \in Views^{ag_A}$ and $view_0 \in Views_0^{ag_A}$.

The proof of Lemma 3.3.17 makes uses of the definition of the global evolution function which is given later. Therefore, it is given in Appendix A, Section A.1.

Having given the agents' local protocol function, we proceed to give the Environment's local protocol function.

3.3.2.7 Environment Local Protocol

Let l_{Env}=$(msg\_log, atoms\_log, insider\_vars, agent\_names, values\_log, analz\_log, synth\_log)$ be a local state of the environment. We define the local protocol of the Environment in the following way: $P_{Env}(l_{Env}) = Act_{Env}$. This symbolises that at any of the Environment's local state any of his actions is possible.

3.3.2.8 Agents' Local Evolution

Let $l = \langle i, view\rangle$ be a local state of ag_A, $view_0$ denote an initial view of ag_A and $\overline{a}$ be a joint action. The local evolution function of ag_A can be given under the denotation: $[|E_{ag_A}|] = if\ (preconditions(l, \overline{a}) = true)\ then\ (postconditions(l, \overline{a}))$, for arbitrary $l \in L_{ag_A}$ and $\overline{a} \in P_{ag_A}(l)$. Thus, we follow this approach in defining the local evolution function E_{ag_A} of agent ag_A.

1. for $\overline{a}_{ag_A} = receive$ and $\overline{a}_{Env} = transmit\_to(ag_A, t, x)$, where $t \in ReceivedMsg^A$ and $x \in \mathcal{R}_t$:

Let $\rho = \rho[x^{'}/t^{'}]$, where $t^{'} = OSub(t)_i$, $x^{'} = OSub(x)_i$, $Ind = \{1, \ldots, last(OSub(t))\}$ and $i \in Ind$ (i.e., ρ is equal to the substitution $[x/t]$, where t is a message expected by ag_A).

$$preconditions(l, \overline{a}) : \begin{cases} (1).\, i\, \in Steps^A \cap \{step(r) | r = j.B \rightarrow A : t\},\ for\ some\, B \in Ho,\ j \in Steps \\ (2).\, in\_match^{\mathbb{I}^{\rho}}(t, k, k') = true \text{ (for denominated } (k, k') \in Ind \times Ind) \\ (3).\, out\_match^{\mathbb{I}^{\rho}}(view, t) = true \\ (4).\, decryptable^{\mathbb{I}^{\rho}}(view, t) = true \end{cases}$$

$$postconditions(l, \overline{a}) \text{:} \begin{cases} (5).\, i \leftarrow next(Steps^A, i) \\ (6).\, set^{\mathbb{I}^{\rho}}(view, t) \end{cases}$$

2. for $\overline{a}_{ag_A} = send(t, x)$ and $\overline{a}_{Env} = intercept\_from(ag)$, where $t \in SentMsg^A$ and $x \in \mathcal{R}_t^{ag_A}$:

$$preconditions(l, \overline{a}) \text{:} \begin{cases} (1).\, i\, \in Steps^A \cap \{step(r) | r = A \rightarrow B : t\},\ for\ some\, B \in Ho \\ (2).\, construct^{\mathbb{I}}(t, view, view_0) = (t, x),\ for\ some\, x \in \mathcal{R}_t^{ag_A} \end{cases}$$

$$postconditions(l, \overline{a}) \text{:} \begin{cases} (3).\, i \leftarrow next(Steps^A, i) \end{cases}$$

3. $\overline{a}_{ag_A} = wait$: $preconditions(l, \overline{a})$: $\emptyset$, $postconditions(l, \overline{a})$: $\emptyset$

4. $\overline{a}_{ag_A} = \lambda$: $preconditions(l, \overline{a})$: $\emptyset$, $postconditions(l, \overline{a})$: $\emptyset$

Note that here we discuss those cases in the evolution function defined solely by the components $\overline{a}_{ag_A}$ and $\overline{a}_{Env}$ of a joint action $\overline{a}$, for some $A \in Ho$. The synchronisation between ag_A and some agent ag_B of *B-role* ($A \neq B$) will emerge from further definitions, i.e., from the definition of the local evolution of the Environment agent.

From the definition of E_{ag_A} above, observe that the acceptance of messages depends on the preconditions within E_{ag_A}. These preconditions are built using the denotations of function and predicate symbols in Section 3.3.1. The denotation of evolution entry 1 says that ag_A has to be in the step due for receiving t (hence, 1.1) and the value x sent as t has to be decryptable by ag_A (hence, 1.4). According to the denotation of *decryptable* in Section 3.3.1, entry 1.4 requires ag_A to have the keys associated with $[x/t]$. Furthermore, the value x sent as t has to be syntactically correct (hence, 1.2 requires *in_match* to hold). We recall from Section 3.3.1 that *in_match* requires certain subparts of x (i.e., those at indices k and k') to have the same values in order to obey the pattern of t. Moreover, the value x sent as t has to be consistent with the execution carried by ag_A so far (hence, 1.3 requires *out_match* to be *true* under $[x/t]$ at the current *view* of ag_A). Again, by Section 3.3.1, $out\_match^{\mathbb{I}}$ demands that certain subterms t_i of t are sent to ag_A as x_i, if x_i is the value that ag_A has previously stored for t_i in her *views*. The denotation implied by *out_match* in the local evolution E_{ag_A} is a representation in our MAS approach of the *matching-receive* [175] semantics adopted by all standard lines of security verification, e.g., [9, 19, 188, 73].

Upon the action of receiving and validation of all the above, agent ag_A assigns atoms in her view to values in the newly received value-string x (hence, 1.6) and moves into her next execution step (hence, 1.5).

The denotation of evolution entry 2 is as follows. If agent ag_A has reached the protocol-step in which she is due to send t (hence, 2.1) and ag_A can construct the value-string x according to the pattern of t (hence, in 2.2, ag_A has set a correct value for each subterm of t in her current view), then ag_A sends value x for t and moves to the next execution step. Entry 2 also stipulates that whenever ag_A sends a value-string x for an expected message t, the intruder/Environment intercepts that value.

Having given the agents' local evolution function, we proceed to give the Environment's local evolution function.

3.3.2.9 Environment's Local Evolution

Let l_{Env}=($msg\_log, atoms\_log, insider\_vars, agent\_names, values\_log, analz\_log, synth\_log,$ $do\_forge, do\_analz, flag\_analz, already\_analz, stop\_analz, count_1, count_2, \sharp max\_SubstComposites,$ $\sharp max\_SubstMsg$) be an arbitrary possible local state of the Environment.

The set $POOL = insider\_vars \cup agent\_names \cup values\_log$ denotes the union of some of the intruder's data-sets, a selection of data from the local state l_{Env}. Then, the range of the set $POOL$ of terms under the set of substitutions is given by $\mathcal{R}_{POOL}$=$\mathcal{R}_{insider\_vars \cup agent\_names \cup values\_log}$.

The constant $\sharp max\_SubstMsg = \sum_{t \in Msg} |\mathcal{R}_t|$ denotes the number of all possible value-strings that the Environment could compose for messages, where $|\mathcal{R}_t|$ is the cardinality of the (unrestricted) range for a message t. Recall that the Environment operates on unrestricted ranges $\mathcal{R}_{t_0}$ for atomic terms t_0 and under generalised Cartesian products $\mathcal{R}_t = \prod\limits_{t_0 \in OSubt(t)} \mathcal{R}_{t_0}$ over these unrestricted ranges for non-atomic terms t. Then, under the set of substitutions considered for the protocol, we can similarly infer the number $\sharp max\_SubstComposites$ of all possible value-strings that the Environment could find by analysing values for composites, i.e., $\sharp max\_SubstComposites = \sum\limits_{t \in Composites} |\mathcal{R}_t|$.

Let $\overline{a}$ be a joint action, $l \in L_{Env}$ and $\sigma(A\text{-}role) \mapsto ag_A$, for some $\sigma \in \Sigma_{Pr}$, $A \in Ho$. To define the local evolution function E_{Env} of the Environment, we give the description of $E_{Env}(l_{Env}, \overline{a})$:

1. for $\overline{a}_{ag_A} = send(t,x)$ and $\overline{a}_{Env} = intercept\_from(ag_A)$, where $t \in SentMsg^A$ and $x \in \mathcal{R}_t^{ag_A}$:

   $preconditions(l_{Env}, \overline{a})$: $none$

   $postconditions(l_{Env}, \overline{a})$:
   $$\begin{cases} (1.) setMsg^{\mathbb{I}^\sigma}(msg\_log, t) \\ (2.) setAt^{\mathbb{I}^\sigma}(atoms\_log, t) \\ (3.) analz^{\mathbb{I}}(values\_log, analz\_log, t) \\ (4.)(do\_analz := true) \text{ and } (count_1 := 0) \end{cases}$$

2. for $t' \in Composites$, $x' \in \mathcal{R}_{t'}$, $\overline{a}_{Env} = analz(t', x')$:

   $preconditions(l_{Env}, \overline{a})$:
   $$\begin{cases} (1.)(do\_analz = true) \text{ and } (count_1 < \sharp maxSubstComposites) \\ (2.)(t', x') \in analz\_log \end{cases}$$
   $postconditions(l_{Env}, \overline{a})$:
   $$\begin{cases} (3.) flag\_analz[count_1] := analz^{\mathbb{I}}(values\_log, analz\_log, t', x') \\ (4.) count_1 := count_1 + 1; \end{cases}$$

3. for $t' \in Composites$, $x' \in \mathcal{R}_{t'}$, $\overline{a}_{Env} = analz(t', x')$:

   $preconditions(l_{Env}, \overline{a})$:
   $$\begin{cases} (1.)(do\_analz = true) \text{ and } (count_1 < \sharp maxSubstComposites) \\ (2.)(t', x') \notin analz\_log \end{cases}$$
   $postconditions(l_{Env}, \overline{a})$:

$$\begin{cases}(3.)flag\_analz[count_1] := false;\\ (4.)count_1 := count_1 + 1;\end{cases}$$

4. if $(count_1 = \sharp maxSubstComposites$ and $do\_analz = true$ and $(\forall i \in \overline{1, \sharp maxSubstComposites})(flag\_analz[i] = false))$ $then$ $(stop\_analz := true)$;

5. if $(count_1 = \sharp maxSubstComposites$ and $do\_analz = true$ and $(\exists i \in \overline{1, \sharp maxSubstComposites})(flag\_analz[i] = true$ and $already\_analz[i] = false))$ $then$ $(count_1 := 0)$ and $(already\_analz[i] := true)$;

6. if $(count_1 = \sharp maxSubstComposites$ and $do\_analz = true$ and $(\forall i \in \overline{1, \sharp maxSubstComposites})(already\_analz[i] = true))$ $then$ $(stop\_analz := true)$;

7. if $(stop\_analz = true)$ $then$ $(do\_forge := true)$

8. $\overline{a}_{Env} = forge(t, x)$:

   $preconditions(l_{Env}, \overline{a})$:

   (1).$(do\_forge = true)$ and $(count_2 < \sharp maxSubstMsg)$

   (2).$synth^{\mathbb{I}}(t, POOL) = (t, x)$, for current $x \in \mathcal{R}_t$

   $postconditions(l_{Env}, \overline{a})$:

   (3).$synth\_log := synth\_log \cup (t, x)$

   (4).$count_2 := count_2 + 1$

9. for $t \in Msg$, $x \in \mathcal{R}_t$, $\overline{a}_{Env} = forge(t, x)$:

   $preconditions(l_{Env}, \overline{a})$:

   (1).$(do\_forge = true)$ and $(count_2 < \sharp maxSubstMsg)$

   (2).$synth^{\mathbb{I}}(t, POOL) \neq (t, x)$, for current $x \in \mathcal{R}_t$

   $postconditions(l_{Env}, \overline{a})$:

   (3).$count_2 := count_2 + 1$

10. if $(count_2 = \sharp maxSubstMsg)$ $then$ $(do\_forge := false)$;

11. $\overline{a}_{ag_A} = receive$ and $\overline{a}_{Env} = transmit\_to(ag, t, x)$, where $t \in ReceivedMsg^A$, $x \in \mathcal{R}_t$,:

    $preconditions(l_{Env}, \overline{a})$:

    (1).$(t, x) \in synth\_log$ or $\exists ag' \in Ag, ag' \neq ag,\ s.that\ msg\_log(t, ag') = x$

    $postconditions(l_{Env}, \overline{a})$: $void$

We are now going to explain the local evolution E_{Env} of the Environment agent. Recall that we assumed an arbitrary $A \in Ho$, $\sigma(A\text{-}role) \in \Sigma_{Pr}$, an agent $ag_A \in Ag$ corresponding to $\sigma(A\text{-}role)$, $i \in Steps^{ag_A}$ a step of ag_A, $l = \langle i, view\rangle \in L_{ag_A}$ and $\overline{a} \in Act$ a joint action. The evolution-case 1 captures the synchronisation between ag_A sending the value-string x for t and the intruder/Environment intercepting it. For an intruder to intercept a message, there are no preconditions. As postconditions, the Environment recalls the action by storing data about the message and its parts (hence, postconditions 1.1, 1.2). After the interception of $[x/t]$, the intruder analyses the composite $[x/t]$ (hence, post-condition 1.3). In fact, the intruder enables an entire analysis cycle (hence, the post-condition 1.4). This forces the evolution-case 2. The intruder tries to analyse any composites $[x'/t']$ as some might have become intelligible given the new interception of $[x/t]$. There are entries in this evolution-case for analysing (t', x'), for each $t' \in Composite$, for each $x' \in \mathcal{R}_{Composites}$, i.e., the intruder could potentially analyse every possible value under the given protocol signature and instantiation. However, given a protocol instantiation and the proceedings of roles, not all such analyses would be executed at a given state. Recall the denotational semantics of symbol *analz*: when the decryptions of composites in $[x/t]$ are possible, the intruder records decrypted non-atomic parts of $[x/t]$ in his *analz_log* and un-encrypted, "in-plain" atoms of $[x/t]$ in *values_log*. Whilst there are still value-composites in his possesion (i.e., pre-condition 2.2) that could be analysed (i.e., $count_1 < \sharp maxSubstComposites$ in pre-condition 2.1) the intruder tries to analyse them (as per post-condition 2.3). The attempted is counted (i.e., post-condition 2.4). If a composite is not is his possesion (pre-condition 3.2), he signals that he cannot analyse it (post-condition 3.3) and he records his unsuccessful trial (post-condition 3.4). Following the denotation of the symbol *analz*, at each entry 2.3 the knowledge-sets of *analz_log* and *values_log* can potentially grow, i.e., if *analz* is successful for some value-composite $[x'/t']$ under the newly acquired data. If there has been no successful analysis under a full *analz*-cycle (e.g., $flag\_analz$ is constantly

$false$ in the pre-condition of evolution-case 4), then evolution-case 4 enforces that *stop_analz* is set to *true*. If a composite has been newly analysed in the current cycle (i.e., pre-condition $flag\_analz[i] = true$ and $already\_analz[i] = false$) in the evolution-case 5), then the fact that composite i is now learned is recorded (i.e., post-condition $already\_analz[i] := true$ in the evolution-case 5). If all that can be analysed at this cycle has been analysed previously (i.e., *already_analz* is constantly *true*), evolution-case 6 enforces that *stop_analz* is set to *true*. Otherwise, if at least one new analysis has succeeded (i.e., preconditions of the evolution-case 5), the counter $count_1$ is reset and a new *analz*-cycle starts. Thus, evolution-cases 4–6 denote the closure of knowledge-sets under the operator $analz^{\mathbb{I}}$, after each interception-step. The analysis cycles 2–6 can be written procedurally as:

```
//initialisation of the model
stop_analz:=false;
for(i:=0;i<#maxSubstComposites;i++)
          flag_analz[i]:=false;
          already_analz[i]:=false;
endfor

//Dolev-Yao analysis cycles in the model
label l1:
while(!stop_analz)
  count1:=0;
  foreach [x'/t']
     if([x'/t'] in analz_log)
          flag_analz[count_1]:=analz(values_log, analz_log,x');
          count_1++;
     else if([x'/t'] not in analz_log)
          flag_analz[count_1]:=false;
          count_1++;
     endif
  endfor
  noNewAnalz_counter:=0;
  allAnalz_counter:=0;
  for(i:=0;i<#maxSubstComposites;i++)
        if (flag_analz[i]==false)
          noNewAnalz_counter++;
        else if (flag_analz[i]==true and already_analz[i]==false)
          already_analz[i]:=true;
          continue l1;
        else if(flag_analz[i]==true and already_analz[i]==true)
          allAnalz_counter++;
        endif
  endfor
  if(noNewAnalz_counter==#maxSubstComposites)
      stop_analz:=true;
  if(allAnalz_counter==#maxSubstComposites)
      stop_analz:=true;
  endif
endwhile
```

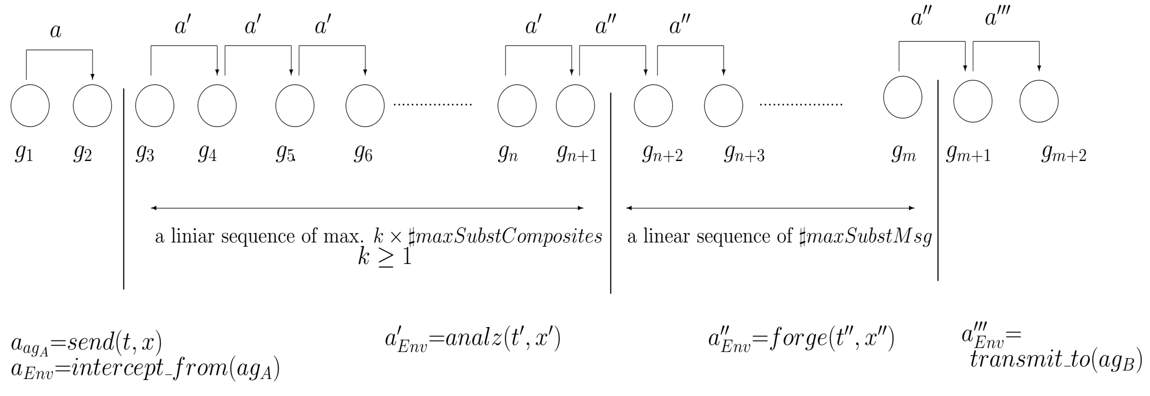
Appendix A contains details of this procedure and a proof the fact that closure of some local state l_{Env} of the Environment under *analz* operations in enforced in the model (see Section A.1.*II* and Lemma A.1.1).

According to the evolution-case 7, as soon as the Dolev-Yao analysis stops (i.e., $stop\_analz = true$) the forging can begin, i.e., $do\_forge := true$. If a trial of synthesising terms at a given state is successful (hence, evolution-case 8), the newly synthesised term is taken into account (hence, post-condition 8.4, $count_2 := count_2 + 1$). Conversely, if the trial of synthesising terms is unsuccessful, the trial is still counted (hence, post-condition 9.3, $count_2 := count_2 + 1$). The latter is done in order to dismiss this trial as a future *synth*-attempt at the current local state. When all the possibilities are exhausted, the Environment stops trying to compose new messages (hence, the precondition $count_2 = \sharp maxSubstMsg$ of evolution-case 10 forces do_forge to be set to $false$, therefore triggering no more evolution steps through evolution-cases 8 and/or 9). However, at a given state, not all $\sharp maxSubstMsg$ forging actions would necessarily be executed.

Function *synth* is applied over the knowledge-set $POOL$. It denotes that $[x/t]$ is composable from atoms in $values\_log$ and non-atomic terms stored in $analz\_log$ (hence, the data-sets in $POOL$). The previous analysis parts of the Environment's local evolution forced $analz\_log$ and $values\_log$ to be closed under *analz*-type actions. Thus, $forge$ and $analz$ actions as enforced by E_{Env} at a local state l_E give $synthesis(closure\_analysis(l_E.POOL))$ (upon the traditional sense [163] and with respect to a protocol description and instantiation).

The intruder will inject a message into the network if he has synthesised it (i.e., the first part of the precondition of evolution-case 11) or he has previously intercepted it (i.e., the second part of the precondition of evolution-case 11). From the local evolution function $E_{ag_A}(l, \overline{a})$, recall that the action *receive* does not trigger for ag_A unless it is synchronised with a *transmit* local action of the Environment and some validation tests on the local state of ag_A are successful. Agent ag_A will "wait" (i.e., perform the action *wait*) at all local states that contain a "receive-due" step until the intruder "transmits" her a message.

Remark 3.3.18 *The evolution functions presented imply a certain order over the application of joint actions in this modelling. A joint action $\overline{a_1}$ of type $\overline{a_1}_{ag_A} = send(t, x), \overline{a_1}_{Env} = intercept\_from(ag_A)$ is always followed by a series of up to $k \times \sharp maxSubstComposites$ joint actions $\overline{a_2}$ of type $\overline{a_2}_{Env} = analz(t^{'}, x')$ (k a constant, $k \geq 1$). This sequence of analz actions is followed by a series of $\sharp maxSubstMsg$ joint actions $\overline{a_3}$ of type $\overline{a_3}_{Env} = synth(t^{''}, x'')$. Figure 3.1 depicts these facts.*



a
a'
a'
a'
a'
a''
a''
a''
a'''
g_1
g_2
g_3
g_4
g_5
g_6
g_n
g_{n+1}
g_{n+2}
g_{n+3}
g_m
g_{m+1}
g_{m+2}
a liniar sequence of max. $k \times \sharp maxSubstComposites$ $k \geq 1$
a linear sequence of $\sharp maxSubstMsg$
a_{ag_A}=$send(t, x)$
a_{Env}=$intercept\_from(ag_A)$
a'_{Env}=$analz(t', x')$
a''_{Env}=$forge(t'', x'')$
a'''_{Env}= $transmit\_to(ag_B)$

Figure 3.1 The Sequence of Linearly Applied (Dolev-Yao) Actions in Υ_{IS}

Even if $\sharp max\_SubstComposites$, $\sharp max\_SubstMsg$ are of a double-exponential factor in the size of the protocol description, the above technique of enforcing all the *analz* and *forge* actions in the local evolution of the Environment at a certain due state yields a smaller state-space than allowing *analz* and/or *forge* to possibly trigger at every state of the Environment. Also, many of the possible *analz* and *forge* actions, included as above in the local evolution of the Environment at a generic current state, will not trigger at that state for a given instantiation and its homomorphic application over the global evolution function.

3.3.2.10 The MAS Formalism for Pr: Υ_{IS}^{Pr}

Based on the definition of the local evolution functions for the agents and for the Environment, we can now define the moves of the interpreted system model depicted so far.

Definition 3.3.19 (The Global Evolution Function) *The* global evolution function *is a function $E : G \times Act \to G$ such that $E(g, \overline{a}) = g^{'}$ if and only if $E_{ag}(g_{ag}, \overline{a}) = g^{'}_{ag}$ for all $ag \in Ag \cup Environment$, where $g, g' \in G$ and $\overline{a} \in Act$.*

Definition 3.3.20 (The IS-based Formalism for Pr) *Let Pr be a receiver-transparent protocol described by a `CAPSL` description D and a corresponding signature $\mathcal{S}$. Let Σ_{Pr} be a set of role-substitutions. Let Ag be the set of A^{σ}-agents, for all for $A \in Ho$, for all $\sigma(A\text{-}role) \in \Sigma_{Pr}$ and Env be the Environment agent as above. Let G be the set of global states, $I \subseteq G$ be the set of initial states, $P = \{P_i, P_{Env} \,|\, i \in Ag\}$ be the set of protocol functions and E be the evolution function as modelled before. Consider at least the atomic propositions PV implied by the predicate symbols corresponding to the `CAPSL` goals. Let $V : PV \times G \to \{true, false\}$ be a valuation function. The tuple $\Upsilon_{IS}^{Pr} = (G, I, P, E, V)$ is the* IS-based formalism for the multi-session execution of *Pr, rendered by* the set of substitutions *Σ_{Pr}.*

When Pr is implicit, we simply write Υ_{IS}.

We are now going to introduce the model unwound by the IS-based formalism Υ_{IS}^{Pr}.

Let $n \geq 1$, $1 \leq i \leq n$, $\overline{a}_i$ be a joint action and $\overline{a}_1 \dots \overline{a}_n$ be a sequence of actions such that $E(g_{i-1}, a_i) = g_i$, for $g_0 \in I$ and $g_i \in G$, $i = \overline{1, n}$. Then, $g_0\overline{a}_1 g_1 \dots \overline{a}_n g_n$ describes a *computation* (or a *run*) implied by Υ_{IS}. Starting from the initial states, the computations designate the *reachable* states.

For Pr an RT protocol, the local-indistinguishability[4] relation $\approx_i \subseteq L_i \times L_i$ in Υ_{IS}^{Pr} is the equality relation $=$, for any $i \in Ag$. Note that this is a reasonable choice for an indistinguishability relation, as in the models for RT protocols the local states contain only pairs of the form "(atomic-term, value)", i.e., the local states contain do not contain pairs of the form "(encrypted-term, value)". The local-indistinguishability for the Environment agent is also assimilated to the simple equality

[4]See Chapter 2, page 25 for details.

relation. This relation is lifted to an indistinguishability relation over global states $\sim_i \subseteq G \times G$ in a standard way: $g \sim_i g'$ if and only if $g_i \approx_i g'_i$, for any $g, g' \in G$, $i \in Ag \cup \{Env\}$.

Definition 3.3.21 (The MAS System Model for Pr) *Let Pr be a receiver-transparent protocol described by a* `CAPSL` *description D and a corresponding signature $\mathcal{S}$. Let Σ_{Pr} be an arbitrary set of role-substitutions render a multi-session execution of Pr. Let $\Upsilon_{IS}^{Pr} = (G, I, P, E, V)$ be the IS-based formalism for this multi-session execution of Pr, with G, I, P, E, V as before, $Ag = \{1, \ldots, n\}$, $G' \subseteq G$ the set of reachable states implied by Υ_{IS}^{Pr} and $\sim_i \subseteq G' \times G'$ the indistinguishability relations described above. Then, the tuple $M_{IS}^{Pr} = (G', \sim_1, \sim_2, \ldots \sim_n, \sim_{Env}, V)$ is the* MAS System Model for the arbitrary multi-session execution *of Pr.*

When Pr is implicit, we simply write M_{IS}.

To sum up, Section 3.3 has introduced Υ_{IS}, an IS-based formalisation for multi-session execution of RT security protocols. Also, Section 3.3 introduced M_{IS} as the unwound model implied by this specialised interpreted system. In a sense, the M_{IS} model exhibits the branching of the multi-session execution of RT security protocols. We proceed with the expression of the security requirements in CTLK, the underlying logic language of the Υ_{IS} model. These are interpreted the M_{IS} model, in the same sense that Chapter 2 recalled the usual satisfaction of CTLK on interpreted systems.

3.3.3 Security-Requirements Expressed in the Model Υ_{IS}

Remember that in Section 3.2 we formalised and specialised `CAPSL` goals into atomic and complex. Having now introduced the Υ_{IS} model, we continue that formalisation.

For atomic `CAPSL` assertions we add corresponding predicate symbols to the signature $\mathcal{S}$. We thus produce a logically-extended signature $\mathcal{S}_L$. For the atomic `CAPSL` assertions `AGREE` and `HOLDS` we add the predicate symbol *Agree* and the predicate symbol *Holds*, respectively. The predicate *Agree* operates on $Ho \times Ho \times 2^{\mathcal{T}_0}$ with the codomain *Boolean*, whereas *Holds* operates on $Ho \times 2^{\mathcal{T}_0}$ with the codomain *Boolean*. More precisely, if `AGREE A,B:` $\overline{\texttt{Var}}$ is a `CAPSL` assertion, then $Agree(A, B, Vars)$ is a predicate in $\mathcal{S}_L$, where the set $Vars \subseteq \mathcal{T}_0$ of atomic terms on $\mathcal{S}$ corresponds pointwise to the list $\overline{\texttt{Var}}$ specified in the `CAPSL` assertion. Similarly, if `HOLDS A :` $\overline{\texttt{Var}}$ is a `CAPSL` assertion, then

$Holds(A, Vars)$ is a predicate in $\mathcal{S}_L$, where the set $Vars \subseteq OwnedAtoms^A$ of atomic terms on $\mathcal{S}$ corresponds point-wisely to the list $\overline{\texttt{Var}}$ in `CAPSL`.

Let $\mathcal{S}$ be the signature for a protocol Pr, $A, B \in Ho$ and $\sigma_1, \sigma_2 \in \Sigma_{Pr}$ such that $\sigma_1(A\text{-}role) \overset{\Upsilon_{IS}}{\mapsto} ag_A^{\sigma_1}$ and $\sigma_2(B\text{-}role) \overset{\Upsilon_{IS}}{\mapsto} ag_B^{\sigma_2}$. Using the $[|\cdot|]$ notation, the *denotational interpretation* $\mathbb{I}$ *under substitutions* σ_1 *and* σ_2 for the predicate symbols introduced is given as follows:

$$\begin{cases} Agree^{\mathbb{I}^{\sigma_1,\sigma_2}}(A, B, Vars) \equiv [|(\forall t \in Vars)(ag_A^{\sigma_1}.view(t) = ag_B^{\sigma_2}.view(t))|] \quad (1) \\ Holds^{\mathbb{I}^{\sigma_1}}(A, Vars) \equiv [|(\forall t \in Vars)(ag_A^{\sigma_1}.view(t) = \sigma_1(t))| \quad (2) \end{cases}$$

The meaning of (1) is that, at some point in the Υ_{IS} computation, the same values should respectively be found in the views of $ag_A^{\sigma_1}$ and $ag_B^{\sigma_2}$ under the entries for atoms $t \in Vars$. If this is so, then $Agree^{\mathbb{I}^{\sigma_1,\sigma_2}}(A, B, Vars)$ evaluates to *true*, otherwise $Agree^{\mathbb{I}^{\sigma_1,\sigma_2}}(A, B, Vars)$ evaluates to $false$. By the Υ_{IS} initialisation phase, for $k \in OwnedAtoms^A \cap Vars$, it is the case that $ag_A^{\sigma_1}.view_0(k) = \sigma_1(k)$ and $ag_B^{\sigma_2}.view_0(k) = \perp$, whereas for $k' \in OwnedAtoms^B \cap Vars$, it is the case that $ag_B^{\sigma_2}.view_0(k') = \sigma_2(k')$ and $ag_A^{\sigma_1}.view_0(k') = \perp$. Then, under a Dolev-Yao environment, it is of interest to evaluate the tests $ag_A.view(k) = ag_B.view(k)$ and $ag_A.view(k') = ag_B.view(k')$ at some non-initial stage in the protocol execution.

The meaning of (2) is that of querying if (indeed) $ag_A^{\sigma_1}$ holds a certain value for her own atoms t. Recall the initialisation phase in Υ_{IS}, i.e., every atom t in $OwnedAtoms^A$ is set to the persistent value of $\sigma(t)$ in the initial view of ag_A^{σ} agent. Then, since $Vars \subseteq OwnedAtoms^A$, $[|Holds^{\mathbb{I}^{\sigma_1}}(A, Vars)|]$ is trivially evaluated *true*. However, the denotation $[|Holds^{\mathbb{I}^{\sigma_1}}(A, Vars)|]$ contributes to less trivial meanings when used inside more involved formulae rendering security requirements.

In relation to `CAPSL` assertions and `CAPSL` goals, we add some helper predicates and notations. Predicate *end* operates over Ag and $end^{\mathbb{I}^{\sigma_1}}(ag_A) = [|ag_A^{\sigma_1}.step = last(Steps^A) + 1|]$; hence, $end^{\mathbb{I}^{\sigma_1}}(ag_A)$ denotes that the *A-role* corresponding to agent $ag_A^{\sigma_1}$ has ended. Similarly, $start(ag_A)$ denotes that some agent *A-role* agent is in its first execution step. Formally, $start^{\mathbb{I}^{\sigma_1}}(ag_A) = [|ag_A^{\sigma_1}.step = first(Steps^A)|]$. The notation $ag_A^{\sigma_1}.id$ denotes the name of the participant spawning $ag_A^{\sigma_1}$, i.e., it is an abbreviation for $ag_A^{\sigma_1}.view(A)$. The notation $ag_A^{\sigma_1}.PartnerB$ denotes the identity of some *B-role* agent acting as a communication partner of agent $ag_A^{\sigma_1}$; the notation $ag_A^{\sigma_1}.PartnerB$

is an abbreviation for $ag^{\sigma_1}_A.view(B)$.

We are now going to show how we express `CAPSL` goals as specifications over Υ_{IS}. Let Pr be a protocol, $\mathcal{G}$ be its set of `CAPSL` atomic and complex goals and $\mathcal{S}$ be the signature for the protocol Pr. Let $A, B \in Ho$ and $\sigma_1, \sigma_2 \in \Sigma_{Pr}$ such that $\sigma_1(A\text{-}role) \overset{\Upsilon_{IS}}{\mapsto} ag_A$ and $\sigma_2(B\text{-}role) \overset{\Upsilon_{IS}}{\mapsto} ag_B$. To express `CAPSL` goals as specifications in Υ_{IS}, we give a relation $\rho_{IS} \subseteq \mathcal{G} \times 2^{Form}$, where $Form$ is the set of well-founded CTLK formulae. The relation ρ_{IS} is called the *expression-relation* in Υ_{IS}.

We introduce a relation of this kind for three reasons: *1)* in order to systematise and generalise the expression of goals, as the rest of the section will show; *2)* in order to construct some of the proofs presented in Chapter 4; *3)* in order to clarify the procedure of automatic generation of such expressions presented in Chapter 5. We will define the relation progressively (i.e., not in one step).

Expression of Atomic `CAPSL` goals in Υ_{IS}. The Υ_{IS} expression of the `CAPSL` atomic goal g=`AGREE A, B :` $\overline{\texttt{VAR}}$ is partially given as follows.

The expression ρ_{IS}(`AGREE A, B :` $\overline{\texttt{VAR}}$) contains

$$\begin{cases} \bigwedge\limits_{i \in \cup ag_A} AG(end(i) \rightarrow \bigvee\limits_{j \in \cup ag_B} (i.PartnerB = j.id)) \;\; (A1) \\ \bigwedge\limits_{i \in \cup ag_A} AG(end(i) \rightarrow \bigvee\limits_{j \in \cup ag_B} (i.PartnerB = j.id \wedge Agree(i, j, Vars))) \;\; (A2) \\ \bigwedge\limits_{i \in \cup ag_A} AG(end(i) \rightarrow \bigvee\limits_{j \in \cup ag_B} (i.PartnerB = j.id \wedge j.PartnerA = i.id \wedge Agree(i, j, Vars))) \;\; (A3) \end{cases}$$

The goal g=`AGREE A, B :` $\overline{\texttt{VAR}}$ is *in relation ρ_{IS} with the set of formulae implied by schemata* $(A1)$*,* $(A2)$ *and* $(A3)$; i.e., in Υ_{IS}, the goal g can be expressed as any of the elements of this set.

In the above, " $\bigwedge\limits_{i \in \cup ag_A}$ " denotes universal quantification over agents instantiating an *A-role*. In that sense, " $\bigwedge\limits_{i \in \cup ag_A}$ " could be dropped and, instead, we could implicitly assume universal quantification over $i \in \bigcup\limits_{\sigma(A-role) \in \Sigma_{Pr}} ag^{\sigma}_A$. Furthermore, instead of using " $\bigvee\limits_{j \in \cup ag_B}$ ", we can make ρ_{IS}(`AGREE A,B :` $\overline{\texttt{VAR}}$) more precise, by arbitrarily fixing $i \in \bigcup\limits_{\sigma(A-role) \in \Sigma_{Pr}} ag^{\sigma}_A$ and making j range over $\bigcup\limits_{\sigma(B-role) \in \Sigma_{Pr}} ag^{\sigma}_B$ as in the following.

For each $j \in \bigcup\limits_{\sigma(B-role) \in \Sigma_{Pr}} ag^{\sigma}_B$, the expression ρ_{IS}(`AGREE A, B :` $\overline{\texttt{VAR}}$) contains

$$\begin{cases} AG(end(i) \rightarrow (i.PartnerB = j.id)) \;\; (A1_j) \\ AG(end(i) \rightarrow (i.PartnerB = j.id \wedge Agree(i, j, Vars))) \;\; (A2_j) \\ AG(end(i) \rightarrow (i.PartnerB = j.id \wedge j.PartnerA = i.id \wedge Agree(i, j, Vars))) \;\; (A3_j). \end{cases}$$

For each agent i, the goal g=`AGREE A, B :` $\overline{\texttt{VAR}}$ is *in relation ρ_{IS} with the set of formulae implied by* $(A1_j)$*,* $(A2_j)$ *and* $(A3_j)$, for j ranging over *B-agents* as above. In order words, the goal g can be expressed in Υ_{IS} as any of the elements of the set implied by schemata $(A1_j)$, $(A2_j)$ and $(A3_j)$, where i, j are as above.

The expression-relation $\rho_{IS}(g)$ for g=`AGREE A, B :` $\overline{\texttt{VAR}}$ is only given by $(A1)$–$(A3)$ and $(A1_j)$–$(A3_j)$, as above.

We now explain the formulae implied above. Schema $(A2)$ says that for every *A-role* agent i there exists a *B-role* agent j such that whenever agent i finishes her role, she “thinks” of agent j as her communication partner and the agents agree on the values of atoms in $Vars$. The formal interpretation in Υ_{IS}^{Pr} of $(A2)$ is that, for every $\sigma_1(A\text{-}role) \in \Sigma_{Pr}, \sigma_1(A\text{-}role) \overset{\Upsilon_{IS}}{\mapsto} i$, it is needed that the model M_{IS} validates the formula

$$AG(end(i) \rightarrow \bigvee_{\sigma_2(B\text{-}role), j := ag_B^{\sigma_2}} (i.PartnerB = j.id \wedge Agree^{\mathbb{I}^{\sigma_1,\sigma_2}}(i, j, Vars))).$$

In turn, the interpretation of $(A2_j)$ is that, for an arbitrarily fixed $\sigma_1(A\text{-}role) \in \Sigma_{Pr}, \sigma_1(A\text{-}role) \overset{\Upsilon_{IS}}{\mapsto} i$, it is needed that the model M_{IS} validates the formula

$$AG(end(i) \rightarrow (i.PartnerB = j.id \wedge Agree^{\mathbb{I}^{\sigma_1,\sigma_2}}(i, j, Vars))),$$

for some $\sigma_2(B\text{-}role) \in \Sigma_{Pr}, \sigma_2(B\text{-}role) \overset{\Upsilon_{IS}}{\mapsto} j$. Hence, there exists $\sigma_2(B\text{-}role) \in \Sigma_{Pr}$ such that $j \in Ag$ is given by $\sigma_2(B\text{-}role) \overset{\Upsilon_{IS}}{\mapsto} j$ and the formula implied by $(A2_j)$ is validated in Υ_{IS}^{Pr}. Similar interpretations and readings are attributed to $(A1)$, $(A1_j)$, $(A3)$ and $(A3_j)$.

In the following we will exemplify more with $(A1)$–$(A3)$ rather than with $(A1_j)$–$(A3_j)$, for some $j \in \bigcup_{\sigma(B-role) \in \Sigma_{Pr}} ag_B^{\sigma}$. Nevertheless, the discussions we make are also applicable to $(A1_j)$–$(A3_j)$.

The expressions $(A1)$ to $(A3)$ are in correspondence with Lowe's hierarchy of agreement, which we presented in Section 2.2.3.1 of Chapter 2. In that sense, $(A1)$ requires that some *B-agent* with the identity $i.PartnerB$ (e.g., *bob*) is present on every execution where the *A-agent* is present. Hence, it requires Lowe's aliveness of participant $i.PartnerB$. In turn, $(A2)$ corresponds to Lowe's non-injective agreement, whereas $(A3)$ relates to Lowe's injective agreement (e.g., the agents quantified are mutual partners in the communication implied).

In the ρ_{IS} expression of the `PRECEDES` goals, we include a stronger temporal aspect in the specification. We express a `PRECEDES` assertion as a reactivity formula[5]. This makes the correspondence to Lowe's agreement even tighter (i.e., whenever an agent of the *A-role* has started a communication, eventually it is the case that an agreement with a *B-role* agent takes place). We give the Υ_{IS} specifications for `PRECEDES` goals.

The expression ρ_{IS}(`PRECEDES A, B :` $\overline{\texttt{VAR}}$) contains

$$\begin{cases} \bigwedge_{i \in \cup ag_A} AG(start(i) \rightarrow AF \bigvee_{j \in \cup ag_B} (i.PartnerB = j.id)) \ \ (P1) \\ \bigwedge_{i \in \cup ag_A} AG(start(i) \rightarrow AF \bigvee_{j \in \cup ag_B} (i.PartnerB = j.id \wedge (end(i) \rightarrow Agree(i,j,Vars)))) \ \ (P2) \\ \bigwedge_{i \in \cup ag_A} AG(start(i) \rightarrow AF \bigvee_{j \in \cup ag_B} (i.PartnerB = j.id \wedge j.PartnerA = i.id \wedge (end(i) \rightarrow Agree(i,j,Vars)))) \ \ (P3) \end{cases}$$

To complete the definition of ρ_{IS}(`PRECEDES A , B :` $\overline{\texttt{VAR}}$), schemata $(P1_j)$–$(P3_j)$ respectively analogous to $(A1_j)$–$(A3_j)$ are also considered.

The correspondence of $(P1)$–$(P3)$ with Lowe's hierarchy for agreement is analogous with the previous case of $(A1)$–$(A3)$.

We now give the Υ_{IS} specifications for `SECRET` goals in `CAPSL`.

ρ_{IS}(`SECRET VAR, HOLDS A: VAR`) $=$

$$\begin{cases} \bigwedge_{i \in \cup ag_A} AG(Environment.POOL.var \neq ag_A.var) \ \ (S1) \\ \bigwedge_{i \in \cup ag_A} AG(\neg K_{Env}(Holds(A, var))) \ \ (S2), \end{cases}$$

where $A \in Ho$, `VAR` is a `CAPSL` variable stipulated in a `HOLDS A:Var` assertion and it corresponds in the signature $\mathcal{S}$ to $var \in OwnedAtoms^A$. An attack on specification $(S1)$ reduces itself to a CTL reachability property: there exists a path, there exists a point on the path where the intruder learns the value $\sigma_1(var)$, where $\sigma_1(A\text{-}role) \in \Sigma_{Pr}$ and $\sigma_1(A\text{-}role) \mapsto ag_A$. In turn, specification $(S2)$ hinges over a much stronger property than $(S1)$: it is never the case that the intruder can definitely link $ag_A^{\sigma_1}$ with the possession of the pair $(var, \sigma_1(var))$.

Expression of Complex `CAPSL` goals in Υ_{IS}. Let $\mathcal{S}$ be the signature and $\mathcal{S}_L$ be the logically extended signature for a protocol Pr. Let $\sigma_1, \sigma_2 \in \Sigma_{Pr}$ such that $\sigma_1(A\text{-}role) \overset{\Upsilon_{IS}}{\mapsto} ag_A^{\sigma_1}$ and $\sigma_2(B\text{-}role) \overset{\Upsilon_{IS}}{\mapsto} ag_B^{\sigma_2}$, where $A, B \in Ho$.

[5]For "reactivity formula", see page 26.

All the formulae in the taxonomies for agreement are of the form $AG(end(i) \rightarrow subformula(i))$, for every $i \in \bigcup_{\sigma(A-role) \in \Sigma_{Pr}} ag_A^{\sigma}$. In other words, in each formula there is a property regarding agent $i \in \bigcup_{\sigma(A-role) \in \Sigma_{Pr}} ag_A^{\sigma}$ to be evaluated at the end of i's instantiated role. We use ρ_{IS}^i to denote this property relative to agent i, $subformula(i)$, within a given specification in the expression ρ_{IS} of a goal. We call it the expression-relation ρ_{IS} *relative to agent* i (i.e., we arbitrarily fix one agent i of role A and refer to the part of the expression ρ_{IS} that varies with i). The same observations can be made about the taxonomy of expressions for the goal `PRECEDES` (i.e., where $end(i)$ is replaced with $start(i)$).

Let g= `KNOWS A :` α be a complex `CAPSL` goal such that α is an atomic `CAPSL` fact. Recall from Section 3.2 that g=`KNOWS A :` α states that the *A-agent* acknowledges that the fact α is the case. Let $\rho_{IS}(\alpha)$ be the expression-relation of its inner atomic `CAPSL` goal α as given before. Then, in the following, we lift ρ_{IS} given for atomic goals to complex goals.

The expression of g as a CTLK formula under Υ_{IS} is:

$$\rho_{IS}(\texttt{KNOWS A : } \alpha) = \bigwedge_{i \in \cup ag_A} AG(end(i) \rightarrow K_i \rho_{IS}^i(\alpha)),$$

where $\rho_{IS}^i(\alpha)$ is one expression of α relative to agent i (i.e., one expression relative to agent i from all those possible for the atomic goal α, chosen at the specifier's preference).

To illustrate $\rho_{IS}^i(\alpha)$ we consider only some branches of ρ for `AGREE` and `PRECEDES` goals, e.g., $(A2)$ and $(P2)$, respectively. For instance, for the above α being `AGREE A, B :` $\overline{\texttt{VAR}}$, assume that from all formulations in ρ_{IS}(`AGREE A, B :`$\overline{\texttt{VAR}}$) we select the expression in $(A2)$. This gives the following expansion of the goal's expression.

Example 3.3.22 (An Expression of α = `AGREE A, B :` $\overline{\texttt{VAR}}$ Relative to Agent i of A-*role*)

$\rho_{IS}^i(\alpha) = \rho_{IS}^i(\textit{AGREE A, B : }\overline{\textit{VAR}}) = \bigvee_{j \in \cup ag_B} (i.PartnerB = j.id \wedge Agree(i, j, Vars))$

Example 3.3.22 triggers the following expression of the complex goal g.

Example 3.3.23 (Expression of `KNOWS A : AGREE A, B :` $\overline{\texttt{VAR}}$ under $(A2)$ for $\rho_{IS}(\alpha)$)

$\rho(\textit{KNOWS A : } \alpha) = \rho(\textit{KNOWS A : AGREE A, B : }\overline{\textit{VAR}}) =$

$\bigwedge_{i \in \cup ag_A} AG(end(i) \rightarrow K_i \rho_{IS}^i(\alpha)) = \bigwedge_{i \in \cup ag_A} AG(end(i) \rightarrow K_i(\bigvee_{j \in \cup ag_B} (i.PartnerB = j.id \wedge Agree(i, j, Vars))))$

We will now exemplify how to lift ρ_{IS} to complex goals using ρ^i_{IS} for the branches of ρ_{IS} chosen above. The method is systematic and it is transferable to other formulations of ρ_{IS} in Υ^{Pr}_{IS} (apart from $(A2)$ and $(P2)$, which we selected for illustration purposes).

We first lift expressions ρ^i_{IS} relative to an agent i to express goals that contain `CAPSL` assertion of `KNOWS` or `BELIEVES`:

$$\rho^i_{IS}(\texttt{KNOWS A:}\alpha) = \bigvee_{j \in \cup ag_A} (i.PartnerA = j.id \wedge K_j \rho^j_{IS}(\alpha)).$$

Thus, $\rho^i_{IS}(\texttt{KNOWS A :}\ \alpha)$ states that there is some A-agent j, representing the partner of the previously quantified agent i and this agent j knows $\rho^j_{IS}(\alpha)$.

We give similar expression-relations ρ_{IS} for doxastic `CAPSL` goals:

$\rho_{IS}(\texttt{BELIEVES A :}\ \alpha) = \bigwedge_{i \in \cup ag_A} AG(end(i) \rightarrow K_i\, \rho^i_{IS}(\alpha))$

$\rho^i_{IS}(\texttt{BELIEVES A :}\ \alpha) = \bigvee_{j \in \cup ag_A} (i.PartnerA = j.id \wedge K_j\, \rho^j_{IS}(\alpha))$

We give some examples of full expression of CTLK formula starting from `CAPSL` goals.

Example 3.3.24 (Atomic goal) *According to the inductive definition and the appointed selection of relative ρ_{IS} above, the NSPK `CAPSL` goal `AGREE A, B : B,Na` translates into:*

$$\bigwedge_{i \in \cup ag_A} AG(end(i) \rightarrow \bigvee_{j \in \cup ag_B} (i.PartnerB = j.id \wedge i.Na = j.Na))$$

stating that whenever an A-agent i has completed the protocol, participant $i.PartnerB$ is represented by some B-agent j who agrees with i on the variable `Na`.

Example 3.3.25 (Complex goal) *Consider the goal `BELIEVES B: AGREE B,A: K`, the goal of KSL protocol. Applying ρ_{IS} on this goal yields:*

$$\bigwedge_{i \in \cup ag_B} AG(end(i) \rightarrow K_i\, \rho^i_{IS}(\texttt{AGREE A,B : K}))$$

In turn, $\rho^i_{IS}(\texttt{AGREE A,B : K})$ is rewritten as:

$$\bigvee_{j \in \cup ag_A} (i.PartnerA = j.id \wedge i.K = j.K)$$

Thus, the goal fully translates to:

$$\bigwedge_{i \in \cup ag_B} AG(end(i) \rightarrow K_i \bigvee_{j \in \cup ag_A} (i.PartnerA = j.id \wedge i.\mathtt{K} = j.\mathtt{K}))$$

stating that whenever a B-agent i, e.g., representing a participant bob, has completed five protocol steps, agent i knows that the participant i.PartnerA, e.g., alice, is represented by some A-agent j which agrees with i on the variable `K`.

Example 3.3.26 (Acknowledged authentication) *The acknowledged authentication goal in Example 3.2.2,* `KNOWS` *A:* `KNOWS` *B:* `AGREE` *B,A — :*`Ma`*, translates into:*

$$\bigwedge_{i \in \cup ag_A} AG(end(i) \rightarrow K_i \, \rho^i_{IS}(\mathtt{KNOWS\ B:\ AGREE\ B,\ A:\ Ma}))$$

where $\rho^i_{IS}(\mathtt{KNOWS\ B:\ HOLDS\ A:\ Ma})$ *in turn expands to:*

$$\bigvee_{j \in \cup ag_B} (i.PartnerB = j.id \wedge K_j \bigvee_{k \in \cup ag_A} (j.PartnerA = k.id \wedge Agree(k, j, \mathtt{Ma})))$$

Thus, the CTLK translation of goal states that whenever an A-agent i terminates its protocol run, she knows that participant i.PartnerB is represented by some B-agent j who knows that participant j.PartnerA is represented by some A-agent k who agrees with agent j on `Ma`.

In the above, aligned with much of the security literature, we focused on the properties of individual agents representing protocol participants (*alice*, *bob*, etc.). This is useful as agents are the actual parties in the protocol runs. However, it is also of interest to attempt to analyse the epistemic properties of the participants themselves. The epistemic setting employed here makes this particularly straightforward. Assume that, as far as protocol information is concerned, participants have at their disposal all and only the information acquired by the agents representing them. Then, the participant's knowledge is the combined information of all agents representing her. The relevant epistemic notion here is the one of *distributed knowledge* of the group of agents representing a certain participant (see Chapter 2).

If we map the principal used in the epistemic concepts in `CAPSL` goals, e.g., A to an actual participant, e.g., *alice* of role A, we obtain an alternative for $\rho_{IS}(\texttt{KNOWS A : }\alpha)$:

$$\begin{aligned}\rho_{IS}(\texttt{KNOWS A : }\alpha) &:= \bigwedge_{i\in\cup ag_A} AG(end(i)\to K_{i.id}\,\rho^i_{IS}(\alpha))\\ \rho^i_{IS}(\texttt{KNOWS A : }\alpha) &:= \bigvee_{j\in\cup ag_A} (i.PartnerA = j.id \wedge K_{j.id}\,\rho^j_{IS}(\alpha))\end{aligned}$$

where $K_{i.id}$ (respectively $K_{j.id}$) is an epistemic modality referring to participant $i.id$ ($j.id$ respectively). Since all information available to the participant $i.id$, e.g., *alice*, is information coming from the various agents representing $i.id$, i.e. information from all sessions that *alice* participates in, we interpret $K_{i.id}\varphi$ as $D_{Gr}\varphi$ where Gr is the set of all agents representing $i.id$, i.e., the set $Gr = \bigcup_{\sigma(A-role)\in\Sigma_{Pr}} \{ag^\sigma_A \,|\, A \in Ho, ag^\sigma_A.view(A) = ag^\sigma_A.id = \sigma(A) \text{ equals } alice\}$ or, even, the set $Gr = \bigcup_{A\in Ho}\bigcup_{\sigma(A-role)\in\Sigma_{Pr}} \{ag^\sigma_A \,|\, ag^\sigma_A.view(A) = ag^\sigma_A.id = \sigma(A)\}$.

Example 3.3.27 *Returning to Example 3.3.25, the alternative mapping ρ_{IS} translates the* `CAPSL` *goal (3.1) into:*

$$\bigwedge_{i\in\cup ag_B} AG(end(i)\to K_{i.id}\bigvee_{j\in\cup ag_A}(i.PartnerA = j.id \wedge i.\mathtt{K} = j.\mathtt{K}))$$

stating that whenever a B-agent i has completed three protocol steps, participant i.id knows that participant i.PartnerA is represented by some A-agent j which agrees with i on the variable `K`.

In this section we have presented a systematic methodology of expressing security goals for a protocol Pr as CTLK formulae in Υ^{Pr}_{IS}. Our expression of the `CAPSL` goals relates to standard interpretations (i.e., reachability specification of secrecy, Lowe's hierarchy of agreement), but it also presents a means to the first fully-automatable way of verifying temporal-epistemic formulations of BAN goals against a sound semantics. Thus, we advance the first systematic approach of mapping authentication and key-establishment goals into the language of CTLK.

Evaluation of the Υ_{IS} Formalism

In the Υ_{IS} formalism, we followed our first motivation, presented in Chapter 1, page 13 and we formulated security requirements in a systematic way that essentially pertains to knowledge of facts, i.e., not to the mere possession of terms.

The introduction of the IS formalism and the advances of model checking interpreted systems, provided us with a well-founded starting point for our Υ_{IS} formalisation. Thus, we followed our third motivation, presented in Chapter 1, page 15 and we put forward a model for security protocol executions that is inspired by the seminal work of the BAN-like logics, whilst supported by a sound semantics. Unlike some of the most advanced authentication logics, e.g., the AT logic [5], our formalisation is not protocol dependent and it is in fact fully automatable. We have presented expressions of security protocol requirements that pursue the VO logic [191] in its expressivity[6] and, additionally, offer the possibility to formulate security goals in ways similar to the well-established Lowe hierarchy [137] for authentication. In doing so, we also enriched the expressivity of VO, where the nesting of beliefs was only implicit. We are able to express acknowledged, mutual or chain authentication in a systematic way, using an explicit nesting of knowledge operators (e.g., Example 3.3.26). Our refined and systematic description of the local states of the agents rules out the ambiguities present in approaches based on authentication logics[7] (e.g., possession of keys, quantification, etc.).

As our fifth motivation states, we follow some of steps of the LDYIS model [125] in building a protocol verification framework based on temporal-epistemic logics. However, in Υ_{IS} we relax the LDYIS conditions upon receiving and sending messages and we adopt a more standard matching-receive [175] semantics. This allows us to go beyond the facilities provided by LDYIS and encode general untrusted channels and therefore capture more attacks (i.e., not exclude impersonation and binding attacks[8]). At a higher level, Υ_{IS} generalises the LDYIS model in that it advances a full denotational semantics for the behaviour of agents (i.e., Section 3.3.1). Theoretically, this allows for the verification of any authentication and key-establishment protocol in a uniform way and for a full automation of the methodology. Chapter 5 will detail this precisely.

With Υ_{IS} and the M_{IS} model, we introduce a novel, systematic methodology of formalising the Dolev-Yao execution of `CAPSL`-described protocol in a MAS setting. In doing so, we follow our second motivation, presented in Chapter 1, page 14; we attributed a more expressive semantics to

[6]See Chapter 2, page 51.

[7]See Chapter 2, page 40.

[8]See Chapter 2, page 35 and/or [145,192].

`CAPSL`-described protocols, which are usually denoted under a trace-based formalisation. This implies that whilst we are able to verify reachability properties corresponding to certain security goals (e.g., secrecy), we are also able to go beyond and verify inherent epistemic properties corresponding to those and to other security goals. Section 3.3 of this chapter showed this precisely.

Nevertheless, the fact that we start from a `CAPSL` description and we attribute it an IS semantics requires some proofs of preservation of the properties enjoyed by the traditional, trace-based `CAPSL` semantics. These will be given in Chapter 4.

The main limitation of the Υ_{IS} model is that it deals only with receiver-transparent protocols. Hence, the state of agents contain only values for atomic terms. This is advantageous for automatic verification in that it requires simple indistinguishability relations and it is a state-encoding which should not exacerbate the state-explosion problem. The later chapters show that this intuition is indeed correct.

We recall that the class of RT protocols does not equal the class of protocols with one level of encryption in the messages. The former is a super-class of the latter. As aforementioned, numerous authentication and key-establishment protocols fall into the class of RT protocols. Thus, the Υ_{IS} methodology is well suited for the formalisation of authentication and key-establishment protocols.

Nevertheless, we will also investigate other classes of protocols modelled as MAS. In that sense, Chapter 7 (and Appendix C) will present systematisations and automations in generating and verifying MAS models for receiver-opaque protocols.

In this chapter we have given the first systematic multiagent system model for multi-session executions of `CAPSL`-described receiver-transparent protocols. The core of the semantics is given in terms of interpreted systems. We have also systematically mapped security `CAPSL` goals into a taxonomy of CTLK formulae to be interpreted on the model. In the next chapters we will show: *1)* the correctness of this methodology in relation to standard semantics of security protocols; *2)* an optimised implementation of this methodology, leading to the first automatic verification of security protocols against temporal-epistemic specifications of requirements; *3)* extensions to this approach.

Chapter 4

On the Correctness of the Υ_{IS} Formalism for Receiver-Transparent Protocols

Motto: "All perception of truth is the detection of an analogy."
(Henry Thoreau)

In this chapter we present the algorithm tr that, given a standard model for a receiver-transparent protocol Pr, outputs the Υ_{IS}^{Pr} formalisation presented in Chapter 3. The expressions of the security goals in the two models are also correlated. We further prove the following: if a standard model for a receiver-transparent protocol Pr validates/refutes the specification of an atomic goal, then the unwound model of Υ_{IS}^{Pr} also validates/refutes corresponding specifications of that atomic goal.

Section 4.1 starts by outlining the most sensitive distinctions between Υ_{IS} and the standard semantics for `CAPSL` used in [188]. Driven by these distinctions, we introduce several notions that help relate the two models. Section 4.2 uses these notions as the building blocks for an algorithm, called tr, that takes a standard model for a `CAPSL`-described protocol Pr and outputs the Υ_{IS}^{Pr} formalisation of Pr. Section 4.2 also correlates the respective expressions of security goals in the two formalisms. Recall from Chapter 3 that M_{IS}^{Pr} is used to denote the $S5_n$-based model unwound by Υ_{IS}^{Pr}. Section 4.3 defines several relations between the states in the M_{CAPSL} model and states in the M_{IS} model, unwound by the output of algorithm tr. In Section 4.4 we use these relations to show that there is a homomorphism[1] between the traditional model for CAPSL-described protocols and M_{IS}. Then, we use this to prove the preservation of satisfaction/refutation of formulae respectively corresponding to certain goals in the two models. Figure 4.1 illustrates the structure of this chapter.

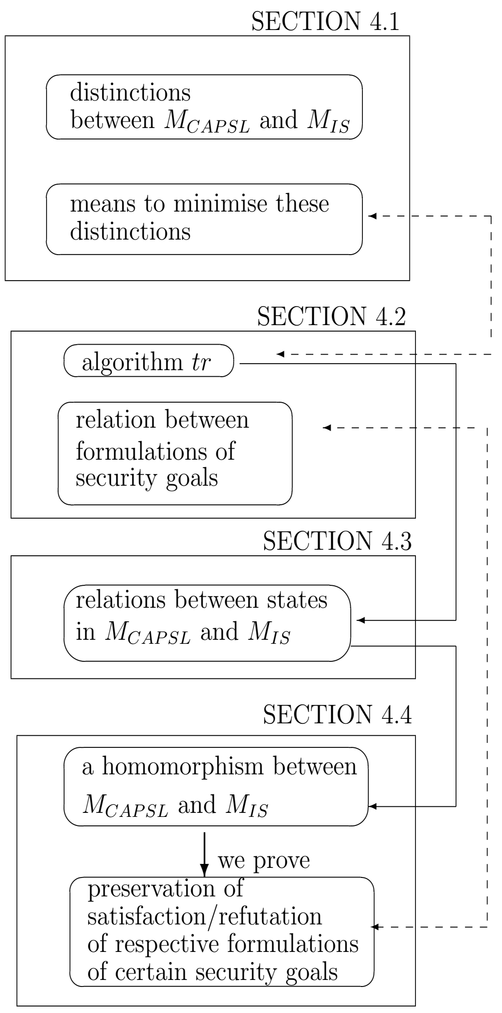

SECTION 4.1
distinctions between M_{CAPSL} and M_{IS}
means to minimise these distinctions
SECTION 4.2
algorithm tr
relation between formulations of security goals
SECTION 4.3
relations between states in M_{CAPSL} and M_{IS}
SECTION 4.4
a homomorphism between M_{CAPSL} and M_{IS}
we prove
preservation of satisfaction/refutation of respective formulations of certain security goals

Figure 4.1 The Flow of Chapter 4

[1]For the definition of a homomorphism, please refer to Appendix B, Definition B.1.1.

4.1 From M_{CAPSL} Standard Semantics for RTP to Υ_{IS} Interpreted Systems

In Section 2.2.2.3 we summarised the bounded protocol model [188] and recalled the full correspondence [188] between this model and the implicit, multiset rewriting semantics [73] for `CAPSL`-described protocols. Therefore, the bounded protocol model is an inherent semantics for `CAPSL` and we further refer to it as $M^{\mathcal{P}}_{CAPSL}$, where $\mathcal{P}$ is the class of receiver-transparent (authentication and key-establishment) protocols.

We will give an algorithm for obtaining the Υ^{Pr}_{IS} formalisation from an M^{Pr}_{CAPSL} model, where Pr is an RT protocol. For that, we first outline the most sensitive points in the differences between the two formalisms. In drawing a relationship between Υ_{IS} and M_{CAPSL}, the main sensitive issues are:

1. multi-session protocol executions are rendered in M_{CAPSL} by a set of protocol instantiations (i.e., substitutions over the entire $\mathcal{T}_0$), whilst multi-session protocol executions are rendered in Υ_{IS} by a set of role-substitutions (i.e., substitutions over subsets $\mathcal{T}_0$ referring to particular roles);

2. the aspect of freshness (check) as in M_{CAPSL} does not appear explicitly in Υ_{IS} (i.e., Υ_{IS} is a fully ground (i.e., not symbolic, or without uninstantiated variables) model. This is because in Υ_{IS}, there is no explicit on-the-fly generation;

3. suitable substitutions in M_{CAPSL} have no explicit correspondent in Υ_{IS}.

In the following, we will show how to minimise them systematically in order to bring the models closer. At a high level, to lessen these differences we need to do the following:
a) transform instantiations of terms in M_{CAPSL} in "appropriate" role-substitutions in Υ_{IS};
b) for these newly constructed role-substitutions to operate in a way which is in keeping with the M_{CAPSL} semantics, build bespoke ranges for terms in the fully ground Υ_{IS} model.
The rest of this section formalises the points *a)* and *b)* above.

Definition 4.1.1 (Generated Atoms in M_{CAPSL}) *Let Pr be a receiver-transparent protocol and* *let $A \in Ho$. The set $Generated^A = \bigcup_{a=(A!B:(M)t)\in\omega|_A} M(a)$ denotes the set of all atoms generated in M^{Pr}_{CAPSL} by the A-role.*

Remark 4.1.2 *The set $OwnedAtoms^A$ in Υ_{IS} relates to atoms in M_{CAPSL} as follows:*
$OwnedAtoms^A = Secret_A \cup Generated^A$.

Issue 2 and issue 3 signalled above arise from the following facts: *a)* in Υ_{IS}, $Generated^A$ is embedded in $OwnedAtoms^A$, as Remark 4.1.2 underlines; *b)* the atoms in $Generated^A$ are assigned to concrete values in the initial views/initial states of Υ_{IS}. To mitigate issue 2 and issue 3, we need to constrain the space of possible initial views in Υ_{IS} in such a way that honest fresh generation and suitable substitutions from M_{CAPSL} are enforced onto Υ_{IS}. To achieve this, we need to constrain the way role-substitutions in Υ_{IS} map terms over ranges. In Definitions 4.1.3–4.1.5 we formalise this, step by step. Definition 4.1.6 eventually introduces the concept of *initial Ho^+-freshness enforcing substitutions*. These describe the type of role-substitutions in Υ_{IS} suitably constrained such that the differences between Υ_{IS} and M_{CAPSL}, as signalled by issues 1–3, are diminished.

Let $A \in Ho$ be an arbitrary principal and $X \in \{\mathcal{N}, \mathcal{K}_0\}$ be a sort. Then, let $Atoms^A|_X$ denote those atoms of the *A-role* which are of sort X. This extends naturally to subsets of $Atoms^A$, e.g., $OwnedAtoms^A$, $LearnedAtoms^A$, etc.

Definition 4.1.3 (Initial Inter-Role Ho-Freshness Enforcing Substitutions) *Let Pr be an RT protocol and Σ_{Pr} be a set of role-substitutions in Υ^{Pr}_{IS}. The set Σ_{Pr} is called* initial inter-role Ho-freshness enforcing with respect to some sort X *if for all $A, B \in Ho$, $A \neq B$, for all $\sigma(A\text{-}role), \sigma'(B\text{-}role) \in \Sigma_{Pr}$, for all $t \in OwnedAtoms^A|_X$, $t' \in OwnedAtoms^B|_X$, it is implied that $\sigma(t) \neq \sigma'(t')$.*

Definition 4.1.3 expresses that a set Σ_{Pr} of role-substitutions is initial inter-role Ho-freshness enforcing with respect to some sort X if every two atoms of sort X, each within a distinct role, are respectively mapped into two different values under Σ_{Pr}.

Recall that in Υ_{IS} the intruder can be an insider of any role, i.e., $OwnedAtoms^{\mathbf{I}} = \bigcup_{A\in Ho} OwnedAtoms^A$ (see Section 3.3.2.2, page 91). Therefore, Definition 4.1.4 will extend the previously introduced concept of initial inter-role Ho-freshness enforcing with respect to some sort X to refer to the insider also.

Definition 4.1.4 (Initial Inter-Role Ho$^+$-Freshness Enforcing Substitutions) *Let Pr be an RT protocol and Σ_{Pr} be a set of role-substitutions in Υ_{IS}^{Pr} (Σ_{Pr} containing a substitution for the intruder agent* **I***). The set Σ_{Pr} is called* initial inter-role Ho^+-freshness enforcing with respect to some sort X *if for all $A \in Ho$, for all $B \in Ho \cup \mathbf{I}$, $A \neq B$, for all $\sigma(A\text{-}role), \sigma^{'}(B\text{-}role) \in \Sigma_{Pr}$, for all $t \in OwnedAtoms^A|_X, t' \in OwnedAtoms^B|_X$, it is implied that $\sigma(t) \neq \sigma^{'}(t')$.*

Definition 4.1.4 enforces the atoms of the insider are mapped distinctively from the atoms of other roles.

Definitions 4.1.3 and 4.1.4 referred themselves to constraints in instantiating atoms belonging to different roles. The next definition is concerned with restraining the ways of instantiating atoms belonging to the same role.

Definition 4.1.5 (Initial Intra-Role Ho-Freshness Enforcing Substitutions) *Let Pr be an RT protocol and Σ_{Pr} be a set of role-substitutions in Υ_{IS}^{Pr} (Σ_{Pr} containing a substitution for the intruder agent* **I***). The set Σ_{Pr} is called* initial intra-role Ho-freshness enforcing with respect to some sort X *if for all $A \in Ho$, for all $\sigma_1(A\text{-}role), \sigma_2(A\text{-}role) \in \Sigma_{Pr}$, $\sigma_1 \neq \sigma_2$, for all $t \in OwnedAtoms^A|_X$, it is implied that $\sigma_1(t) \neq \sigma_2(t)$.*

Definition 4.1.5 expresses that a set Σ_{Pr} of role-substitutions is initial intra-role Ho-freshness enforcing if an atom of some sort X belonging to the same *A-role* is mapped differently under different *A-role* substitutions in Σ_{Pr}, for any $A \in Ho$.

The following concept of *initial Ho^+-freshness enforcing substitutions* subsumes all the constraints underlined by Definitions 4.1.3–4.1.5.

Definition 4.1.6 (Initial Ho$^+$-Freshness Enforcing Substitutions) *Let Pr be an RT protocol and Σ_{Pr} be a set of role-substitutions in Υ_{IS}^{Pr} (Σ_{Pr} containing a substitution for the intruder agent* **I***). The set Σ_{Pr} is called* initial Ho^+-freshness enforcing with respect to some sort X *if Σ_{Pr} is initial intra-role Ho-freshness enforcing with respect to the sort X and initial inter-role Ho^+-freshness enforcing with respect to the sort X.*

To sum up, through Definitions 4.1.3–4.1.6, we introduced step by step the concept of initial Ho^+-freshness enforcing substitutions. If we use them to create the fully ground model Υ_{IS}, then we diminish the differences concerning freshness between Υ_{IS} and M_{CAPSL}. However, working with initial Ho^+-freshness enforcing substitutions in Υ_{IS} will have an impact on the ranges for terms in Υ_{IS}. In a nutshell, the ranges for terms need to be large enough to allow the Υ_{IS} role-substitutions to operate as initial Ho^+-freshness enforcing. We will now formalise which ranges are to be used in Υ_{IS} if we aim to consider initial Ho^+-freshness enforcing role-substitutions in Υ_{IS}, i.e., if we are to reduce the differences concerning freshness between Υ_{IS} and M_{CAPSL}.

Minimal Ranges under Initial Ho$^+$-Freshness Enforcing Role-Substitutions in Υ_{IS}. Let $min\_\mathcal{R}_X$ denote a range for the sort X which is large enough for initial Ho^+-freshness enforcing substitutions with respect to some sort X to operate on. We give a (unoptimised) procedure to determine such a range $min\_\mathcal{R}_X$ for Υ_{IS}.

Procedure 4.1.3 – **Build** $min\_\mathcal{R}_X$, X **sort**

Input : — a set Σ_{Pr} of role-substitutions

Output: — the range $min\_\mathcal{R}_X$ for sort X which is large enough for Σ_{Pr} to be initial Ho^+-freshness enforcing

```
0. $S_1 := 0, S_2 := 0; S_3 := 0;$
1. for each $A \in Ho$
2.     for each $\sigma(A\text{-}role) \in \Sigma_{Pr}$
3.         for each $t \in OwnedAtoms^A|_X$
4.             $S2 := 0$
5.             for each $B \in Ho, B \neq A$
6.                 for each $\sigma'(B\text{-}role) \in \Sigma_{Pr}$
7.                     for each $t' \in OwnedAtoms^B|_X$
8.                         $S3 := 0$
9.                         for each $t'' \in OwnedAtoms^{\mathbf{Insider}}|_X$
10.                          $S_3.add\_new(1);$
11.                          endfor;
12.                          $S_3.add\_new(1);$
13.                          $S_2.add(|S_3|);$
14.                    endfor
15.                    $S_2.add\_new(1);$
16.                endfor
17.            endfor
18.            $S_1.add\_new(|S_2|);$
19.            $S_1.add\_new(1);$
20.        endfor
21.    endfor
22. endfor
23. $min\_\mathcal{R}_X := S_1$
24. return $min\_\mathcal{R}_X$
```

In the for loop at lines 9–11, for each of the intruder's atoms of sort X we add one new element to a set S_3 of values (i.e., operation $S_3.add\_new(1)$). At line 12, we add another value to the resulting set S_3. This is done in order to increase the number of possibilities in which a substitution could map atoms over the elements of S_3, i.e., in order to obfuscate if a value for an atom is indeed drawn from S_3. Such an obfuscation is enforced also when constructing the value-sets S_2 and S_1 (i.e., line 15 and line 19). For each *B-role* substitution σ', for each atom of sort X belonging to an $ag_B^{\sigma'}$ agent, $|S_3|$ new elements are added to the set S_2 (i.e., in the for loop starting at line 7, inner operation $S_2.add\_new(|S_3|)$ at line 13). For each *A-role* substitution σ, for each atom of sort X of an ag_A^{σ} agent, $|S_2|$ new elements are added to S_1 (i.e., in the for loop starting at line 3, inner operation $S_1.add\_new(|S_2|)$ at line 18). Since we count one element for each atom in the inner for loop (i.e., line 10), Σ_{Pr} can act as intra-role freshness enforcing with respect to the sort X. Furthermore, by the result of the outer for loop, atoms of sort X belonging to different roles and to the intruder can be mapped under Σ_{Pr} to distinct values over the set S_1. This means that Σ_{Pr} can act as an inter-role Ho^+-freshness enforcing with respect to the sort X over S_1. So, by lines 23 and 24, the range $min\_\mathcal{R}_X$ is large enough for Σ_{Pr} to operate as initial Ho^+-freshness enforcing with respect to the sort X. Smaller $min\_\mathcal{R}_X$ can be obtained, e.g., by dismissing the operations $add\_new(1)$ when building the sets S_2 and S_1. This can be done without losing the obfuscation explained, but we maintained the operations for clarity.

We call $min\_\mathcal{R}_X$ obtained as above the *minimum range for the sort* X in Υ_{IS}^{Pr} such that initial Ho^+-freshness enforcing substitutions with respect to X can operate.

Recall that Remark 3.1.9 expressed informally the constraints applied to the relation between the set Σ_{Pr} of substitutions and the range $\mathcal{R}_X$ for a sort X in the Υ_{IS} model. We note here that the set Σ_{Pr} of role-substitutions being initial Ho^+-freshness enforcing with respect to some sort X and Σ_{Pr} operating over $min\_\mathcal{R}_X$ formalise the original desiderata, expressed in Remark 3.1.9.

From Procedure 4.1.3, it follows that the size of the range $min\_\mathcal{R}_X$ is:

$$card(\Sigma_{Pr}(A\text{-}role)) \times \left\{ card(OwnedAtoms^A|_X) \times \prod_{B \neq A} card(\Sigma_{Pr}(B\text{-}role)) \times \left[card(OwnedAtoms^B|_X) \times \left(card(OwnedAtoms^{\mathbf{I}}|_X) + 1 \right) + 1 \right] + 1 \right\} \quad (1),$$

where $A \in Ho, B \in Ho$, *card* is an abbreviation for cardinality and $\Sigma_{Pr}(A\text{-}role)$, $\Sigma_{Pr}(B\text{-}role)$ respectively denote the set of *A-role* substitutions and *B-role* substitutions within Σ_{Pr}. We use $min\_size(\mathcal{R}_X)$ to denote the size of the range $min\_\mathcal{R}_X$ as above.

Summing up, we can simulate fresh generation of terms in Υ_{IS} and thus bring our MAS model closer to the traditional trace-based semantics for protocol executions. In order to achieve that, in Υ_{IS} we will consider initial Ho^+-freshness enforcing substitutions with respect to a sort X. Consequently, $min\_\mathcal{R}_X$ will be the range for a sort X used in Υ_{IS} under these circumstances.

From Protocol Instantiations in M_{CAPSL} to Initial Ho^+-Freshness Enforcing Substitutions in Υ_{IS}.

In this subsection we will see an actual method of obtaining the initial Ho^+-freshness enforcing substitutions in the Υ_{IS} model, by starting from the M_{CAPSL} model. The procedures used to achieve this will later be employed in an algorithm that takes the M_{CAPSL} and produces the Υ_{IS} formalism corresponding to it.

Let $Pr \in \mathcal{P}$ be an RT protocol and Ev^{Pr} be the set of all (T,k)-*events* possible in M^{Pr}_{CAPSL}. Recall that, in the M^{Pr}_{CAPSL} model, the cardinality of Ev^{Pr} is bounded by an exponential factor in the size of the protocol Pr (for details, refer to [188], page 694 or see Chapter 2, page 42). We give a systematic method which, given Ev^{Pr}, produces the corresponding set of initial Ho^+-freshness enforcing substitutions of roles for Υ^{Pr}_{IS}. The sub-routines of this method are as follows.

```
(1) /**@returns $Ev^{Pr}|_{\omega|_A}$*/
  –for any $A \in Ho$
      –select all events $(\omega|_A, \sigma, j)$ from $Ev^{Pr}$
```

The output of procedure (1) is the set of all the events pertaining to the *A-role*, for each $A \in Ho$. Procedure (1) is possible simply by confronting Ev^{Pr} with $Rules^A$. Whilst unordered, the set returned by (1) is equal to:

$$\begin{cases} (\omega|_A, \sigma_1, 1),\ \ldots,\ (\omega|_A, \sigma_1, n), & \text{where } \omega|_A{=}a_1 \ldots a_n \text{ and } a_1, \ldots, a_n \text{ are actions} \\ \vdots & \\ (\omega|_A, \sigma_p, 1),\ \ldots,\ (\omega|_A, \sigma_p, n), & \text{where } \omega|_A{=}a_1 \ldots a_n \text{ and } a_1, \ldots, a_n \text{ are actions} \end{cases}$$

Let $\Sigma_{Ev^{Pr}}|_{\omega|_A}$ be the set $\{\sigma_1, \ldots \sigma_p\}$ of *A-role* substitutions occurring in the events Ev^{Pr} of M_{CAPSL}. From the output of (1), by inspecting it in relation with $Steps^A$ and considering the precedence of events in M_{CAPSL}, we can reconstruct the set $\Sigma_{Ev^{Pr}}|_{\omega|_A}$ of substitutions applied in $Ev^{Pr}|_{\omega|_A}$:

```
(1') /**@returns $\Sigma_{Ev^{Pr}}|_{\omega|_A}$*/
  –from $Ev^{Pr}|_{\omega|_A}$ select $\{\sigma_1, \ldots \sigma_p\}$.
```

Among the substitutions in $\Sigma_{Ev^{Pr}}|_{\omega|_A}$ some (might) map A to the insider. In M_{CAPSL} this is stipulated simply by $\sigma_j(A) = \mathbf{I}$ in some action *act*($(\omega|A, \sigma_j, l)$), for $1 \leq l \leq p$. Therefore, by iterating again through $Ev^{Pr}|_{\omega|_A}$, we can construct the set $\Sigma^{Leg}_{Ev^{Pr}}|_{\omega|_A}$ of substitutions that map the principal A to a honest participant and the set $\Sigma^{ILeg}_{Ev^{Pr}}|_{\omega|_A}$ of substitutions that map A to the intruder/insider.

```
(1'') /**@returns $\Sigma^{Leg}_{Ev^{Pr}}|_{\omega|_A}$ and $\Sigma^{ILeg}_{Ev^{Pr}}|_{\omega|_A}$*/
    /* partitions and re-indexes $\Sigma_{Ev^{Pr}}|_{\omega|_A}$ in $\Sigma^{Leg}_{Ev^{Pr}}|_{\omega|_A} = \{\sigma_1, \dots \sigma_k\}$ -substitutions that map $A$ into legal names
    and $\Sigma^{ILeg}_{Ev^{Pr}}|_{\omega|_A} = \{\sigma_{k+1}, \dots \sigma_p\}$ –substitutions that map $A$ into "illegal" names/the intruder */

    –for each $e = (\omega|_A, \sigma, j) \in Ev^{Pr}|_{\omega|_A}$
        –if $\sigma(A) = \mathbf{I}$ then $\Sigma^{ILeg}_{Ev^{Pr}}|_{\omega|_A} := \Sigma^{ILeg}_{Ev^{Pr}}|_{\omega|_A} \cup \sigma$;
        –else $\Sigma^{Leg}_{Ev^{Pr}}|_{\omega|_A} := \Sigma^{Leg}_{Ev^{Pr}}|_{\omega|_A} \cup \sigma$;
    –endfor
```

Remark 4.1.7 *We can simply assume that the sets $\Sigma^{Leg}_{Ev^{Pr}}|_{\omega|_A}$ and $\Sigma^{ILeg}_{Ev^{Pr}}|_{\omega|_A}$ of substitutions of the A-role are given (i.e., algorithm A1 in [188], page 695, outputs a maximal leaky/non-leaky run of M_{CAPSL}; it can also give the set of A-role substitutions used in constructing this run). For clarity of intuition, we opted for using the procedures above to sketch the extraction of the set of substitutions of the A-role from a set Ev^{Pr} of events in M^{Pr}_{CAPSL}.*

Because we extracted the set of substitutions for the *A-role* in M^{Pr}_{CAPSL} from the entire event-space in M^{Pr}_{CAPSL}, some of these substitutions might be applied only to a proper subset of the atoms in the role of A (i.e., an extracted substitution could potentially encode an incomplete session executed by the *A-role*). In order to initialise a MAS model corresponding to Υ_{IS}, we need to consider the application of role-substitutions over the entire *OwnedAtoms*^{A}. Also, the substitutions extracted could map some of atoms in *LearnedAtoms*^{A} into values. In order to initialise a MAS model corresponding to Υ_{IS}, we need to consider the application of role-substitutions that do not map atoms in *LearnedAtoms*^{A} to concrete values (but, instead, they map atoms in *LearnedAtoms*^{A} to $\perp$). The procedures below restrict each substitution in $\sigma \in \Sigma_{Ev^{Pr}}|_{\omega|_A}$ to not operate on *LearnedAtoms*^{A}. Also, when a substitution σ did not map $t \in$ *OwnedAtoms*^{A} into a value throughout the events Ev^{Pr} of M_{CAPSL}, the following procedures extend σ to map atom $t \in$ *OwnedAtoms*^{A} into a random value over the correct range.

```
(2) /**@restrict $\Sigma^{Leg}_{Ev^P}|_{\omega|_A}$ to OwnedAtoms$^A$ */
–for each $\sigma_j \in \Sigma^{Leg}_{Ev^P}|_{\omega|_A} = \{\sigma_1, \dots, \sigma_k\}$
```

```
    for $t \in OwnedAtoms^A$
        if $[v/t] \in \sigma$
            $\sigma_j^+[v/t]$;
            $register_A(v, sort(t))$;
        $\sigma_j^+[\perp/t']$, for $t' \in LearnedAtoms^A$
    –endfor
–endfor

(2′) /**@restrict $\Sigma^{ILeg}_{Ev^P}|_{\omega|_A}$ to $OwnedAtoms^A$ */
–for each $\sigma_j \in \Sigma^{ILeg}_{Ev^P}|_{\omega|_A} = \{\sigma_{k+1}, \ldots, \sigma_p\}$
    $\sigma_j^+[\sigma_j(t)/t, \perp/t']$, for $t \in OwnedAtoms^A$, $t' \in LearnedAtoms^A$
    $v' := new(unregistered_{\cup_{A \in Ho}, sort(t)})()$;
    $register_A(v', sort(t))$;
–endfor

(2″) /**@extend $\Sigma^{Leg}_{Ev^P}|_{\omega|_A}$ to all of $OwnedAtoms^A$ */
–for each $\sigma_j \in \Sigma^{Leg}_{Ev^P}|_{\omega|_A} = \{\sigma_1, \ldots, \sigma_k\}$
    –for $t \in OwnedAtoms^A$
        if $[v/t] \notin \sigma$
        $v' := new(unregistered_{\cup_{A \in Ho}, sort(t)})()$;
        $register_A(v', sort(t))$;
        $\sigma_j^+[v'/t]$
    –endfor

–endfor
```

If we compute the union of $\Sigma^{ILeg}_{Ev^P}|_{\omega|_A}$ for all $A \in Ho$, we will obtain the substitution to be applied in $\varUpsilon_{IS}$ to the intruder as an insider.

Let σ be a role-substitution applied uniformly to an *A-role* in an arbitrary run of M_{CAPSL}, extracted from Ev^{Pr} as above. Let σ^+ be obtained through procedures (2), (2′) and (2″). By "registering" v as a possible value for the term t of sort X (i.e., $register_A(v, sort(t))$), the construction of σ^+ also contributes to creating $min\_\mathcal{R}_X$ (as per procedure 4.1). The call to $v' := new(unregistered_{\cup_{A \in Ho}, sort(t)})$ in procedure (2″) denotes that if no value was defined for t through σ (i.e., σ was a partial role-substitution), a yet unused value v' is added to $min\_\mathcal{R}_X$ and $[v'/t]$ is added to σ^+. In (2′) a new value is added to $min\_\mathcal{R}_X$ for each value that the intruder has used in the events Ev^{Pr}.

Then, through procedures (1), (1′), (1″), (2), (2′), (2″) we create initial Ho^+-freshness enforcing substitutions in $\varUpsilon_{IS}$ starting from the set of events/substitutions in M_{CAPSL}. Through procedures (2), (2′), (2″) emulating procedure 4.1, these initial Ho^+-freshness enforcing substitutions also give $min\_\mathcal{R}_X$ for sort X in $\mathcal{S}$. For each $A \in Ho$, these initial Ho^+-freshness enforcing substitutions leave atoms t in $LearnedAtoms^A$ unmapped (i.e., they iterate freely over $\mathcal{R}_t$). This implies that any results of compatibility over substitutions, states and events/actions between M_{CAPSL} and Υ_{IS} will be true up to renaming of values for atoms in $LearnedAtoms^A$ over their respective ranges, for each $A \in Ho$. Given the construction of σ^+ out of σ, the initial Ho^+-freshness enforcing substitutions obtained for Υ_{IS} preserve the inherent properties of suitable substitutions in M_{CAPSL}.

By introducing the notion of *extending substitutions*, Definition 4.1.8 will now summarise the entire concept of obtaining such "appropriate" role-substitutions in Υ_{IS} from the instantiations used in M_{CAPSL}.

Definition 4.1.8 (Extended Substitutions in Υ_{IS}) *For $A \in Ho$, let σ be a role-substitution applied uniformly to an A-role in an arbitrary run of M_{CAPSL}. The substitution σ^+ obtained from σ via the procedures* (1), (1′), (1″) (2), (2′), (2″) *is the* substitution extending σ *in Υ_{IS}. We write $\sigma^+ \cong \sigma$.*

To conclude, in Definitions 4.1.3–4.1.6 we have expressed how role-substitutions in Υ_{IS} should be constrained such that they are closer to the instantiations used in M_{CAPSL}. In Procedure 4.1 we showed which ranges for terms are to be considered in Υ_{IS} under these circumstances. Through procedures (1), (1′), (1″) (2), (2′), (2″) we gave an algorithmic method to obtain these constrained Υ_{IS} role-substitutions starting from the M_{CAPSL} model. Thus, it is possible to simulate honest fresh generation of terms and preserve properties of suitable substitutions as per M_{CAPSL} into Υ_{IS}, i.e., to bring our Υ_{IS} formalisation closer to the M_{CAPSL} model.

4.2 M_{CAPSL} to Υ_{IS}: A Translation Schema

In this section we will formally correlate M_{CAPSL}, i.e., the standard trace-based model for authentication protocols, with Υ_{IS}, our MAS-based formalism for RT authentication protocols.

4.2.1 M_{CAPSL} to Υ_{IS}: A Model-Translation Algorithm

In this subsection we give a translation algorithm tr that takes a trace-based model M^{Pr}_{CAPSL} and produces Υ^{Pr}_{IS}, our IS-based formalism for protocol executions, where Pr is an RT protocol. In doing so, we will use the procedures previously given in Section 4.1. The output of the algorithm is given in an ISPL[2]-like syntax.

We begin by briefly explaining the ideas behind the algorithm tr. Given the set Ev^{Pr} of events in M^{Pr}_{CAPSL}, the algorithm tr starts by creating all the initial Ho^{+}-freshness enforcing substitutions and the corresponding ranges for terms in Υ^{Pr}_{IS}. Then, it lays the foundations of Υ^{Pr}_{IS} by drawing all the fine distinctions between terms as per Chapter 3 (e.g., $OwnedAtoms^{A}$, $LearnedAtoms^{A}$, for an A-*role*). It uses these to construct the local states and initial local states of each agent $ag^{\sigma^+}_{A}$, where σ is a substitution in Ev^{Pr} and $\sigma \cong \sigma^{+}$. Then, for each event in Ev^{Pr}, it creates the corresponding ground actions in Υ_{IS} (i.e., actions were each parameter/variable is instantiated to a concrete value). Local protocol functions and local evolution functions, as well as the Environment agent are added accordingly and following their original description in Chapter 3. Full explanations of the algorithm tr are given immediately after the presentation of its pseudocode.

`Algorithm` tr

–let $\mathcal{S}$ be a sorted signature, X a sort

–let Pr be an RTP, formalised as $Pr = (\mathcal{S}, C, \omega)$ and let M^{Pr}_{CAPSL} be its CAPSL-model

Input: –Ev^{Pr} the set of all (T,k)-*events* possible in M^{Pr}_{CAPSL}

Output: – the IS-based formalism Υ^{Pr}_{IS} (under role-substitutions extending those in M^{Pr}_{CAPSL})

Body:

```
1). /** @returns:
the due data–structures, e.g., for each $A \in Ho$, $Atoms^{A}$, $Generated^{A}$, $OwnedAtoms^{A}$, $LearnedAtoms^{A}$,
$PublicAtoms$, $A\text{-}store$, etc. */

extendSignatureTo$\Upsilon_{IS}(Pr)$;
2). /** @returns:
the initial-$Ho^{+}$ freshness enforcing substitutions, freshness-permissive ranges for each atomic term $t \in T_0$ (e.g.,
for each $X$ sort, returns $\mathcal{R}_X$, for each $t \in Atoms^{A}$, for each $\sigma \in \Sigma^{Leg}_{Pr}|_{\omega|_A}$, returns $\mathcal{R}^{\sigma^+}_{t}$), ordered n-tuples for
send actions (e.g., for each $m \in SentMsg^{A}$, returns generalised Cartesian product $R^{\sigma^+}_{m}$) */

$\Sigma^{+}_{Pr}$ :=createExtendedSubstitutions$(Pr, Ev^{Pr})$; //proc. (1)–(2'')
$(\sigma, \sigma^{+}) \in SubstMap \subseteq (\Sigma_{Pr}, \Sigma^{+}_{Pr})$
createRanges$(Pr, \Sigma^{+}_{Pr})$; //procedure for building $min\_\mathcal{R}_X$
3).
for each $A \in Ho$
    for each $\sigma \in \Sigma^{Leg}_{Pr}|_{\omega|_A}$
        /** @returns:
```

[2]See Chapter 2, page 28, for details on ISPL (Interpreted Systems Programming Language).

```
        $\mathbf{mapAg}(ag_A, \langle\sigma^+, A\rangle)$, $\mathbf{mapSubst}(A, map().put(\langle\sigma^+, A\rangle))$, $\mathbf{agRole}(A, vector\langle Agent\rangle)$
        */
        $ag_A$ :=$\mathbf{denominate}(\sigma, A)$;
        $Ag := Ag \cup \{ag_A\}$;
    endfor
endfor
4).
find $\sigma \in \Sigma_P$, $\sigma$ for $\mathbf{I}$
    $Env$ :=$\mathbf{denominate}(\sigma, \mathbf{I})$;
    $Ag := Ag \cup \{Env\}$;
endfind
5). /* initial views*/
for each $A \in Ho$
    for each $\sigma \in \Sigma_{Pr}^{Leg}|_{\omega|_A}$
        $ag := mapSubst.get(A).get(\sigma^+)$
        $\mathbf{createInitialViews}(ag, views^0)$
            $views^0 := views^0 \cup \{view_{ag}^0\}$
            //accurately, $views^0.put(ag, view_{ag}^0)$;
    endfor
endfor
6)./*local actions of agents of role $A$, any $A \in Ho$*/
for each $A \in Ho$
    for each $\sigma \in \Sigma_{Pr}^{Leg}|_{\omega|_A}$
        $ag := mapSubst.get(A).get(\sigma^+)$
        $L\_act_{ag}.create()$; //or accurately, $Act.put(ag, vector())$;
        for each $(a \in \omega|_A)$
            if $(a = A!B : (M)t,\ B \in Ho)$
                //$\mathcal{R}_t^{ag}$ calculated at $step$ 2)., in $createRanges(Pr, \Sigma_{Pr}^{=})$
                for all $x \in \mathcal{R}_t^{ag}$ (6.1)
                    $act$ := create(send, t, x);
                    $L\_act_{ag} := L\_act_{ag} \cup \{act\}$;
                    //accurately, $Act.get(ag).add(act)$;
                endfor
            endif
            else if $(a = A?B : t,\ B \in Ho)$
                if $(act(\text{``}receive\text{''}) \notin L\_act_{agA})$
                    $act$ := create(receive);
                    $L\_act_{ag} := L\_act_{ag} \cup \{act\}$;
                    //accurately, $Act.get(ag).add(act)$;
                    continue;
                endif
            endif
        endfor
    $act_1$ := create(wait); $act_2$ := create(empty);
    $L\_act_{ag} := L\_act_{ag} \cup \{act_1, act_2\}$;
    //accurately, $Act.get(ag).add(act_1)$; $Act.get(ag).add(act_1)$
    endfor
endfor
7)./*local actions of agent $Environment$, hence the $intruder$*/
/* sending to honest agents*/
for each $A \in Ho$
```

```
        for each $\sigma \in \Sigma_{Pr}^{Leg}|_{\omega|_A}$
            $ag := mapSubst.get(A).get(\sigma^+)$
            for each $(a \in \omega|_A)$
                if $(a = A?B : t,\ B \in Ho)$
                    for $x \in \mathcal{R}_t$ (7.1)
                        $act$ :=**create(transmit, ag, t, x)**;
                        $L\_act_{Env} := L\_act_{Env} \cup \{act\}$;
                        //accurately, $Act.get(Env).add(act)$;
                    endfor
                endif
            endfor
        endfor
    endfor
    /* possibly analysing any composite*/
    for each $t \in Composites$
        for $x \in \mathcal{R}_t$
            $act$ :=**create(analz, t, x)**;
            $L\_act_{Env} := L\_act_{Env} \cup \{act\}$;
            //accurately, $Act.get(Env).add(act)$;
        endfor
    endfor
    /* possibly forging any message*/
    for each $t \in Msg$
        for $x \in \mathcal{R}_t$
            $act$ :=**create(forge, t, x)**;
            $L\_act_{Env} := L\_act_{Env} \cup \{act\}$;
            //accurately, $Act.get(Env).add(act)$;
        endfor
    endfor
    /* possibly intercepting from any agent*/
    for each $ag \in Ag \setminus Env$
        $act$ := **create(intercept_from, ag)**;
        $L\_act_{Env} := L\_act_{Env} \cup \{act\}$;
        //accurately, $Act.get(Env).add(act)$;
    endfor
    /* the empty action*/
    $act$ :=**create(empty)**;
    $L\_act_{Env} := L\_act_{Env} \cup \{act\}$;
    //accurately, $Act.get(Env).add(act)$;
    8). /*local protocol of an agent of role $A$, any $A \in Ho$*/
    for each $A \in Ho$
        for each $\sigma \in \Sigma_{Pr}^{Leg}|_{\omega|_A}$
            $ag := mapSubst.get(A).get(\sigma^+)$
            $P_{ag} := P_{ag}.create()$;
            for each $(a \in \omega|_A)$
                find $r \in Rules\ r \rightsquigarrow a$;
                $n := step(r)$; (8.0) //hence, event $(\omega|_A, \sigma, n)$ in $M_{CAPSL}$
                if $(a = A!B : (M)t,\ B \in Ho)$
                    $\overline{t_1 \ldots t_n} := OSub(t)$;
                    for $x \in \mathcal{R}_t^{ag}$
                        $act := L\_act_{ag}.get(send(t, x))$; (8.1)
                        $\overline{x_1 \ldots x_n} := OSub(x)$; (8.2)
                        $P_{ag} := P_{ag}.add([step = n \wedge t_1 = x_1 \wedge \ldots \wedge t_n = x_n] : act)$ (8.3);
                    endfor
                endif
                else if ($a = A?B : t$, $B \in Ho$)
                    $act_1 := L\_act_{ag}.get(receive)$ (8.4);
                    $act_2 := L\_act_{ag}.get(wait)$ (8.5);
                    $P_{ag} := P_{ag}.add([step = n] : act_1)$ (8.6);
                    $P_{ag} := P_{ag}.add([step = n] : act_2)$ (8.7);
                endif
            endfor
            $act := L\_act_{ag}.get(``empty'')$ (8.8);
            $P_{ag} := P_{ag}.add([step = last(Steps^A) + 1] : act)$ (8.9);
        endfor
    endfor
    9). /*local protocol of the $Environment$ agent*/
    $P_{ag} := P_{ag}.create([\lambda] : L\_act_{Env})$;
    10). /*local evolution of an agent of role $A$, any $A \in Ho$*/
    for each $A \in Ho$
        for each $\sigma \in \Sigma^{Leg}_{Pr}|_{\omega|_A}$
            $ag := mapSubst.get(A).get(\sigma^+)$
            $E_{ag} := E_{ag}.create()$
            for each ($a \in \omega|_A$)
                find $r \in Rules\ r \sim> a$;
                n:=step(r); //hence, event $(\omega|_A, \sigma, n)$ in $M_{CAPSL}$
                if ($a = A!B : (M)t$, $B \in Ho$)
                    $\overline{t_1 \ldots t_n} := OSub(t)$;
                    for $x \in \mathcal{R}^{ag}_t$
                        $\overline{x_1 \ldots x_n} := OSub(x)$ (10.0);
                        $act_1 := L\_act_{ag}.get(send(t, x))$ (10.1);
                        $act_2 := L\_act_{Env}.get(intercept\_from(ag))$ (10.2);
                        $preconds := list()$;
                        $preconds := preconds.add((ag.Action = act_1))$ (10.3);
                        $preconds := preconds.add((Env.Action = act_2))$ (10.4);
                        $preconds := preconds.add((ag.step = n))$ (10.5);
                        $preconds := preconds.add((t_1 = x_1 \wedge \ldots \wedge t_n = x_n))$ (10.6);
                        $postconds := list()$;
                        $postconds := postconds.add((ag.step = next(Steps^A, n)))$ (10.7);
                        $E_{ag} := E_{ag}.add(``if\ \langle preconds\rangle\ then\ \langle postconds\rangle\ ")$;
                    endfor
                endif
                else if ($a = A?B : t$, $B \in Ho$)
                    $\overline{t_1 \ldots t_n} := OSub(t)$;
                    /* for $set$ */
                    // $I \subsetneq \{1, \ldots n\}$
                    $T_I := (LearnedAtoms^A \setminus n\text{–}LearnedAtoms^A) \cap OSub(t)$; (10.8)
                    /* for $out\_match$ and $decryptable$ */
                    $T_J := n\text{–}LearnedAtoms^A \cap OSub(t)$; //$J \subsetneq \{1, \ldots n\}$ (10.9)
                    /* for $in\_match$ */
                    $D :: list\langle Pairs\rangle.create()$; $D := duplicates(OSub(t))$; (10.10)
                    for $x \in \mathcal{R}_t$
```

```
                    $\overline{x_1 \ldots x_n} := OSub(x)$;
                    $act_1 := L\_act_{ag}.get(receive)$;
                    $act_2 := L\_act_{Env}.get(transmit(ag, t, x))$;
                    $preconds := list()$;
                    $preconds := preconds.add((ag.Action = act_1))$ (10.11);
                    $preconds := preconds.add((Env.Action = act_2))$ (10.12);
                    $preconds := preconds.add((ag.step = n))$ (10.13);
                    for each $t_k \in T_J$
                        $preconds := preconds.add((t_k = x_k))$ (10.14);
                    endfor
                    for each $(k, k') \in D$
                        $preconds := preconds.add((x_k = x'_k))$ (10.15);
                    endfor
                    $postconds := list()$;
                    for each $t_k \in T_I$
                        $postconds := postconds.add((t_k := x_k))$ (10.16);
                    endfor
                    $postconds := postconds.add((ag.step = next(Steps^A, n)))$ (10.17);
                    $E_{ag} := E_{ag}.add($"$if \langle preconds \rangle then \langle postconds \rangle$"$)$;
                endfor
            endif
        endfor
    endfor
endfor
/*local evolution of the $Environment$ agent */
$E_{Env} := E_{Env}.create()$;
11). // the intruder intercepting messages
for each $A \in Ho$
    for each $\sigma \in \Sigma^{Leg}_{Pr}|_{\omega|_A}$
        $ag := mapSubst.get(A).get(\sigma^+)$
        for each $(a \in \omega|_A)$
            if $(a = A!B : (M)t,\ B \in Ho)$
                $\overline{t_1 \ldots t_n} := OSub(t)$;
                for $x \in \mathcal{R}^{ag}_t$
                    $\overline{x_1 \ldots x_n} := OSub(x)$;
                    $act_1 := L\_act_{ag}.get(send(t, x))$;
                    $act_2 := L\_act_{Env}.get(intercept\_from(ag))$;
                    $preconds := list()$;
                    $preconds := preconds.add((ag.Action = act_1))$;
                    $preconds := preconds.add((Env.Action = act_2))$;
                    $postconds := list()$;
                    $p_1$:="$Env.msg\_log := Env.msg\_log \cup \{t, ag, x\}$" (11.1);
                    $postconds := postconds.add(p_1)$ (11.2);
                    for $k = \overline{1, n}$;
                        $p_2$:="$Env.at\_log := Env.at\_log \cup \{t_k, t, ag, x_k\}$" (11.3);
                        $postconds := postconds.add(p_2)$ (11.4);
                    endfor;
                    $p_3$:="$Env.analz\_log := Env.analz\_log \cup \{t, x\}$" (11.5);
                    $postconds := postconds.add(p_3)$ (11.6);
                    $p_4$:="$do\_analz := true;$" (11.7);
                    $postconds := postconds.add(p_4)$ (11.8);
                $E_{ag} := E_{ag}.add(``if\ \langle preconds\rangle\ then\ \langle postconds\rangle\ ")$;
            endfor;
        endfor;
    endfor;
12). //analysis and something new analysed
    for each $t' \in Composites$
        for each $x' \in \mathcal{R}_t$
                $i := last(t')$;
                //decryptable composite
                $p_1$:=“$((t'_i, x'_i) \in values\_log) = true \wedge (do\_analz = true)$;” (12.1)
                $preconditions.add(p_1)$; (12.2)
                //$t^{''}$ encrypted within $t^{'}$
                for $t^{''} \in ESub(t^{'})$ $(f1)$
                    //$t^{''}$ not “seen” before by the Intr.
                    $p_2$:=“$(t^{''}, x^{''}) \notin analz\_log$” (12.3)
                    $preconditions.add(p_2)$; (12.4)
                    //sth. new emerged from analysing $t^{'}$
                    $p_3$:=“$flag\_analz[count_1] := true$;” (12.5)
                    $postconditions.add(p_3)$; (12.6)
                    //add the “novelty” to be later analysed
                    $p_4$:=“$analz\_log := analz\_log \cup (t^{''}, x^{''})$;” (12.7)
                    $postconditions.add(p_4)$ (12.8)
                    $E_{ag} := E_{ag}.add(``if\ \langle preconds\rangle\ then\ \langle postconditions\rangle")$
                    // $t^{'''}$ is an atom in plain in $t^{'}$
                    for $t^{'''} \in (OSub(t^{'}) \setminus OSub(t^{''}))$ $(f2)$
                        //$t^{'''}$ not “seen” so far
                        $p_2$ :=“ $(t^{'''}, x^{'''}) \notin values\_log$” (12.3′)
                        $preconditions.add(p_2)$; (12.4′)
                        //something new emerged
                        $p_3$:=“$flag\_analz[count_1] := true$;” (12.5′)
                        $postconditions.add(p_3)$; (12.6; )
                        //add the “novelty”
                        $p_4$:=“$values\_log := values\_log \cup (t^{'''}, x^{'''})$;” (12.7′)
                        $postconditions.add(p_4)$ (12.8′)
                    endfor
                endfor
            //count the composite $t^{'}$ as analysed
            $p_5$:=“$count_1 + +$;” (12.9)
            $postconditions.add(p_5)$ (12.10)
        endfor
    endfor
13). //analysis and nothing new emerging (anymore)
    for each $t' \in Composites$
        for each $x' \in \mathcal{R}_t$
            $i := last(t')$;
            //$t'$ decryptable
            $p_1$:=“$((t'_i, x'_i) \in values\_log) = true \wedge (do\_analz = true)$;” (13.1)
            $preconditions.add(p_1)$; (13.2)
            //$t^{''}$ encrypted with $t'$
            for $t^{''} \in ESub(t^{'})$
                //$t^{'''}$ plain atom in $t^{'}$
```

```
                  for $t''' \in OSub(t') \setminus OSub(t'')$
                    //nothing new emerges
                    $p_2$:="$((t'', x'') \in analz\_log \wedge (t''', x''') \in values\_log)$" (13.3)
                    $preconditions.add(p_2)$;
                   $p_3$:="$flag\_analz[count_1] := false$;" (13.4)
                    $postconditions.add(p_3)$;
                    endfor
                  endfor
                  //count the attempt of analysis
                  $p_5$:="$count_1 + +$;" (13.5)
                  $postconditions.add(p_5)$;
                  $E_{ag} := E_{ag}.add($"$if\ \langle preconds\rangle\ then\ \langle postconditions\rangle$"$)$
            endfor
        endfor
14). //analysis not possible
        for each $t' \in Composites$
            for each $x' \in \mathcal{R}_t$
                $i = last(t')$;
                //$t'$ not decryptable
                    $p_1$:="$((t'_i, x'_i) \notin values\_log) = true \wedge (do\_analz = true)$;" (14.1)
                $preconditions.add(p_1)$; (14.2)
                //record the impossibility to analyse $t'$
                $p_3$:="$flag\_analz[count_1] := false$;" (14.3)
                $postconditions.add(p_3)$;
                // count the attempt to analyse $t'$
                $p_5$:="$count_1 + +$;" (14.4)
                $postconditions.add(p_5)$;
                $E_{ag} := E_{ag}.add($"$if\ \langle preconds\rangle\ then\ \langle postconditions\rangle$"$)$
                endfor
            endfor
        endfor
15). //analz-closure not acquired yet
        p1:="$(count_1 = \sharp maxSubstComposites \wedge$
        $(flag\_analz[0] = true \vee \ldots \vee flag\_analz[\sharp maxSubstComposites] = true);)$"
        $preconditions.add(p_1)$;
        $p_2$:="$(count_1 := 0)$;"
        $postconditions.add(p_2)$;
        $E_{ag} := E_{ag}.add($"$if\ \langle preconds\rangle\ then\ \langle postconditions\rangle$"$)$
16). //analz–closure acquired
        p1:="$(count_1 = \sharp maxSubstComposites \wedge$
        $(flag\_analz[0] = false \wedge \ldots \wedge flag\_analz[\sharp maxSubstComposites] = false);)$"
        $preconditions.add(p_1)$;
        $p_2$:="$(stop\_analz := true)$;"
        $postconditions.add(p_2)$;
        $E_{ag} := E_{ag}.add($"$if\ \langle preconds\rangle\ then\ \langle postconditions\rangle$"$)$
17). //start applying synth over the analysis
        $p_1$:="$stop\_analz = true$;"
        $preconditions.add(p_1)$;
        $p_2$:="$do\_analz := false;\ do\_forge := true$;"
        $postconditions.add(p_2)$;
        $E_{ag} := E_{ag}.add($"$if\ \langle preconds\rangle\ then\ \langle postconds\rangle$"$)$;
```

```
18). //synthesis
          for $t \in Msg$
             $//\{t_1 \dots t_n\} := Sub(t);$
             for $x \in \mathcal{R}_t$ (18.1)
                $\overline{x_1 \dots x_n} := Sub(x);$
                //succesful "forge"
                $act := L\_act_{Env}.get(forge(t, x));$
                $preconds := list();$
                $p_1$:="$(do\_forge = true) \wedge (count_2 < \sharp maxSubstMsg)$;" (18.2)
                $preconds.add(p_1);$
                $preconds.add(Env.Action = act);$
                $preconds.add((x_1 \in Env.POOL \wedge \dots \wedge x_n \in Env.POOL));$ (18.3)
                $postconds := list();$
                $p$:="$synth\_log := synth\_log \cup \{(t, x)\}$" (18.4);
                $p'$:="$count_2 + +$" (18.5);
                $postconds.add(p), postconds.add(p')$
                $E_{ag} := E_{ag}.add($"$if\ \langle preconds \rangle\ then\ \langle postconds \rangle$ ");
                //unsuccesful "forge"
                $act := L\_act_{Env}.get(forge(t, x));$
                $preconds := list();$
                $p_1$:="$(do\_forge = true) \wedge (count_2 < \sharp maxSubstMsg)$" (18.2′)
                $preconds.add(p_1);$
                $preconds.add(Env.Action = act);$
                $preconds.add((x_1 \notin Env.POOL \vee \dots \vee x_n \notin Env.POOL));$ (18.3′)
                $postconds := list();$
                $p'$:= "$count_2 + +$",   $postconds.add(p')$ (18.5′);
                $E_{ag} := E_{ag}.add($"$if\ \langle preconds \rangle\ then\ \langle postconds \rangle$");
             endfor
          endfor
19). //synth closure
       $preconds := list();$
       $p_1$:="$(count_2 = \sharp maxSubstMsg)$" (19.1)
       $preconds.add(p_1);$
       $postconds := list();$
       $p_2$:="$(do\_forge := false)$" (19.2)
       $postconds.add(p_2);$
       $E_{ag} := E_{ag}.add($"$if\ \langle preconds \rangle\ then\ \langle postconds \rangle$");
    endif
    else if $a = A?B : t$
    $//\overline{t_1 \dots t_n} := OSub(t);$
       for $x \in \mathcal{R}_t$
       $\overline{x_1 \dots x_n} := OSub(x);$
          $act_1 := L\_act_{Env}.get(transmit(ag, t, x));$
          $act_2 := L\_act_{ag}.get(receive);$
          $preconds := list();$
          $preconds.add(Env.Action = act_1);$
          $preconds.add(ag.Action = act_2);$
          $preconds.add((*, t, x) \in Env.msg\_log \vee (t, x) \in Env.synth\_log));$
          $postconds := list();$
          $E_{ag} := E_{ag}.add($"$if\ \langle preconds \rangle\ then\ \langle postconds \rangle''$);
       endfor          endif
```

Comments were inserted in the description of the algorithm *tr* to ease the understanding. Also, the design of algorithm *tr* makes it transparent that the output of *tr* respects the semantics in Chapter 3. Nonetheless, in the following, we briefly explain the algorithm *tr*.

Explanations on the Algorithm *tr*. Entry 1 of the algorithm adds designated symbols to the signature of M_{CAPSL} in order to form the signature $\mathcal{S}$ of Υ_{IS}; sets of terms on $\mathcal{S}$, upon Chapter 3, are also constructed at this step. Following procedures (1)–(2″) shown in Section 4.1, entry 2 uniformly extends the set Σ_{Pr} of role-substitutions encountered in $Ev^{Pr}_{M_{CAPSL}}$ to the set Σ^{+}_{Pr} of initial Ho^{+}-freshness enforcing role-substitutions. A map *SubstMap* is built: for each σ a role-substitution in M_{CAPSL}, the corresponding σ^{+} is made available (i.e., $\sigma \cong \sigma^{+}$). As in procedures (1)–(2″), the algorithm *tr* constructs ranges which are large enough for the set Σ^{+}_{Pr} of initial Ho^{+}-freshness enforcing role-substitutions to operate on. For (non-atomic) terms t, the ranges $\mathcal{R}^{\sigma^{+}}_{t}$ restricted under the newly built role-substitutions are also created (i.e., as per Definition 3.3.13, for $t \in \mathcal{T}$, the range $\mathcal{R}^{\sigma^{+}}_{t}$ is the generalised Cartesian product of the restricted ranges of its atoms). Entries 3, 4 and 5 denominate agents and create the initial states, according to the set Σ^{+}_{Pr} of role-substitutions.

For each possible event underpinning an *A-role* under instantiation σ in M_{CAPSL}, entry 6 creates a corresponding local action of $ag^{\sigma^{+}}_{A}$ in Υ_{IS}. Additionally, as per Chapter 3, actions *empty* and *wait* are added to the set of possible local actions of Υ_{IS}. Entry 7 creates the local actions of the *Environment* agent with respect to the given set of substitutions Σ^{+}_{Pr} (e.g., as per Chapter 3, $analz(t', x')$ for each $t' \in Composites$, $x' \in \mathcal{R}_{t'}$; $forge(t, x)$, for each $t \in Msg$, $x \in \mathcal{R}_{t}$; $intercept\_from(ag)$, for each $\sigma^{+}(A\text{-}role) \mapsto ag$, $\sigma \cong \sigma^{+}$, for each $\sigma \in \Sigma_{Pr}$ extracted from $Ev^{Pr}_{M_{CAPSL}}$, for each $A \in Ho$). Entries 8 and 9 create the local protocol of each $ag^{\sigma^{+}}_{A}$ and of the *Environment*. For instance, if in M_{CAPSL} an *A-role* under σ sends t at step n, the corresponding local action is made possible at a local state $l@n$ of $ag^{\sigma^{+}}_{A}$ if $construct^{\mathbb{I}}(t, l_{ag^{\sigma^{+}}_{A}}@n, view^{0}_{ag^{\sigma^{+}}_{A}})$ (i.e., lines (8.0)–(8.3)). As per Chapter 3, entry 9 makes any of the actions of the *Environment* possible at any of its local states.

For all $A \in Ho$, for all σ in M_{CAPSL}, the *A-role* instantiated under σ in M_{CAPSL} and $\sigma^{+} \cong \sigma$, entry 10 outputs the evolution function of corresponding agents $ag^{\sigma^{+}}_{A}$ in Υ_{IS}. The output for send events is constructed similarly to the case of the local protocol (i.e., lines (10.5) and (10.6)).

Entries (10.1)–(10.4) additionally enforce, as per Chapter 3, that the action $send(t, x)$ of agent $ag_A^{\sigma^+}$ triggers only in synchronisation with the action $intercept\_from(ag_A^{\sigma^+})$ on the *Environment*'s side. Algorithm *tr* also adds the evolution post-condition for consequently incrementing the value of the step-variable of $ag_A^{\sigma^+}$ (i.e., line (10.7)).

We explain the cases for receiving a message t. By (10.9), the set T_J is the set of elements that appear in t and that $ag_A^{\sigma^+}$ has already seen in her execution, i.e., as per Chapter 3, T_J is the set $n\text{-}LearnedAtoms^A$, for the current step n. By (10.8), the atoms in the set T_I are atoms appearing in t and not learned by $ag_A^{\sigma^+}$ previous to the current step n. Then, the precondition in (10.14) expresses "$if\ ((out\_match(l_{ag_A^{\sigma^+}}@n, t) = true) \wedge (decryptable(l_{ag_A^{\sigma^+}}@n, t) = true))$", aligned with Chapter 3. The $\varUpsilon_{IS}$ semantics for the local evolution function described in Chapter 3 is enforced in the output of algorithm *tr*: the agent accepts the messages if the tests on decryption and matching succeed. If so, the output in line (10.17) makes her local step be incremented and the output in line (10.16) shows the assignment of the atoms in T_I to the values received (i.e., line (10.16) implements $set^{\mathbb{I}}(l_{ag_A^{\sigma^+}}@n, t)$ presented in Chapter 3).

The synchronisation $ag.send(t, x)$ and $Env.intercept\_from(ag)$ is the output of *tr* in entry 11. It further enforces that the Environment records tuples message-sender-value (i.e., (11.1) – (11.2), (11.3) –(11.4)), to store the message $[x/t]$ (i.e., (11.5)) and to trigger the analysis (i.e., variable *do_analz* is set to true in (11.5)).

The output achieved through lines 12–16 depicts the closure of an arbitrary, reached local state of the *Environment* under analysis operations. Entry 12 depicts the cases where new terms emerge through Dolev-Yao analysis. To illustrate, assume that composite $t^{'} = \{\{t_1, t_2\}_{k_2}, t_3\}_{k_1}$ is under analysis. Then, $t^{''} = \{t_1, t_2\}_{k_2}$ (i.e., an encrypted part within $t^{'}$) is an iterator of the for loop annotated $(f1)$. In turn, $t^{'''} = t_3$ (i.e., an un-encrypted, "in-plain" atom within $\{t_1, t_2\}_{k_2}, t_3)$ is an iterator of the for loop annotated $(f2)$. If $t^{''}$ has not been "seen" already (i.e., (12.3)), then the current iteration is a successful analysis trial (i.e., (12.5)). Also, if $t^{'''}$ is newly acquired via the current decryption, then the analysis trial counts as successful (i.e., (12.5′)). The value for composite $t^{''}$ is added to the *analz_log* (i.e., (12.7)), whereas the value for atom $t^{'''}$ is added to the *value_log*. Synonymously, the denotation of $analz^{\mathbb{I}}(values\_log, analz\_log, t')$ is expressed through this part of *tr*'s output. Entries (13.3) and (13.4) output the evolution lines for the cases where the analysis is unsuccessful, i.e., no new terms emerge. Entry 14 produces the case where the analysis fails due to the lack of decryption keys for t'. Entries 15 and 16 generate the test whether the actual closure under analysis has been produced. If the closure is attained (i.e., condition p_1 in entry 16 denoting that in a full set of analz-type iteration no new data has been produced), then the analysis is ceased (i.e., *stop_analz* $:=$ *true*). Otherwise (i.e., condition p_1 in entry 15 denoting that at least one new term has been produced in the last round of analysis), a new analz cycle starts (i.e., all the evolutions lines generated by 14–16 will be triggered again). We show in Appendix A, Section A.1.*II* that indeed repetitive applications of $analz^{\mathbb{I}}$ as considered in Υ_{IS} and produced by algorithm *tr* reach a fixpoint, as described in the above. In this algorithm for optimisation purposes the variables *already_analz* (used in Chapter 3) have been dismissed from the local state of the Environment; their functionalities have been embedded however in the entries 13–16 by using explicit `ISPL`-like syntax.

According to entry 17, if the analysis cycle is stopped (hence, *stop_analz* is set to *true*) then the synthesis actions are enabled (i.e., post-condition p_2 in entry 17, where *do_synth* is set to true). When an analysis cycle finishes, *analz_log* and *values_log* give part of the object *POOL* in the local state of the *Environment*, as described in Chapter 3. Entry 18 in the algorithm *tr* produces the part of the Environment's evolution responsible for $synth^{\mathbb{I}}(t, POOL)$.

For each receive-event of a step n in the σ-instantiated *A-role* in M_{CAPSL}, the agent $ag_A^{\sigma^+}$ in Υ_{IS}, at step n, receives $[x/t]$ from the intruder (i.e., entry 19). For the intruder to transmit $[x/t]$ it is either that he has simply intercepted it or he has forged it (i.e., $[x/t] \in$ *synth_log*).

For a procedure to be an algorithm, D. Knuth identifies in [113] that it needs to be *finite*, *definite*, *effective* and *with some output*. Then, following the meaning of these characterisations in [113], we conclude that the procedure *tr* is indeed an algorithm, as it is:

- finite (given the finiteness of number of roles, role-instantiations, events);
- definite (given that each part has been defined/explained in the previous sections);
- effective (given that all steps are finite);
- with some output (given that the output can be clearly observed to be a model Υ_{IS}^{Pr} upon the descriptions in Chapter 3).

By inspection of the algorithm tr, we conclude that its worst-case complexity is in fact the worst-case complexity of generating the evolution function. In this generation, the entire range of a message t is iterated over, within an outer loop over all possible instantiations for an arbitrary *A-role*. Hence, both these loops have a complexity order which is exponential in the size of the protocol, i.e., $\mathcal{O}(2^{2^{poly(size(Pr))}})$. The construction of initial Ho^+-freshness enforcing role-substitutions in Υ_{IS}, given at the beginning of the algorithm tr, is also exponential in the size of the protocol. Thus, the worst-case complexity of the algorithm tr is that of a double-exponential factor in the size of the protocol Pr.

4.2.2 A Correspondence Between Goal-Specification in M_{CAPSL} and Υ_{IS}

In Section 4.2.1 we have presented an algorithm that given M^{Pr}_{CAPSL} produces the interpreted system Υ^{Pr}_{IS}. To complete this high-level correlation between the two models, in this section we relate the expressions of security goals in each of the two formalisms.

To begin with, we reiterate that the meaning of `CAPSL` goals is presented only informally in [64, 65, 73]. Therefore, we will now explicitly relate the `CAPSL` goals with their formulation in M_{CAPSL} by giving an *expression-relation* ρ_{CAPSL}, similar to the relation ρ_{IS} in Chapter 3. This add-on to the M_{CAPSL} model is purely notational and we introduce it for two reasons: *1)* clarity of the relations established between the expression of goals in Υ_{IS} and in M_{CAPSL}, respectively; *2)* conciseness of proofs to follow in this chapter.

Specifications of `CAPSL` goals in M_{CAPSL}. Let $\mathcal{G}$ be the set of `CAPSL` goals and let the set $\mathcal{G}' \subsetneq \mathcal{G}$ designate the atomic `CAPSL` goals. The expression-relation ρ_{CAPSL} is defined on $\mathcal{G}'$ in the following.

Definition 4.2.1 (Initial Secrecy in M_{CAPSL}) *Let D be a `CAPSL` description for an RT protocol Pr with `HOLDS: A t` and the atomic goal $g =$ `SECRET t` amongst the `CAPSL` assertions. Let $s_0 \in S_0$, $s \in S$ and ξ be an arbitrary run in M^{Pr}_{CAPSL} such that $s_0[\xi\rangle s$. The formulation of the goal g in M^{Pr}_{CAPSL} is given by $\rho_{CAPSL}(g) = t \in analz(s_A) \setminus analz(s_I)$, for $t \in Secret_A$, $A \in Ho$ (where t, A are terms on the signature $\mathcal{S}$ respectively corresponding to the `CAPSL` variables `t` and `A` present in the assertions of the description D).*

Definition 4.2.1 expresses that if assertions `HOLDS: A t` and `SECRET t` both appear in the `CAPSL` protocol description, then the M_{CAPSL} formulation of one security requirement of the protocol is initial secrecy, i.e., $t \in analz(s_A) \setminus analz(s_I)$, for $t \in Secret_A$ and $A \in Ho$.

Definition 4.2.2 (Secrecy in M_{CAPSL}) *Let D be a `CAPSL` description for an RT protocol Pr with the atomic goal g = `SECRET t` and with no `CAPSL` assertion of type `HOLDS: A t`. Let $s_0 \in S_0$, $s \in S$ and ξ be an arbitrary run in M^{Pr}_{CAPSL} such that $s_0[\xi\rangle s$. The formulation of the goal g in M^{Pr}_{CAPSL} is given by $\rho_{CAPSL}(g) = t \in analz(s_A) \setminus analz(s_I)$, for $t \in T_0$, $A \in Ho$ (where t, A are terms on the signature $\mathcal{S}$ respectively corresponding to the `CAPSL` variables `t` and `A` present in the assertions of the description D).*

Definition 4.2.2 expresses that if assertion `SECRET t` appears in the `CAPSL` protocol description, whereas the assertion `HOLDS: A t` does not appear in the same description, then the M_{CAPSL} formulation of one security requirement of the protocol is secrecy, i.e., $t \in analz(s_A) \setminus analz(s_I)$, for $t \in T_0$ and $A \in Ho$.

In [188] there is no explicit expression for agreement. However, in MSR and [188], the notion of agreement is subsumed to expressions of reachability, similar to those of secrecy. We explicitly lift the expression-relation ρ_{CAPSL} that we introduced above, to support agreement formulations as informally denoted in MSR. Again, we operate this purely notational add-on to M_{CAPSL} only for ease of relating the M_{CAPSL} model with the Υ_{IS} model.

Definition 4.2.3 (Agreement on Initials in M_{CAPSL}) *Let D be a `CAPSL` description for an RT protocol Pr with the `CAPSL` assertion `HOLDS A:t` and the atomic goal g=`PRECEDES A,B:t` or g=`AGREE A,B:t`. Let $s_0 \in S_0$, $s \in S$, ξ be a maximal run with respect to A in M^{Pr}_{CAPSL} such that $s_0[\xi\rangle s$ (i.e., A has ended her role at s). The formulation of the goal g in M^{Pr}_{CAPSL} is given by $\rho_{CAPSL}(g) = t \in analz(s_A) \cap analz(s_B) \setminus analz(s_I)$, for $t \in Secret_A$, for $A, B \in Ho$ (where t, A, B are terms on the signature $\mathcal{S}$ respectively corresponding to the `CAPSL` variables `t`, `A` and `B` present in the assertions of the description D).*

Definition 4.2.4 (Agreement in M_{CAPSL}) *Let D be a `CAPSL` description for an RT protocol Pr with the atomic goal g = `PRECEDES A,B:t` or g = `AGREE A,B:t` and with no `CAPSL` assertion of* *type* `HOLDS A:t`. *Let $s_0 \in S_0$, $s \in S$, ξ be a maximal run with respect to A in M^{Pr}_{CAPSL} such that $s_0[\xi\rangle s$ (i.e., A has ended her role at s). The formulation of the goal g in M^{Pr}_{CAPSL} is given by $\rho_{CAPSL}(g) = t \in analz(s_A) \cap analz(s_B) \setminus analz(s_I)$, for $t \in T_0$, for $A, B \in Ho$ (where t, A, B are terms on the signature $\mathcal{S}$ respectively corresponding to the* `CAPSL` *variables* `t`, `A` *and* `B` *present in the assertions of the description D).*

4.2.2.1 Security Specifications in M_{CAPSL} versus Υ_{IS}

We can now correlate the expression of `CAPSL` goals in M_{CAPSL} and the expressions of `CAPSL` goals in Υ_{IS}, i.e., relate ρ_{CAPSL} and ρ_{IS}. Such an association can only refer to atomic `CAPSL` goals (hence, the domain of ρ_{CAPSL}). Furthermore, it can only regard specifications of initial secrecy and agreement on initials (hence, in Υ_{IS} the generation of atoms is encompassed into a fully instantiated initial setup). Such a correspondence between the formulations of goals in the two models is needed in order to express more easily the preservation of goal satisfaction/refutation from one model to another.

Definition 4.2.5 (Correspondence on Initial Secrecy) *Let D be a* `CAPSL` *description for an RT protocol Pr with the atomic goal $g =$* `SECRET t` *and the* `CAPSL` *assertion* `HOLDS: A t`. *Let $\rho_{CAPSL}(g)$ and $\rho_{IS}(g)$ give the formulations of g in M^{Pr}_{CAPSL} and Υ^{Pr}_{IS} respectively. We say that $\rho_{CAPSL}(g)$ and $\rho_{IS}(g)$ are* in correspondence *and we write $\rho_{CAPSL}(g) \asymp \rho_{IS}(g)$.*

Definition 4.2.6 (Correspondence on Agreement on Initials) *Let D be a* `CAPSL` *description for an RT protocol Pr with the* `CAPSL` *assertion* `HOLDS A:t` *and the atomic goal $g =$* `AGREE A,B:t`. *Let $\rho_{CAPSL}(g)$ and $\rho_{IS}(g)$ give the formulations of g in M^{Pr}_{CAPSL} and Υ^{Pr}_{IS} respectively. We say that $\rho_{CAPSL}(g)$ and $\rho_{IS}(g)$ are* in correspondence *and we write $\rho_{CAPSL}(g) \asymp \rho_{IS}(g)$.*

Definitions 4.2.5 and 4.2.6 simply express that formulations of initial secrecy and agreement on initial in the two models are correlated, i.e., their satisfaction/refutation in the two models is to be investigated.

For an atomic goal g such that $\rho_{CAPSL}(g) \asymp \rho_{IS}(g)$, we are going to show the following: when algorithm tr is used to build Υ^{Pr}_{IS} from M^{Pr}_{CAPSL}, the traditional semantics M^{Pr}_{CAPSL} validates $\rho_{CAPSL}(g)$ if and only if the unwinding of Υ^{Pr}_{IS} validates (all the formulations of) $\rho_{IS}(g)$.

In Section 4.2.1 we have given the *tr* algorithm that produces the Υ_{IS}^{Pr} formalisation starting from an M_{CAPSL}^{Pr} model of a RT protocol Pr. Together with the $\asymp$-relation between the expression of security goals in the two models, Section 4.2 gives a translation schema from the M_{CAPSL} formalism to the Υ_{IS} formalism.

4.3 M_{CAPSL} Model and M_{IS} Model: Related by the Translation Schema

Figure 4.2 relates all the formalisms and models that have been previously discussed in this chapter. In a nutshell, algorithm *tr* translated the trace-based model M_{CAPSL}^{Pr} of the execution of a RT

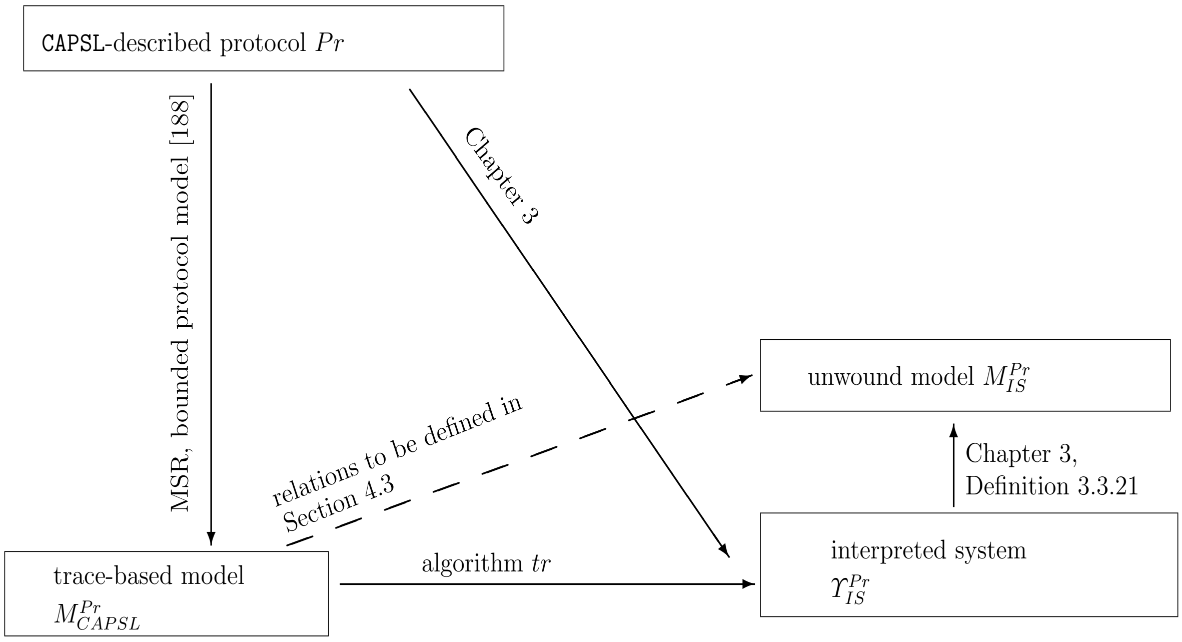

CAPSL-described protocol Pr
MSR, bounded protocol model [188]
Chapter 3
unwound model M_{IS}^{Pr}
relations to be defined in Section 4.3
Chapter 3, Definition 3.3.21
interpreted system Υ_{IS}^{Pr}
trace-based model M_{CAPSL}^{Pr}
algorithm tr

Figure 4.2 The High Level Relationships between M_{CAPSL} and M_{IS}

protocol Pr into an interpreted system Υ_{IS}^{Pr}. In this section we are going to present the relations induced by the algorithm *tr* between the M_{CAPSL}^{Pr} model given at its input and the M_{IS}^{Pr} model, i.e., the unwinding of the Υ_{IS}^{Pr} formalisation given at *tr*'s output. Most importantly, certain relations will be drawn between the states in the two models. In the next sections, we are going to prove that these tr-induced relations between states are such that they describe a homomorphism between the M_{CAPSL} model and its *tr–correspondent*, the unwound M_{IS} model. This will help us prove the preservation of validation/refutation of $\asymp$-related formulations of goals between the two models.

4.3.1 Preamble

In this subsection we present the intuitions behind the relations that we are going to define in Section 4.3.2. We then briefly explain the flow of lemmas and adjacent steps that lead to the proof in Section 4.4 of homomorphism[3] between the M_{CAPSL} and the M_{IS} model.

The Intuitions behind Relations Induced by tr between M_{CAPSL} and M_{IS}.

Intuition 1. The first intuition behind these relations comes precisely from the nature of the transition structures of M_{CAPSL} and M_{IS}.

On the one hand, the transitions in the unwound system of M_{CAPSL} are based on send and receive events in M_{CAPSL}. In M_{CAPSL}, the actions of the intruder (e.g., analysis, synthesis, interception) are embedded in the semantics attributed to the application of such send and receive events. Moreover, the closure of the set X of terms within an arbitrarily given state under Dolev-Yao analysis and synthesis operations is instantaneous (i.e., it is "silent" and takes immediate effect).

On the other hand, the transitions in the underlying structure of M_{IS} are based on joint actions where the Dolev-Yao intercepting, analysing and/or synthesising is not "silent". Moreover, as Chapter 3 and Appendix A explain, the closure under Dolev-Yao analysing and/or synthesising under the bounded-size instantiations in M_{IS} is enforced over a series of joint actions.

Figure 4.3 shows two fragments of computations from M_{CAPSL} and M_{IS}, respectively. These describe the differences between the underlying transition structures, as explained above.

In light of the above, we need to select those relations between the states of M_{CAPSL} and the states of M_{IS} that counterweigh the silent closure under Dolev-Yao analysis and synthesis in M_{CAPSL} and the effectively applied actions in M_{IS} leading to the same closure.

Intuition 2. We are interested in finding a certain homomorphism between the M_{CAPSL} and the M_{IS} model, i.e., a transformation that will preserve the satisfaction/refutation of the $\asymp$-related for-

[3]For the definition of a homomorphism, please refer to Appendix B, Definition B.1.1.

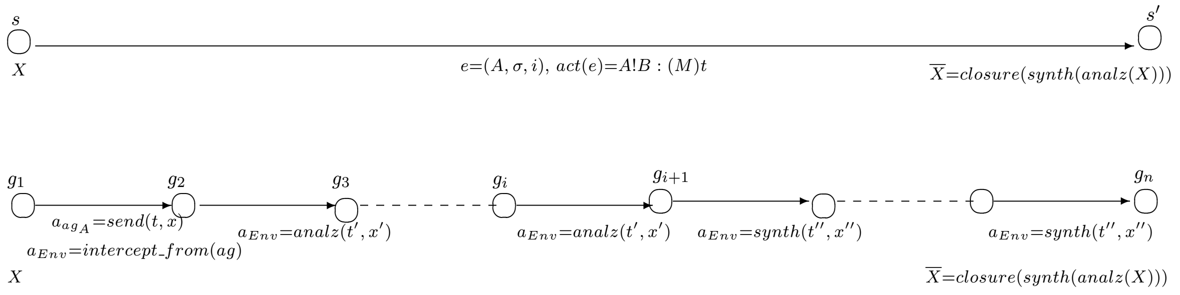

s
X
$e{=}(A,\sigma,i),\ act(e){=}A!B:(M)t$
s'
$\overline{X}{=}closure(synth(analz(X)))$
g_1
g_2
g_3
g_i
g_{i+1}
g_n
$a_{ag_A}{=}send(t,x)$
$a_{Env}{=}intercept\_from(ag)$
$a_{Env}{=}analz(t',x')$
$a_{Env}{=}analz(t',x')$
$a_{Env}{=}synth(t'',x'')$
$a_{Env}{=}synth(t'',x'')$
X
$\overline{X}{=}closure(synth(analz(X)))$

Figure 4.3 Closure under Dolev-Yao Synthesis and Analysis in M_{CAPSL} and in M_{IS}

mulations of CAPSL goals in the two models. Therefore, the formulation of these goals is also a factor in the way we relate the states of the models. Goals in M_{CAPSL} essentially stipulate the intruder being able to analyse secret atomic terms, i.e., $\rho_{CAPSL}(\texttt{SECRET t,HOLDS A:t}){=}t \in analz(s_A) \setminus analz(s_{\mathbf{I}})$, for s being a final M_{CAPSL} state, $A \in Ho$ and $t \in Secret_A$. In the following, we summarise the idea behind a relation between states in M_{CAPSL} and M_{IS} that takes into account the analysis-based formulation of security goals.

The $\tilde{p}$-relation between states, driven by the Dolev-Yao analysis.
In principle, if the intruder is able/unable to analyse the same terms at a state s in M_{CAPSL} as he is able/unable to analyse at a state g in M_{IS}, then we will say that the states are in the $\tilde{p}$-relation, i.e., $s\,\tilde{p}\,g$.

Additionally, note that the M_{CAPSL} states are usually composed of non-atomic terms, whereas the M_{IS} states always comprise only atomic terms, i.e., $s_A \subseteq T$ and $g_{ag_A} \subseteq T_0 \times \mathcal{R}_{T_0}$. Therefore, it is reasonable to use analysis operators in constructing a relation like $\tilde{p} \subseteq S_{M_{CAPSL}} \times G_{M_{IS}}$.

The relation $\tilde{p} \subseteq S_{M_{CAPSL}} \times G_{M_{IS}}$ is essentially driven by the Dolev-Yao analysis of atomic terms. However, in order for the intruder to analyse an atom, he may have to synthesise several non-atomic terms. Then, another aspect to be considered when relating states between the two models is the semantics of Dolev-Yao analysis and synthesis. We summarise this in the following.

The $\tilde{p'}$-relation between states, driven by the Dolev-Yao synthesis.
In principle, if the intruder is able/unable to synthesise the same terms at a state s in M_{CAPSL} as he is able/unable to synthesise at a state g in M_{IS}, then we will say that the states are in the

$\tilde{p}'$-relation, i.e., $s\,\tilde{p}'\,g$.

In this chapter, the $\tilde{p}'$-relation will be given less prominence than the $\tilde{p}$-relation. The $\tilde{p}'$-relation will only be used in showing that the $\tilde{p}$-relation is preserved throughout (fragments of) computations of the two models.

Intuition 3. Between the events in M_{CAPSL} and the joint actions in M_{IS}, we introduce two relations called $\tilde{v}$ and $\tilde{v}'$. These will be secondary and mainly used in order to show more easily how the relations between states evolve along corresponding computations in the two models.

Using the main relations above and other helping notions, we will then be able to show that the relation $\tilde{p} \subseteq S_{M_{CAPSL}} \times G_{M_{IS}}$ induces a homomorphism between the two models. This homomorphism will be then be exploited to prove the preservation of satisfaction/refutation of formulations of atomic goals between the two models.

A Brief Guidance Towards the Proof of Homomorphism between M_{CAPSL} and M_{IS}. The steps that we are going to take are as follows.

1. We will show that the initial states in M_{CAPSL} and those induced by the algorithm *tr* in M_{IS} are in the $\tilde{p}$-relation (Definition 4.3.1, Definition 4.3.2 and Remark 4.3.3).

2. We will prove that M_{CAPSL} send events and specific fragments of M_{IS} computations preserve the $\tilde{p}$-relation between states (Lemma 4.3.8). To achieve these, in Definitions 4.3.6 and 4.3.7 we formalise the concept of linear series Dolev-Yao analysis actions in M_{IS}.

3. In Definitions 4.3.10 and 4.3.11, we formalise the concept of (linear) intercept-analysis-synthesis series of actions in M_{IS}. Then, we prove that send events in M_{CAPSL} and maximal series of intercept-analysis-synthesis in M_{IS} still preserve the $\tilde{p}$-relation between states (Lemma 4.3.16).

4. We prove that receive events in M_{CAPSL} and receive actions in M_{IS} preserve the $\tilde{p}'$-relation and the $\tilde{p}$-relation, i.e., preserve the (un-)ability of the intruder to analyse and/or synthesise terms in the two systems (Lemma 4.3.21).

Eventually, in Theorem 4.4.9 we use the aforementioned lemmas to prove that the $\tilde{p}$-relation induces a homomorphism between the two systems. Figure 4.4 expresses in a schematic way our roadmap to proving the existence of this homomorphism, as explained above.

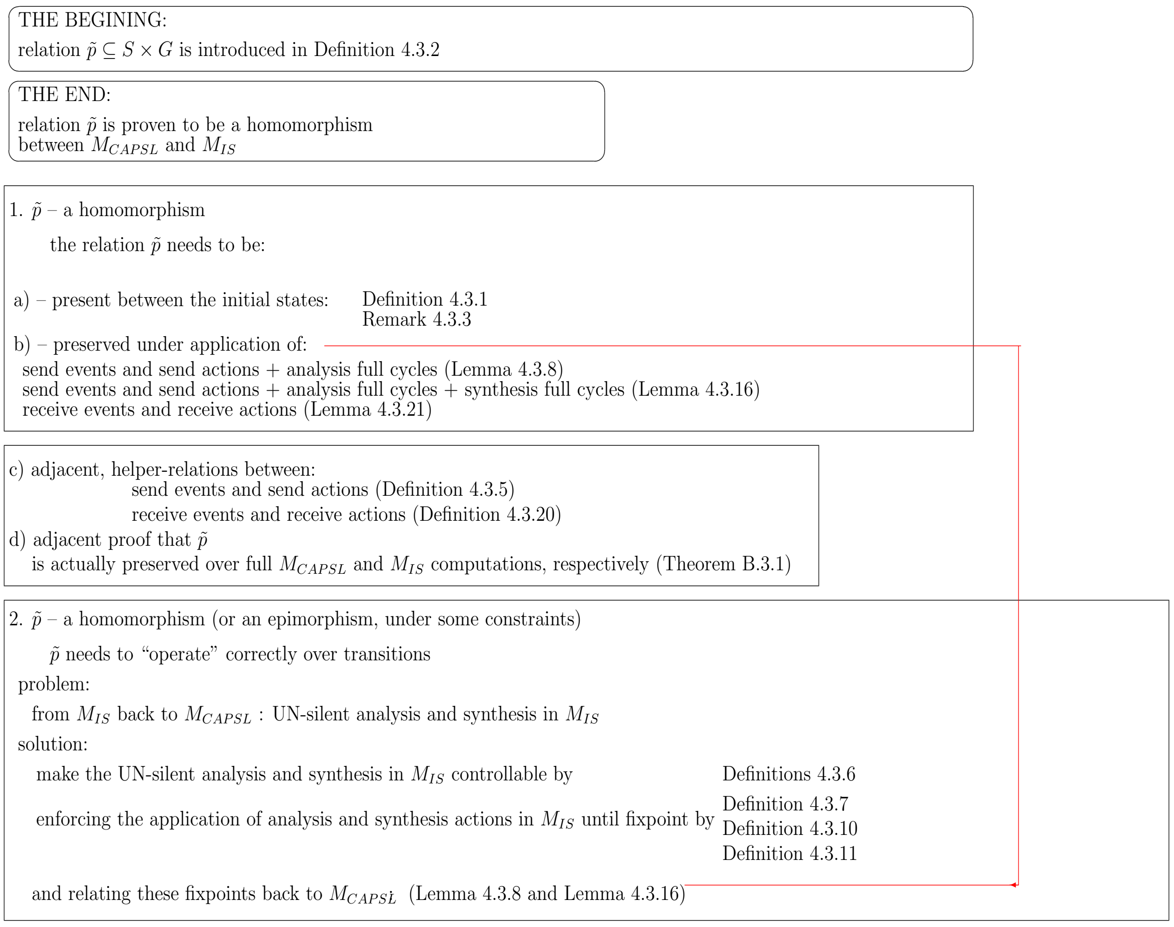

THE BEGINING:
relation $\tilde{p} \subseteq S \times G$ is introduced in Definition 4.3.2
THE END:
relation $\tilde{p}$ is proven to be a homomorphism between M_{CAPSL} and M_{IS}
1. $\tilde{p}$ – a homomorphism
the relation $\tilde{p}$ needs to be:
a) – present between the initial states: Definition 4.3.1 Remark 4.3.3
b) – preserved under application of:
send events and send actions + analysis full cycles (Lemma 4.3.8)
send events and send actions + analysis full cycles + synthesis full cycles (Lemma 4.3.16)
receive events and receive actions (Lemma 4.3.21)
c) adjacent, helper-relations between:
send events and send actions (Definition 4.3.5)
receive events and receive actions (Definition 4.3.20)
d) adjacent proof that $\tilde{p}$ is actually preserved over full M_{CAPSL} and M_{IS} computations, respectively (Theorem B.3.1)
2. $\tilde{p}$ – a homomorphism (or an epimorphism, under some constraints)
$\tilde{p}$ needs to "operate" correctly over transitions
problem:
from M_{IS} back to M_{CAPSL} : UN-silent analysis and synthesis in M_{IS}
solution:
make the UN-silent analysis and synthesis in M_{IS} controllable by
Definitions 4.3.6
enforcing the application of analysis and synthesis actions in M_{IS} until fixpoint by
Definition 4.3.7
Definition 4.3.10
Definition 4.3.11
and relating these fixpoints back to M_{CAPSL} (Lemma 4.3.8 and Lemma 4.3.16)

Figure 4.4 The Roadmap towards the Proof of Homomorphism between Models

4.3.2 Selected Relations Induced By the Algorithm *tr*

Let Pr be an RT protocol, M^{Pr}_{CAPSL} be its bounded protocol model, Σ be the set of role-instantiations given by the set Ev^{Pr} of all (T,k)-*events* in M^{Pr}_{CAPSL}, Υ^{Pr}_{IS} be the MAS-based system resulting from applying the algorithm tr to M^{Pr}_{CAPSL} and the set $\Sigma^{+} = \{\sigma^{+} \,|\, \sigma \in \Sigma \text{ and } \sigma^{+} \cong \sigma\}$ of role-substitutions be as constructed by algorithm tr. Let M^{Pr}_{IS} be the unwound model corresponding to the interpreted system Υ^{Pr}_{IS}.

The algorithm tr induces a relation between the initial states in the two unwound models. We reinstate this relation in Definition 4.3.1. The left-hand side of the Definition 4.3.1 shows the initial states in the M_{CAPSL} model, given at the input of the algorithm tr. The right-hand side of the Definition 4.3.1 shows the corresponding initial states in the M_{IS} model, produced by the output of the algorithm tr.

Definition 4.3.1 (Relation $\tilde{p}_0$ between Initial States in M_{CAPSL} and in M_{IS}) *The relation $\tilde{p}_0 \subseteq S_0 \times G_0$ between initial states in M^{Pr}_{CAPSL} and initial states in M^{Pr}_{IS} is induced by algorithm tr as following:*

M_{CAPSL}, Σ_{Pr}, $\mathcal{A} = Ho \cup \mathbf{I}$	Υ_{IS}, Σ^+_{Pr}, $Ag = \bigcup_{A \in Ho} \bigcup_{\sigma \in \Sigma_{Pr}} \{ag_A \mid \sigma^+(A) \mapsto ag_A,\ \sigma^+ \cong \sigma\} =$
	$= \{ag_1, ag_2, \ldots, ag_n\}$ and the *Env* agent
$s_0 = (s_{0_A} \mid A \in \mathcal{A})$	
$s_{0_A} = \mathcal{A} \cup \mathcal{C} \cup \mathcal{K}_A \cup Secret_A,\ A \in Ho$	
$s_{0_\mathbf{I}} = \mathcal{A} \cup \mathcal{C} \cup \mathcal{K}_\mathbf{I} \cup \{n_\mathbf{I}, k_\mathbf{I}\}$	
$\sigma(A\text{-}role)$, $A \in Ho$,	ag_i, $1 \le i \le n$, $\sigma^+(A\text{-}role) \mapsto_{\Upsilon_{IS}} ag_i$, where:
σ arbitrary with $\sigma(A) \neq \mathbf{I}$	$\sigma^+ \cong \sigma$,
	$ag_i.id = \sigma(A) = \sigma^+(A)$ in $l_{ag_i}@first(Steps^A)$,
	and $\forall B \in Ho, B \neq A, ag_i.PartnerB = \sigma^+(B)$ in $l_{ag_i}@first(Steps^A)$,
	for all $t \in Atoms^A$, $\exists t$ variable, $ag_i.t :: \mathcal{R}_t$,
	for all $t \in OwnedAtoms^A$, $ag_i.t = \sigma^+(t)$ in $l_{ag_i}@first(Steps^A)$,
	for all $t \in LearnedAtoms^A$, $ag_i.t = \sigma^+(t)$ in $l_{ag_i}@first(Steps^A)$,
$\Sigma_1 := \{\sigma_1 \mid \sigma_1(A) = \mathbf{I},\ A \in Ho\}$	
σ_2 with $\sigma_2(n_\mathbf{I}), \sigma_2(k_\mathbf{I})$	$\Sigma^+ = \Sigma^+_1 \cup \sigma^+_2$
	$\forall t \in T_0$, $\exists t$ variable in *Env*, $Env.t :: \mathcal{R}_t$
	$\forall t \in T_0 \quad Env.t = \sigma^+_2(t)$ in $l_{Env}@0$, where $\sigma^+ \in \Sigma^+$
	other data (e.g., $t \in Msg, t' \in Composites$, etc.) unfixed in $l_{Env}@0$

For $s_0 \in S_0$, $g_0 \in G_0$, if $(s_0, g_0) \in \tilde{p}_0$, we write $s_0\, \tilde{p}_0\, g_0$ and say that the state s_0 and the state g_0 are *in the $\tilde{p}_0$-relation.*

We do not explain Definition 4.3.1 further as it simply reiterates how algorithm tr operates on initial states (i.e., this was already explained in Section 4.2).

Several intuitions behind how to choose relations between states were presented in pages 144–146. Now, Definition 4.3.2 formalises a relation which follows all the aforementioned intuitions.

Definition 4.3.2 (Relation $\tilde{p}$ between States in M_{CAPSL} and in M_{IS}) *Let M^{Pr}_{CAPSL} and Υ^{Pr}_{IS} be the models in algorithm tr as above, and let M^{Pr}_{IS} be the unwinding of Υ^{Pr}_{IS}. Let ξ be an arbitrary (T,k)-run in M^{Pr}_{CAPSL} and α be a computation in M^{Pr}_{IS}. Let $s \in S$ be a non-initial state of ξ and $g \in G$ be a non-initial state of α. State $s \in S$ is* in the relation $\tilde{p} \subseteq S \times G$ with $g \in G$, $s\,\tilde{p}\,g$, *if:*

for $A \in Ho$, $\sigma(A\text{-}role) \in \Sigma$ with $\sigma(A) \neq \mathbf{I}$, $\sigma^+ \in \Sigma^+$, $\sigma^+ \cong \sigma$, $\sigma^+(A\text{-}role) \mapsto_{\Upsilon_{IS}} ag$,

for $Gen \subset Generated^A$, for $t \in Atoms^A$ arbitrary:

(1). $[\sigma(t)/t] \in analz(\sigma(s_A \cup Gen)) \Rightarrow [\Sigma^+(t)/t] \in g_{ag}$

for $\sigma_1 \in \Sigma|_{\mathbf{I}}$, $\sigma_1^+ \in \Sigma^+, \sigma_1^+ \cong \sigma$, $\sigma_2^+ \supset \sigma_1^+$, with $\sigma_2^+(\mathbf{I}) \mapsto_{\Upsilon_{IS}} Env$:

(2). $\forall t \in T_0,\ \Sigma(t) \notin analz(\sigma_1(s_I)) \Rightarrow \Sigma^+(t) \notin g_{Env}.values\_log$

for $\sigma_1 \in \Sigma|_{\mathbf{I}}$, $\sigma_1^+ \in \Sigma^+, \sigma_1^+ \cong \sigma$, $\sigma_2^+ \supset \sigma_1^+$, with $\sigma_2^+(\mathbf{I}) \mapsto_{\Upsilon_{IS}} Env$:

(3). $\forall t \in T_0,\ \Sigma(t) \in analz(\sigma_1(s_I)) \Rightarrow \Sigma^+(t) \in g_{Env}.values\_log$

For $s \in S$ and $g \in G$, if $s\,\tilde{p}\,g$ we sometimes say that the state s is in the $\tilde{p}$-relation *with the state g.*

In (1), the notation $[\Sigma^+(t)/t] \in g_{ag}$ means that $[v/t] \in g_{ag}$, for $v \in \mathcal{R}^{ag}_t$ (i.e., for $t \notin$ *OwnedAtoms*^{A}, any value in $\mathcal{R}_t$ under Σ^+ is possible in the non-initial state g_{ag}). In (2), $\Sigma^+(t) \in S$ (i.e., S a set) means that $\sigma_i(t) \in S$, for some $\sigma_i \in \Sigma^+$. Then, Definition 4.3.2 expresses that two states s and g are in $\tilde{p}$-relation if: *a)* in M^{Pr}_{IS}, an agent ag_A of role A under σ^+ "sees" at state g the same atomic terms as the principal A under σ can generate and analyse at state s in M^{Pr}_{CAPSL}; *b)* the intruder "has" an atomic t at g in M^{Pr}_{IS} if and only if it can deduce it at s in M^{Pr}_{CAPSL} (up to some renaming over $\mathcal{R}_t$ or to the extension of Σ_{Pr} to Σ^+_{Pr}).

Recall the preconditions required in M_{CAPSL} for a principal to send a term: i.e., $A!B : (M)t$ under σ can be applied at a state s if $t \in \overline{s_A \cup M}$, where $\overline{s_A \cup M} = synth(analz(s_A \cup M))$ and $M \subseteq Generated^A$ is the set of terms generated by A to compose t. Hence, in Definition 4.3.2 we consider $[x/t] \in analz(\sigma(s_A \cup Gen))$, with $Gen \subseteq Generated^A$. In other words, to the set of terms that principal A can analyse at s, we add those that she could potentially generate at s. This will not hinder the proofs to follow, due to the following reasons: *a)* in M_{IS} all terms in $Generated^A$

are included in the initial states and are fully instantiated; *b)* the assignment of values to variables in local states is persistent in M_{IS}.

Remark 4.3.3 *Note that the initial states are in a $\tilde{p}$-relation. So, the $\tilde{p}_0$-relation on initial states induced by algorithm tr is also a $\tilde{p}$-relation, i.e., $\tilde{p}_0 \subsetneq p$.*

Definitions 4.3.4 and 4.3.5 will now introduce some relations over send events in M_{CAPSL} and actions in M_{IS}. These relations are a wrapper to the $\tilde{p}$-relation between states. In that sense, they ease the presentation of Lemma 4.3.8; this lemma will show that send events and certain, enforced sequences of actions preserve the $\tilde{p}$-relation between states.

Definition 4.3.4 (Relation $\tilde{u}$ between Send Events and Local Send Actions) *Let M^{Pr}_{CAPSL} and Υ^{Pr}_{IS} be the models in algorithm tr, as above. For $A \in Ho$, $\sigma(A\text{-}role) \in \Sigma$, $\sigma(A) \neq \mathbf{I}$, $i \in Steps^A$, $B \in Ho, B \neq A$, $t \in \mathcal{T}$, let $e = (\omega|_A, \sigma, i)$ be a send event in M^{Pr}_{CAPSL} with $act(e) = A!B : (M)t$. Let e be enabled at a state $s \in S$. Let $a = send(t, x)$ be an action of agent ag_A, where $\sigma^+(A\text{-}role) {\mapsto}_{\Upsilon_{IS}} ag_A$, $\sigma^+ \in \Sigma^+$, $\sigma^+ \cong \sigma$ and $x \in \mathcal{R}^{ag_A}_t$. The send event e is* in the relation $\tilde{u} \subseteq Ev^{Pr} \times LAct$ *with the local action $a \in Act_{ag_A}$, $e\,\tilde{u}\,a$, if: there exists $g \in G_{M_{IS}}$ such that $construct^{\mathbb{I}^{\sigma^+}}(t, g_{ag_A}@i, view^0_{ag_A}) = (t, x)$ or, equivalently, such that $g_{ag_A}.step = i$ and $a \in P_{ag_A}(g_{ag_A})$, where $view^0_{ag_A}$ is the initial view of ag_A.*

For $e \in Ev^{Pr}$ and $a \in LAct$, if $e\,\tilde{u}\,a$ we say that the event e is in the $\tilde{u}$-relation *with the local action a.*

Definition 4.3.4 formalises the sending actions that algorithm *tr* creates in Υ_{IS} starting from send events in M_{CAPSL}. Hence, Definition 4.3.4 expresses that a send event of form $A!B : (M)t$ under σ is in the $\tilde{u}$-relation with a local action $send(t, x)$ of ag_A if the following hold: *a)* principal A of M_{CAPSL} is mapped in Υ_{IS} to the agent ag_A under the extension σ^+ of substitution σ; *b)* in M_{CAPSL} the value $\sigma(t)$ is sent out, whereas in Υ_{IS} the value x is sent for t equals $\sigma^+(t)$; *c)* the event and the local action are possible in M_{CAPSL} and Υ_{IS}, respectively. In other words, Definition 4.3.4 simply expresses that the event and the local action are possible in both systems, are in the scope of corresponding parties in the two models and the terms involved are identical up to renaming of term-values, i.e., up to $\sigma \cong \sigma^+$.

In Definition 4.3.5 we lift the $\tilde{u}$-relation to the operational level (i.e., from Υ_{IS} to M_{IS}). At the same time, we take it beyond what is immediately implied by algorithm tr.

Definition 4.3.5 (Relation $\tilde{v}$ between Send Events and Joint Actions) *Let M^{Pr}_{CAPSL} and Υ^{Pr}_{IS} be the models in algorithm tr, as above and let M^{Pr}_{IS} be the unwinding of Υ^{Pr}_{IS}. For $A \in Ho$, σ(A-role) $\in \Sigma$, $\sigma(A) \neq \mathbf{I}$, $i \in Steps^A$, $B \in Ho$, $B \neq A$, $t \in \mathcal{T}$, let $e = (\omega|_A, \sigma, i)$ be a send event in M^{Pr}_{CAPSL} with $act(e) = A!B : (M)t$. Let e be enabled at a state $s \in S$. Let $a_1 = send(t,x)$, $a_1 \in Act_{ag_A}$, where $\sigma^+(A\text{-}role) \mapsto_{\Upsilon_{IS}} ag_A$, $\sigma^+ \cong \sigma$ and $x \in \mathcal{R}^{ag_A}_t$. Let $a \in Act$ be a joint action in Υ^{Pr}_{IS} such that $\overline{a}_{ag_A} = a_1$ and $\overline{a}_{Env} = intercept\_from(ag_A)$. The event e is* in the relation $\tilde{v} \subseteq Ev^{Pr} \times Act$ *with the joint action $\overline{a}$, $e\,\tilde{v}\,\overline{a}$, if:*

$$\begin{cases} e\,\tilde{u}\,a_1,\ \text{with } g \in G \text{ and } a_1 \in P_{ag_A}(g_{ag_A})\ (1) \\ (\,s[e\rangle s' \text{ and } s\,\tilde{p}\,g\,) \text{ implies } E(g,\overline{a}) = g',\ \text{where } g' \in G \text{ and } s' \in S\ (2) \end{cases}$$

For $e \in Ev^{Pr}$ and $a \in Act$, if $e\,\tilde{v}\,a$ we say that the event e is in $\tilde{v}$-relation *with the global action a.*

Definition 4.3.5 expresses the following. A send event e of an *A-role* under σ in M^{Pr}_{CAPSL} is in the $\tilde{v}$-relation with a joint action of M^{Pr}_{CAPSL} in which an ag_A agent of the *A-role* under σ^+ does the mapping send, if the event/action are respectively triggered at states which are in $\tilde{p}$-relation. In other words, not only are the event and the action possible at points of the systems which are "similar" from the point of view of the intruder (as per relation $\tilde{u}$), but they are effectively applied at states that are equivalent with respect to the possession of values for atomic terms.

Recall that in M_{IS} each joint action which is applied at a state $g \in G$ and contains an $intercept\_from(ag)$ is necessarily followed by a linear series of joint actions with the Environment component being $analz(t,x)$, for $ag \in Ag$, for some $t \in Composites$ and $x \in \mathcal{R}_t$. Definitions 4.3.6 and 4.3.7 will now formalise the latter. This will be useful in Lemma 4.3.8 to prove that send events in M_{CAPSL} and certain sequences of joint actions in M_{IS} are such that they preserve the $\tilde{p}$-relation between states.

Definition 4.3.6 (The $analz$-Sequences of Dolev-Yao Actions in M_{IS}) *Let Υ^{Pr}_{IS} be the IS-based model for an RT protocol Pr obtained via algorithm tr from M^{Pr}_{CAPSL}. Let M^{Pr}_{IS} be the unwinding of Υ^{Pr}_{IS}. Let $A \in Ho$, σ(A-role) $\in \Sigma_{Pr}$, $\sigma^+ \cong \sigma$ and $\sigma^+(A\text{-}role) \mapsto_{\Upsilon_{IS}} ag_A$. Let $t \in Msg$, $x \in \mathcal{R}_t$, $g, g_1 \in G$ and let $\overline{a}$ be a joint action such that $\overline{a}_{ag_A}$=send(t,x), $\overline{a}_{Env}$=$intercept\_from(ag_A)$ and $E(g,\overline{a}) = g_1$. Let $k \geq 1$ be a constant, $1 \leq n \leq k \times \sharp max\_SubstComposite$, $1 \leq i \leq n$,*

$g_i \in G$, $t' \in$ Composites and $x' \in \mathcal{R}_{t'}$. Then, let $\overline{a}_1 \ldots \overline{a}_n$ be a sequence of joint actions with $\overline{a}_{i_{Env}} = analz(t', x')$ and $E(g_i, \overline{a}_i) = g_{i+1}$. We call $\overline{a}_1 \ldots \overline{a}_n$ a g_1-$analz^{\mathbb{I}}$ sequence of Dolev-Yao actions *or an $analz$*-sequence $\overline{a}_1 \ldots \overline{a}_n$ of Dolev-Yao actions enabled after $\overline{a}$.

Definition 4.3.6 expresses the denotation of $(analz^{\mathbb{I}})^n(g_{1_{Env}})$ (i.e., starting the application of $analz_{M_{IS}}$ at $g_{1_{Env}}$ and repeating it n times). But, from Chapter 3 and Appendix A, also recall that such a Dolev-Yao analysis series is linear and it ends when the closure of a state under Dolev-Yao analysis is attained. Therefore, Definition 4.3.7 will formalise this precisely.

Definition 4.3.7 (Complete $analz$-Sequences of Dolev-Yao actions in M_{IS}) *Let Pr be an RTP and Υ_{IS}^{Pr} be obtained from M_{CAPSL}^{Pr} via algorithm tr. Let M_{IS}^{Pr} be the unwinding of Υ_{IS}^{Pr}. Let $g_0 \in G_0$, $g_1 \in G$ and α be some possible computation in M_{IS}^{Pr} such that $g_0 \alpha g_1$. Let $n \geq 1$ and $\overline{a}_1 \ldots \overline{a}_n$ be a g_1-$analz^{\mathbb{I}}$ sequence of Dolev-Yao actions in M_{IS}^{Pr}. If $\overline{a}_1 \ldots \overline{a}_n$ generates the closure of $analz^{\mathbb{I}}$ at g_1, then the sequence $\beta_{Analz} = \overline{a}_1 \ldots \overline{a}_n$ in M_{IS}^{Pr} is called a g_1-analz* complete sequence of Dolev-Yao actions.

Definition 4.3.7 introduces a particular type of g_1-$analz^{\mathbb{I}}$ sequence of Dolev-Yao actions, namely those that generate the closure of $analz^{\mathbb{I}}$ at g_1, for some $g_1 \in G$. Let $g_2 \in G$ such that g_2=$analz^{\mathbf{I}^n}(g_{1_{Env}})$. In terms of the semantics in Chapter 3, Definition 4.3.7 expresses the following. The sequence $\overline{a}_1 \ldots \overline{a}_n$ of actions is a g_1-$analz$ complete sequence of Dolev-Yao actions if, for all $t \in Composites$, for all values $x \in \mathcal{R}_t$, it is the case that $analz^{\mathbf{I}}(analz^{\mathbf{I}^n}(g_{2_{Env}}.analz\_log, g_{2_{Env}}.values\_log, t, x)$=$false$.

We reiterate that, in Lemma A.1.1, we prove that such a maximum number n (for the closure of $analz^{\mathbb{I}}$ to be acquired at g_1 as above) always exists.

We previously suggested that the $\tilde{p}$-relation between states is preserved under the respective application of send events and sending joint actions followed by full cycle of Dolev-Yao analysis. In Appendix B, we informally explain the ideas behind the proof of Lemma 4.3.8 which formalises this fact. In the following, we give and prove this lemma formally.

Lemma 4.3.8 *Let Pr be an RT protocol, M_{CAPSL}^{Pr} be its bounded protocol model and Υ_{IS}^{Pr} be obtained from M_{CAPSL}^{Pr} via algorithm tr. Let M_{IS}^{Pr} be the unwinding of Υ_{IS}^{Pr}. For σ(A-role) $\in \Sigma_{Pr}$, $\sigma(A) \neq \mathbf{I}$, $t \in \mathcal{T}$, let $e = (\omega|_A, \sigma, i)$ with act(e) $= A!B : (M)t$ and let $s, s' \in S$ such that $s[e\rangle s'$. Let*

$\sigma^+(A\text{-}role) \mapsto_{\Upsilon_{IS}} ag_A$, $\sigma^+ \cong \sigma$, $g, g' \in G$ and $g\overline{a'}g'$ be the application of a joint action $\overline{a'} \in Act$ at g in M_{IS}^{Pr} such that $e\,\tilde{v}\,\overline{a'}$. Let $g'' \in G$ and $\beta_{Analz}{=}\overline{a}_1 \ldots \overline{a}_n$ be a g'-analz complete sequence of Dolev-Yao actions in M_{IS}^{Pr} with $g'\beta_{Analz}g''$. Then, $s'\,\tilde{p}\,g''$.

Proof In the hypotheses of this lemma $s\,\tilde{p}\,g$ and the send event e and the local action $\overline{a'}_{ag_A}$ are identical up to $\sigma \cong \sigma^+$, for $s, \in S$, $g \in G$, $e \in Ev^{Pr}$ and $\overline{a'} \in Act$. The event e is applied at s and the action $\overline{a'}$ is applied at g. Formally, in the hypotheses, it is the case that $e\,\tilde{v}\,\overline{a'}$, $s[e\rangle s'$, $E(g, \overline{a'}) = g'$. Then, a linear, complete sequence β_{Analz} of Dolev-Yao analysis actions is applied at g', producing g''. In these circumstances, we need to prove that $s'\,\tilde{p}\,g''$.
Recall that the definition of the $\tilde{p}$-relation has three parts denoted (1), (2) and (3).

For (1), we have to prove that if $[\sigma(t')/t'] \in analz(\sigma(s'_A \cup Gen))$, then $[\Sigma^+(t')/t'] \in g''_{ag_A})$, for all terms $t' \in Atoms^A$, for some $Gen \subset Generated^A$ arbitrarily fixed.

Let us then assume that for all $t' \in Atoms^A$, for some $Gen \subset Generated^A$ arbitrarily fixed, it is the case that $[\sigma(t')/t'] \in analz(\sigma(s'_A \cup Gen))$. (**hyp**)

Since $act(e)$ is $A!B : (M)t$ and given the semantics of M_{CAPSL}, the state $s' \in S$ described in the hypotheses of this lemma is: $s'_A{=}s_A \cup M \cup \{t\}$. (**i**)

Given the semantics of M_{CAPSL} (i.e., $M \subseteq (T_0 \cap Sub(t))$) and given (**i**), it follows that $analz(s'_A) = analz(s_A \cup M)$ (**ii**)

By (**hyp**) and (**ii**), it is the case that $[\sigma(t')/t'] \in analz(\sigma(s_A \cup Gen))$. And, in the given hypotheses, $e\,\tilde{v}\,\overline{a'}$ and $E(g, \overline{a'})$. So, by definition of the relation $\tilde{v}$, it follows that $s\,\tilde{p}\,g$. Then, by point (1) in the definition of the p-relation, it follows for all $t' \in Atoms^A$, $[\Sigma^+(t')/t'] \in g_{ag_A}$. (**iii**)

Following the extension of substitutions in M_{CAPSL} to role-substitutions in M_{IS} (i.e., all atoms in $Atoms^A$ are set to values by σ^+), it is the case that for any $M \subseteq Generated^A$, for any $m \in M$ as above, $[\sigma^+(m)/m] \in g_{ag_A}$. (**iv**)

Given the semantics of M_{IS} (i.e., no updates take place to g_{ag_A} during the β_{Analz} sequence), the states g' and g'' in the hypotheses are such that $g''_{ag_A}{=}g'_{ag_A}{=}g_{ag_A}[next(Step^A, i)/_{step}]$ (**v**), where $i \in Steps^A$ is the step of the event e and $g_{ag_A}[next(Step^A, i)/_{step}]$ denotes that the value of variable $step$ in g_{ag_A} is updated to $next(Step^A, i)$.

From (**hyp**), (**v**) and (**iii**), it follows that if $[\sigma(t')/t'] \in analz(\sigma(s'_A \cup Gen))$, then $[\Sigma^+(t')/t'] \in g''_{ag_A}$, for all $t' \in Atoms^A$, $Gen \subseteq Generated^A$.

So, part (1) of s' being in the $\tilde{p}$-relation with g'' follows.

To prove part (2) of the definition of the $\tilde{p}$-relation, we need to show that if $\Sigma(t) \notin analz(\sigma_1(s'_I))$, then $\Sigma^+(t) \notin g''_{Env}.values\_log$, for every $t \in T_0$. Assume that $\Sigma(t_0) \notin analz(\sigma_1(s'_I))$, for an arbitrary $t_0 \in T_0$ and σ_1 applied to the state of $\mathbf{I}$ in M_{CAPSL}. **(hyp2)**

Consider the semantics of M_{CAPSL} and the fact that, in the hypotheses given, the event e is such that $act(e) = A!B : (M)t$. Then, it is the case that $s'_I{=}s_I \cup \{t\}$. So, **(hyp2)** becomes:
for $t_0 \in T_0$, $\Sigma(t_0) \notin analz(\sigma_1(s_I) \cup \Sigma(t))$. **(vi)**

Since $s\,\tilde{p}\,g$, then for an arbitrary $t_0 \in T_0$ with case that $\Sigma(t_0) \notin analz(\sigma_1(s_I) \cup \Sigma(t))$ in **(vi)**, it is the case that $\Sigma^+(t_0) \notin g_{Env}.values\_log$. **(vii)**

Given the semantics of M_{IS}, for $g, g^{'}, g^{''}$ in the given hypotheses it is the case that $g'_{Env}.analz\_log = g_{Env}.analz\_log \cup (t, \Sigma^+(t))$ and $(analz^{\mathbf{I}})^n(g^{'}_{Env}) = g^{''}_{Env}$, $(analz^{\mathbf{I}})(g^{''}_{Env}) = g^{''}_{Env}$. **(viii)**

By the definition of *analz* in M_{CAPSL} and M_{IS} (i.e., both following the standard semantics in [163]), **(vii)**, **(viii)**, it follows that $\Sigma^+(t_0) \notin g^{''}_{Env}.values\_log$, for $t_0 \in \mathcal{T}_0$ arbitrarily chosen above. In other words, if $analz_{M_{CAPSL}}$ does not produce t_0 out of $s_{\mathbf{I}} \cup \{t\}$, nor does the closure of $g_{Env}.analz\_log \cup (t, \Sigma^+(t))$ under $analz_{M_{IS}}$ contain t_0, when $s\,\tilde{p}\,g$. (This is so because $analz_{M_{CAPSL}}$ and the closure under $analz_{M_{IS}}$ both follow the same standard semantics [163] and they are applied at states that are identical with respect to the analysis of atomic terms).
Hence, part (2) of s' being in the $\tilde{p}$-relation with $g^{''}$ is also proven.

Part (3) of s' being in the $\tilde{p}$-relation with $g^{''}$ is proven similarly to the way we proved part (2) above. ▮

Lemma 4.3.8 ensures the following: if a send event/sending joint action are respectively applied at two states in the $\tilde{p}$-relation and afterwards, at the resulting state in M_{IS}, an *analz*-complete sequence of Dolev-Yao actions takes effect, then the final respective states are in the $\tilde{p}$-relation. This is schematically described by Figure 4.5.

Definitions 4.3.6 and 4.3.7 formalised the complete *analz* sequence of Dolev-Yao actions. To follow, Definition 4.3.10 and 4.3.11 will formalise the series of forge actions and their closure upon a state of the Environment in M_{IS}.

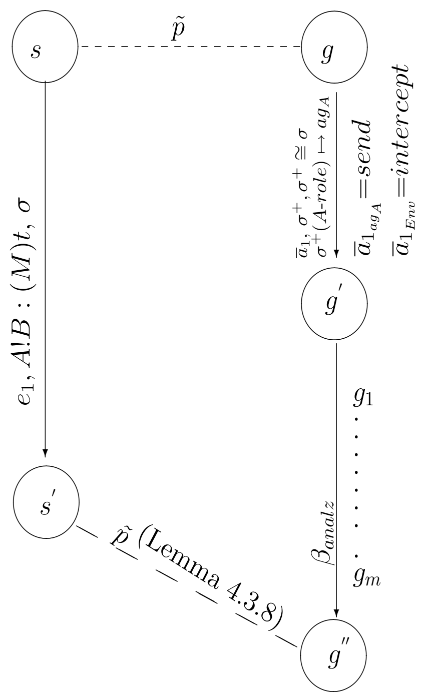

$\tilde{p}$
s
g
$e_1, A!B : (M)t, \sigma$
$\overline{a}_1, \sigma^+, \sigma^+ \cong \sigma$
$\sigma^+(A\text{-}role) \mapsto ag_A$
$\overline{a}_{1_{ag_A}} = send$
$\overline{a}_{1_{Env}} = intercept$
g'
g_1
β_{analz}
g_m
s'
$\tilde{p}$ (Lemma 4.3.8)
g''

Figure 4.5 Proven Preservation of the $\tilde{p}$-relation along Fragments of Computations

Remark 4.3.9 *We reiterate that there are no actions of type analz applied during the sequence of forging actions. This will be formalised in Definition 4.3.10 and 4.3.11 respectively.*

Upon the Υ_{IS} synchronisation, the local states of the honest agents are not updated when a local action of the *Environment* of type *forge* triggers. Thus, in the following, we abstract away the honest components of the joint actions where the Environment is forging messages. The same abstraction was in place for *analz*-sequences of Dolev-Yao actions.

Recall that in M_{IS} an *intercept* Dolev-Yao action is followed by a complete *analz* sequence of Dolev-Yao actions and any (such) complete *analz* sequence of Dolev-Yao actions is followed, at its turn, by a finite sequence of joint actions $\overline{a}_i$ of form $\overline{a}_{i_{Env}} = forge(t', x')$, where $0 \leq i \leq \sharp maxSubstMsg$ (see Section 3.3.2.9 and Figure 3.1). The Definition 4.3.10 and 4.3.11 formalise this precisely.

Definition 4.3.10 (Intercept-analz-forge Series of Dolev-Yao Actions in Υ_{IS}) *Let Pr be an RT protocol and Υ_{IS}^{Pr} be obtained by algorithm tr. Let M_{IS}^{Pr} be the unwinding of Υ_{IS}^{Pr}. Let $\overline{a}_0$ be a joint* *action of the form $\overline{a}_{0_{ag_A}} = send(t,x)$ and $\overline{a}_{0_{Env}} = intercept\_from(ag_A)$. Let $g, g' \in G$ such that $E(g, \overline{a_0}) = g'$. Let β_{Analz} be a g'-analz complete sequence of Dolev-Yao actions and $g_1 \in G$ such that $g'\beta_{Analz}g_1$. Let $1 \leq n \leq \sharp maxSubstMsg$, $1 \leq i \leq n$, $g_i \in G$, $t' \in Composites$ and $x' \in \mathcal{R}_{t'}$ such that $\overline{a_1}, \ldots, \overline{a_n}$ is the finite series of joint actions with $E(g_1, \overline{a}_1)$=g_2, $E(g_2, \overline{a}_2) = g_3, \ldots, E(g_n, \overline{a}_n)$=$g_{n+1}$, where each $\overline{a_i}$ is of the form $\overline{a}_{i_{Env}} = forge(t', x')$. The sequence $\overline{a}_0\beta_{Analz}\overline{a}_1 \ldots \overline{a}_n$ is called an* intercept-analz-forge sequence of Dolev-Yao actions applied at *g. We write $g\overline{a}_0g'\beta_{Analz}g_1\overline{a}_1g_2 \ldots \overline{a}_ng_{n+1}$ or simply, $g\overline{a}_0\beta_{Analz}\overline{a}_1 \ldots \overline{a}_ng_{n+1}$.*

The sequence $\beta_{Analz}\overline{a}_1 \ldots \overline{a}_n$ is called an analz-forge sequence of Dolev-Yao actions applied at *g', i.e., when the intercepting action $\overline{a}_0$ is disregarded. We write $g'\beta_{Analz}g_1\overline{a}_1g_2 \ldots \overline{a}_ng_{n+1}$ or, simply, $g\beta_{Analz}\overline{a}_1 \ldots \overline{a}_ng_{n=1}$.*

Definition 4.3.10 formalises that in M_{IS} a send-intercept action is followed by a complete series of Dolev-Yao analysis actions and a sequence of Dolev-Yao synthesis actions. As we explained in Chapter 3, this sequence of Dolev-Yao synthesis actions is linear. Also, it will end when there are no more messages that the intruder can synthesise at a reached state. Definition 4.3.11 will stipulate the latter.

Definition 4.3.11 (Complete Intercept-Analz-Forge Series of Dolev-Yao actions in Υ_{IS}) *Let Pr be an RT protocol and Υ_{IS}^{Pr} be obtained by algorithm tr. Let M_{IS}^{Pr} be the unwinding of Υ_{IS}^{Pr}. Let $g, g' \in G$ and $\overline{a}_0$ be a joint action of form $\overline{a}_{0_{ag_A}} = send(t,x)$, $\overline{a}_{0_{Env}} = intercept\_from(ag_A)$ such that $E(g, \overline{a_0}) = g'$. Let β_{Analz} be a g'-analz complete sequence of Dolev-Yao actions and $g_1 \in G$ such that $g'\beta_{Analz}g_1$. Let $1 \leq n \leq \sharp maxSubstMsg$, $1 \leq i \leq n$, $g_i \in G$, $t' \in Composites$ and $x' \in \mathcal{R}_{t'}$ such that the sequence $\overline{a}_0\beta_{Analz}\overline{a}_1 \ldots \overline{a}_n$ is an intercept-analz-forge sequence of Dolev-Yao actions at g and $E(g_1, \overline{a}_1) = g_2, E(g_2, \overline{a}_2) = g_3, \ldots, E(g_n, \overline{a}_n) = g_{n+1}$. If $n = \sharp maxSubstMsg$, the sequence $\gamma_{Forge} = \overline{a}_0\beta_{Analz}\overline{a}_1 \ldots \overline{a}_{\sharp maxSubstMsg}$ is called a* complete intercept-analz-forge sequence of Dolev-Yao actions applied at *g. We write $g\overline{a}_0g'\beta_{Analz}g_1\gamma_{Forge}g_{n+1}$ or simply, $g\overline{a}_0\beta_{Analz}\gamma_{Forge}g_{n+1}$.*

If $n = \sharp maxSubstMsg$, the sequence $\beta_{Analz}\overline{a}_1 \ldots \overline{a}_{\sharp maxSubstMsg}$ is called a complete analz-forge sequence of Dolev-Yao actions applied at *g'. We write $g'\beta_{Analz}g_1\gamma_{Forge}g_{n+1}$ or simply, $g'\beta_{Analz}\gamma_{Forge}g_{n+1}$.*

Definition 4.3.11 expresses that a complete intercept-analz-forge series of Dolev-Yao actions is a special case of an intercept-analz-forge sequence of Dolev-Yao actions, where the forging sequence is maximal with respect to what the intruder can synthesise at that protocol stage. Definitions 4.3.10 and 4.3.11 introduce analz-forge sequences as sequences of the kind above in which the first joint action, where the intercepting is performed, is not included.

To sum up, Definitions 4.3.2–4.3.7, Definitions 4.3.10 and 4.3.11 and Lemma 4.3.8 have formally specified the intuitions behind the $\tilde{p}$-relation presented at page 144.

We will now correlate send events in M_{CAPSL} also with the Dolev-Yao synthesis actions in M_{IS}. Definition 4.3.12 will formalise that send event in M_{CAPSL} be precisely assimilated to a complete intercept-analz-forge sequence in Υ_{IS}.

Let the set Act^+ denote sequences of joint actions in M_{IS}, i.e., if $g_1, \dots, g_{n+1} \in G$, $a_1, \dots, a_n \in Act$ such that $E(g_1, a_1) = g_2, E(g_2, a_2) = g_3, \dots, E(g_n, a_n) = g_{n+1}$, then the sequence $a_1 \dots a_n$ is an element of the set Act^+.

Definition 4.3.12 (The Relation $\tilde{v}^+$ between Send Events and Series of Actions)
Let M^{Pr}_{CAPSL} and Υ^{Pr}_{IS} be the models in algorithm tr, as above. Let M^{Pr}_{IS} be the unwinding of Υ^{Pr}_{IS}. For $A \in Ho$, $\sigma(A\text{-}role) \in \Sigma$, $\sigma(A) \neq \mathbf{I}$, $i \in Steps^A$, $B \in Ho$, $B \neq A$, $t \in \mathcal{T}$, let $e = (\omega|_A, \sigma, i)$ be a send event in M^{Pr}_{CAPSL} with $act(e) = A!B : (M)t$. Let e be enabled at a state $s \in S$. Let $a_0 \in Act$ be a joint action in Υ^{Pr}_{IS} such that $\overline{a_0}_{ag_A} = send(t, x)$ and $\overline{a_0}_{Env} = intercept\_from(ag_A)$, where $\sigma^+(A\text{-}role) \mapsto_{\Upsilon_{IS}} ag_A$, $\sigma^+ \cong \sigma$ and $x \in \mathcal{R}^{ag_A}_t$. Let β_{Analz} be an analz complete sequence of Dolev-Yao actions and β_{Analz} enabled after $\overline{a}_0$. Let $1 \leq n \leq \sharp maxSubstMsg$ and $\overline{a}_0\beta_{Analz}\overline{a}_1 \dots \overline{a}_n$ be an intercept-analz-forge series of Dolev-Yao actions. The event e is in the relation $\tilde{v}^+ \subseteq Ev^{Pr} \times Act^+$ *with the intercept-analz-forge series of Dolev-Yao actions $\overline{a}_0\beta_{Analz}\overline{a}_1 \dots \overline{a}_n$, $e\,\tilde{v}^+\,(\overline{a}_0\beta_{Analz}\overline{a}_1 \dots \overline{a}_n)$*

$$\textit{if:} \begin{cases} e\,\tilde{v}\,a_0 \text{ with } g \in G_{M_{IS}},\ \overline{a_0}_{ag_A} \in P_{ag_A}(g_{ag_A}) & (1) \\ (s[e\rangle s_1) \text{ implies } (E(g, \overline{a}_0) = g' \text{ and } g\overline{a}_0 g' \beta_{Analz}\overline{a}_1 \dots \overline{a}_n g_{n+1}), & (2) \end{cases}$$

with $s_1 \in S$, $g, g,' g_{n+1} \in G$, $\overline{a}_0\beta_{Analz}\overline{a}_1 \dots \overline{a}_n$ an intercept-analz- forge series of Dolev-Yao actions, β_{Analz} the analz complete sequence enabled after $\overline{a_0}$ and $\overline{a_1} \dots \overline{a_n}$ a (maximal) forge sequence.

Let $e \in Ev^{Pr}$ be a send event and $\overline{a}_0\beta_{Analz}\overline{a}_1 \dots \overline{a}_n \in Act^+$ be an intercept-analz-forge sequence of Dolev-Yao actions. If $e\,\tilde{v}^+\,\overline{a}_0\beta_{Analz}\overline{a}_1 \dots \overline{a}_n$, then we say that the event e is in $\tilde{v}^+$-relation *with the sequence $\overline{a}_0\beta_{Analz}\overline{a}_1 \dots \overline{a}_n$.*

Definition 4.3.12 expresses that a send event e in M_{CAPSL} is in $\tilde{v}^+$-relation with an intercept-analz-forge series of Dolev-Yao actions if it is in $\tilde{v}$-relation with the comprised send-intercept joint action. From Definition 4.3.12, it follows that a send event e in M_{CAPSL} can also be in $\tilde{v}^{+}$-relation with a *complete* intercept-analz-forge series of Dolev-Yao actions $\overline{a}_0\beta_{Analz}\overline{a}_1\ldots\overline{a}_{\sharp maxSubstMsg}$.

In Remark 4.3.9 and in the intuitions at pages 144–146, we have mentioned that in M_{IS} no Dolev-Yao analysis happens during forging actions. This will now be formalised through Definition 4.3.13 and Remark 4.3.14.

Definition 4.3.13 (Relation $=_{analz}$ over States in M_{IS}) *Let $g, g^{'} \in G$. We say that the state g is* in the relation $=_{analz}\subseteq G\times G$ *with the state $g^{'}$, $g =_{analz} g^{'}$, if:*

$$\begin{cases} for\, t \in Atoms^{A}, [x/t_0] \in g_{ag_A} \Leftrightarrow [x/t_0] \in g^{'}_{ag_A} & (1) \\ for\, t \in T_0, [x/t] \in g_{Env}.values\_log \Leftrightarrow [x/t_0] \in g^{'}_{Env}.values\_log & (2) \end{cases}$$

The relation $=_{analz}\subseteq G\times G$ over the global states in M_{IS} is symmetric, reflexive, transitive, hence it is an equivalence relation. For $g\in G$, the notation $[g]_{analz}$ denotes the equivalence class of the state g induced by the relation $=_{analz}$. The Remark 4.3.14 motivates the introduction of this relation, i.e., to express that along $forge$-based fragments of M_{IS} computations, states stay the same from the point of view of Dolev-Yao analysis.

Remark 4.3.14 *Let $1 \leq n \leq \sharp maxSubstMsg$, $g, g', g_{n+1} \in G$ and $ga_0g'\beta_{Analz}g_1a_1\ldots a_ng_{n+1}$ be a computation-fragment in M_{IS} such that $a_0\beta_{Analz}a_1\ldots a_n$ is a (complete) intercept-analz-forge series of Dolev-Yao actions, with β_{Analz} a $g^{'}$-analz complete sequence of Dolev-Yao actions. Then, from Remark 4.3.9 (that no Dolev-Yao analysis or updates of honest agents' local states happen during the forge sequence), it follows that $[g_1]_{analz} \supseteq \{g_1, g_2, \ldots, g_{n+1}\}$.*

We have formalised that along $forge$-based fragments of M_{IS} computations states stay the same from the point of view of Dolev-Yao analysis. Therefore, we can lift the $\tilde{p}$-relation from a relation between states in the two models to a relation between states in M_{CAPSL} and sets of states in M_{IS}. Definition 4.3.15 will express this precisely. In this fashion, Definition 4.3.15 will formalise the first intuition behind the the $\tilde{p}$-relation (see page 144) and the ideas expressed in Figure 4.3.

Definition 4.3.15 (Relation $\tilde{p}^{+}$ over S_{CAPSL} and $2^{G_{IS}}$) *For $s\in S$, $g \in G$ and $[g]_{analz}$, we say that the state s is* in the relation $\tilde{p}^{+}\subseteq S\times 2^{G}$ with the set $[g]_{analz}$ of states, *$s\,\tilde{p}^{+}\,[g]_{analz}$, if $s\,\tilde{p}\,g$.*

According to Definition 4.3.15, an M_{CAPSL} state s in the relation $\tilde{p}^+$ with a class of equivalence modulo $=_{analz}$ if it in the $\tilde{p}$-relation with a representative of the class.

Now, assume that $s \in S$ is in the $\tilde{p}$-relation with some state $g \in G_{M_{IS}}$ and that at g a series of Dolev-Yao forging starts, yielding the states $g'_1, \ldots, g'_m$. Then, by Definition 4.3.15, it is the case that $s\,\tilde{p}^+\,\{g'_1, \ldots, g'_m\}$. The following lemma proves this.

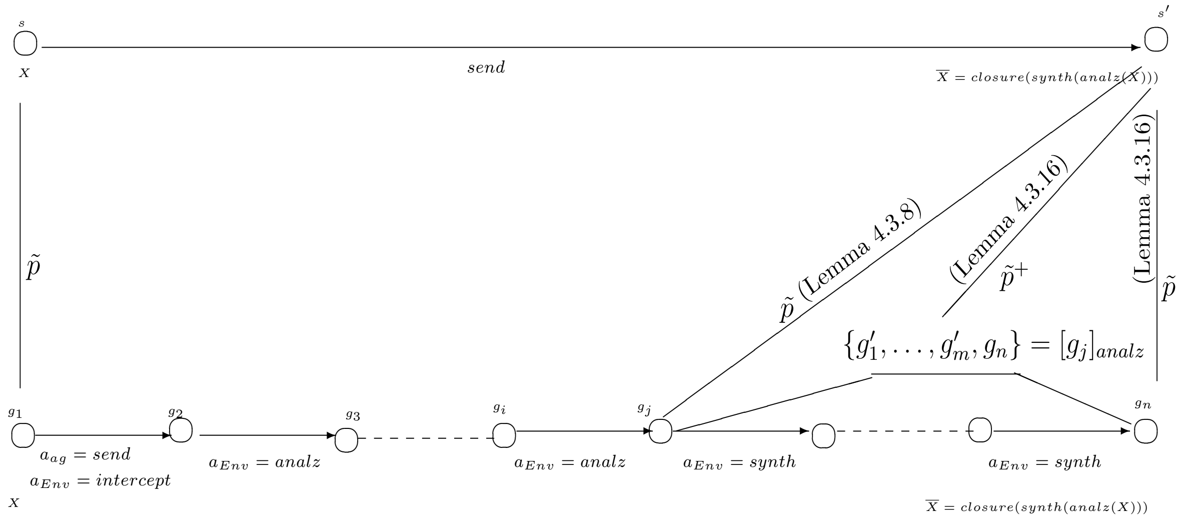
Lemma 4.3.16 *Let Pr be an RT protocol in M^{Pr}_{CAPSL} and Υ^{Pr}_{IS} be obtained from M^{Pr}_{CAPSL} via algorithm tr. Let M^{Pr}_{IS} be the unwinding of Υ^{Pr}_{IS}. In M^{Pr}_{CAPSL}, let $\sigma(A\text{-}role) \in \Sigma$, $\sigma(A) \neq \mathbf{I}$, $s, s' \in S$ and $e = (\omega|_A, \sigma, i)$ be a send event with $act(e) = A!B : (M)t$ and $s[e\rangle s'$. Let $\sigma^+(A\text{-}role) \mapsto_{\Upsilon_{IS}} ag_A$ and $\sigma^+ \cong \sigma$, $g, g', g'' \in G$, $1 \leq n \leq \sharp maxSubstMsg$ and $g\overline{a'}\beta_{Analz}g'\overline{a}_1 \ldots \overline{a}_n g''$ be the application of a (complete) intercept-analz-forge series of Dolev-Yao actions with β_{Analz} a complete analz sequence such that $e\,\tilde{v}^+\,\overline{a'}\beta_{Analz}\overline{a}_1 \ldots \overline{a}_n$. Then, $s'\,\tilde{p}\,g''$ and, in general, $s\,\tilde{p}^+\,[g']_{analz}$.*

Proof The proof follows trivially from the proof of Lemma 4.3.8, Remark 4.3.9 and Remark 4.3.14. ∎

To sum up, by all definitions and lemmas given so far, we have shown that the $\tilde{p}$-relation between $S_{M_{CAPSL}}$ and $G_{M_{IS}}$ follows the original intuition in pages 144–146. More precisely, we have shown that this relation is induced by the algorithm *tr* between the respective initial states of M_{CAPSL} and M_{IS} and that it is preserved under the application of send events in M_{CAPSL} and respectively send-intercept, full Dolev-Yao analysis and full Dolev-Yao synthesis in M_{IS}. Through Figure 4.6 we summarise all the aforementioned results referring to the $\tilde{p}$-relation and, at the same time, we also complete the picture we were giving originally in Figure 4.3. As a side note, it should be clear that relations $\tilde{v}^+$, $=_{analz}$ and even $\tilde{p}^+$ are helper notions to presenting the results in a structured, formal way.

In light of the above, it remains to investigate how the send events/send actions affect a relation between states in the two models which concern the ability of the intruder to synthesise terms. The next definitions and lemmas will view precisely this.

Definition 4.3.17 (Relation $\tilde{p}'$ between States in M_{CAPSL} and M_{IS}) *For Pr a bounded protocol, let M^{Pr}_{CAPSL} and Υ^{Pr}_{IS} be the models in algorithm tr, as above. Let M^{Pr}_{IS} be the unwinding of*



s
s'
X
send
$\overline{X} = closure(synth(analz(X)))$
$\tilde{p}$
$\tilde{p}$ (Lemma 4.3.8)
(Lemma 4.3.16) $\tilde{p}^+$
(Lemma 4.3.16) $\tilde{p}$
$\{g'_1, \ldots, g'_m, g_n\} = [g_j]_{analz}$
g_1
g_2
g_3
g_i
g_j
g_n
$a_{ag} = send$
$a_{Env} = intercept$
$a_{Env} = analz$
$a_{Env} = analz$
$a_{Env} = synth$
$a_{Env} = synth$
X
$\overline{X} = closure(synth(analz(X)))$

Figure 4.6 Proven Preservations of $\tilde{p} \subseteq S_{M_{CAPSL}} \times G_{M_{IS}}$ in M_{CAPSL} and M_{IS}

the Υ_{IS}^{Pr}. Let ξ be an arbitrary (T,k)-run in M_{CAPSL}^{Pr} and α be a computation in M_{IS}^{Pr}. Let $s \in S$ be a non-initial state of ξ and $g \in G$ be a non-initial state of α. We say the state s is in the relation $\tilde{p}' \subseteq S \times G$ *with the state g, $s\,\tilde{p}'\,g$, if:*

for $\sigma_1 \in \Sigma|_{\mathbf{I}}$, $\sigma_1^+ \in \Sigma^+, \sigma_1^+ \cong \sigma$, $\sigma_2^+ \supset \sigma_1^+$, with $\sigma_2^+(\mathbf{I}) \mapsto_{\Upsilon_{IS}} Env$:

(1). $\forall t \in T$, $\Sigma(t) \in \overline{\sigma_1(s_I)} \Rightarrow \Sigma^+(t) \in g_{Env}.synth\_log$

for $\sigma_1 \in \Sigma|_{\mathbf{I}}$, $\sigma_1^+ \in \Sigma^+, \sigma_1^+ \cong \sigma$, $\sigma_2^+ \supset \sigma_1^+$, with $\sigma_2^+(\mathbf{I}) \mapsto_{\Upsilon_{IS}} Env$:

(2). $\forall t \in T$, $\Sigma(t) \notin \overline{\sigma_1(s_I)} \Rightarrow \Sigma^+(t) \notin g_{Env}.synth\_log$

For $s \in S$, $g \in G$, if $s\,\tilde{p}'\,g$ we say that the state s is in the $\tilde{p}'$-relation *with the state g.*

In Definition 4.3.17, the state $s \in S$ being in the $\tilde{p}'$-relation with the state $g \in G$, $s\,\tilde{p}'\,g$, expresses the following: if the intruder is able/unable to synthesise t as some value-string x at state s in M_{CAPSL}, then the intruder is able/unable to synthesise t as some value-string x' at state g in M_{IS}; moreover, $x = \Sigma(t)$ and $x' = \Sigma^+(t)$ imply that the values x and x' are identical up to some permutation of elements in $\mathcal{R}_{T_0}$ and in the initial state setup of Υ_{IS}.

Also note that the initial states are in $\tilde{p}'$-relation, i.e., $\tilde{p}_0 \subseteq \tilde{p}'$.

In the following, Lemma 4.3.18 completes the previous results and it shows that the send events and intercept-analz-forge series applied respectively in the two models preserve the $\tilde{p'}$-relation as well.

Lemma 4.3.18 *For Pr a bounded protocol, let M^{Pr}_{CAPSL} and Υ^{Pr}_{IS} be the models in algorithm tr, as above. Let M^{Pr}_{IS} be the unwinding of the Υ^{Pr}_{IS}. Let $s \in S$ along a run of M_{CAPSL} and $g \in G$ along a computation of M_{IS} such that $s\,\tilde{p}'\,g$. Let $A \in Ho$ and $\sigma(A\text{-}role) \in \Sigma$ be a substitution for the A-role. Let $B \in Ho, t \in \mathcal{T}$, $s' \in S$ and $e = (\omega|_A, \sigma, i)$ be a a send event with $act(e) = A!B : (M)t$ such that $s[e\rangle s^{'}$. Let $\sigma^+ \cong \sigma$ and $\sigma^+(A - role) \mapsto_{\Upsilon_{IS}} ag_A$. Let $g' \in G$ and $\overline{a}_0\beta_{Analz}\gamma_{Forge}$ be a complete intercept-analz-forge series of Dolev-Yao actions enabled at g such that $g\overline{a}_0\beta_{Analz}\gamma_{Forge}g^{'}$ such that $e\tilde{v}^+\beta_{Analz}\gamma_{Forge}$.*

Then, $s^{'}\tilde{p}'g^{'}$.

Proof Let S be a set of terms. The proof follows is a similar way to the proof of Lemma 4.3.8, where instead we focus on synthesis.

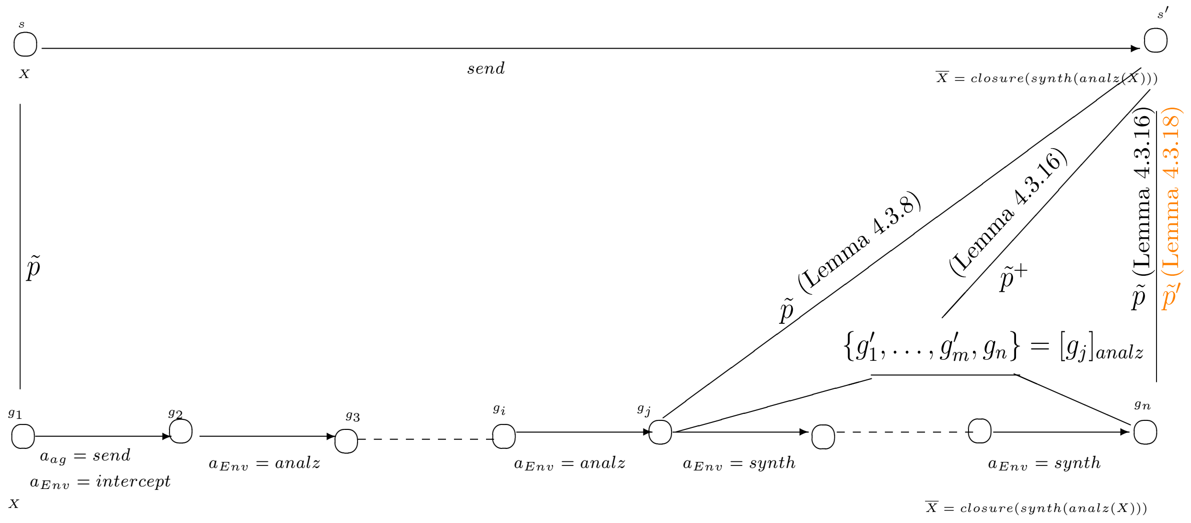
In a nutshell, it is entailed from the facts that $s\tilde{p}'g$ and $(synth^{\mathbb{I}})^{\sharp maxSubstMsg}(S)$=$synth_{M_{CAPSL}}(S)$.

$\blacksquare$

By Lemma 4.3.18, we show that the send events and the intercept-analz-forge series applied respectively in the two models preserve the (in-)ability of the intruder to synthesise the same terms at an M_{CAPSL} state as at an M_{IS} state.

Lemma 4.3.16 and Lemma 4.3.18 imply that send events in M_{CAPSL} and complete intercept-analz-forge series of Dolev-Yao actions in M_{IS} preserve the $\tilde{p}$ and the $\tilde{p'}$ relation between states in the two models, i.e., preserve the state-relations viewing both Dolev-Yao analysis and Dolev-Yao synthesis. At a higher level, the important message of Lemmas 4.3.16 and 4.3.18 is that the abilities of analysis and synthesis of the intruder are kept identical (up to renaming of values) over the application of send events in M_{CAPSL} and corresponding sequences of actions in M_{IS}, respectively. These results are illustrated in Figure 4.7.

Having studied the effect of send events and send actions onto the desired relations between states, we now lay the foundations for the relation between receive-events and receive action in the two respective models.



s
s'
X
$send$
$\overline{X} = closure(synth(analz(X)))$
$\tilde{p}$
$\tilde{p}$ (Lemma 4.3.8)
$\tilde{p}^+$ (Lemma 4.3.16)
$\tilde{p}$ (Lemma 4.3.16)
$\tilde{p}'$ (Lemma 4.3.18)
$\{g'_1, \ldots, g'_m, g_n\} = [g_j]_{analz}$
g_1
g_2
g_3
g_i
g_j
g_n
$a_{ag} = send$
$a_{Env} = intercept$
X
$a_{Env} = analz$
$a_{Env} = analz$
$a_{Env} = synth$
$a_{Env} = synth$
$\overline{X} = closure(synth(analz(X)))$

Figure 4.7 Proven Preservations of $\tilde{p}$ and $\tilde{p}'$ between M_{CAPSL} and M_{IS}

Definition 4.3.19 (Relation $\tilde{u}'$ between Receive Events and Receive Local Actions) *Let Pr be an RT protocol and M^{Pr}_{CAPSL} be its bounded protocol model. Let $A \in Ho$, $B \in Ho$, $t \in \mathcal{T}$, $\sigma(A\text{-}role) \in \Sigma$, $\sigma(A) \neq \mathbf{I}$, $i \in Steps^A$, $e = (\omega, \sigma, i)$ be an event in M^{Pr}_{CAPSL} such that $act(e) = A?B : t$ and let e be enabled at some state $s \in S$ in M^{Pr}_{CAPSL}. Let Υ^{Pr}_{IS} be obtained via algorithm tr from M^{Pr}_{CAPSL}. Let $\sigma^+ \in \Sigma^+$, $\sigma^+ \cong \sigma$, $\sigma^+(A\text{-}role) \mapsto_{\Upsilon_{IS}} ag_A$ and $a = receive$ be in Act_{ag_A} in Υ^{Pr}_{IS}. The receive-event e is* in the *relation $\tilde{u}' \subseteq Ev^{Pr} \times LAct$ with the local action a, $e\,\tilde{u}'\,a$, if: there exists $g \in G_{M_{IS}}$ such that $g_{ag_A}.step = i$ and $a \in P_{ag_A}(g_{ag_A})$.*

For $e \in Ev^{Pr}$ and $a \in LAct$, if $e\,\tilde{u}'\,a$ we say that the event e is in the $\tilde{u}'$-relation *with the local action a.*

Definition 4.3.19 simply reiterates the way receive actions in Υ^{Pr}_{IS} are created by algorithm *tr* from receive events in M^{Pr}_{CAPSL}. It simply states that if a principal A under instantiation σ can receive a term t in M_{CAPSL}, then agent ag_A under σ^+ can receive the term t in Υ_{IS}, too. Hence, the event/action are identical up to some renaming of values (i.e., up to $\sigma^+ \cong \sigma$).

In the following definition, we will lift the $\tilde{u}'$-relation to the operational level, i.e, from Υ_{IS} to M_{IS}.

Definition 4.3.20 (Relation $\tilde{v}'$ between Receive Events and Joint Actions) *Let Pr be an RT protocol and M^{Pr}_{CAPSL} be its bounded protocol model. Let $A \in Ho$, $B \in Ho$, $t \in \mathcal{T}$, $\sigma(A\text{-}role) \in \Sigma$,*

$\sigma(A) \neq \mathbf{I}$, $i \in Steps^A$, $e = (\omega, \sigma, i)$ be an event in M^{Pr}_{CAPSL} such that $act(e) = A?B : t$ and let e be enabled at some state $s \in S$ in M^{Pr}_{CAPSL}. Let Υ^{Pr}_{IS} be obtained via algorithm tr from M^{Pr}_{CAPSL}.Let M^{Pr}_{IS} be the unwinding of the Υ^{Pr}_{IS}.Let $\sigma^+ \in \Sigma^+$, $\sigma^+ \cong \sigma$, $\sigma^+(A\text{-}role) \mapsto_{\Upsilon_{IS}} ag_A$ and $a_1 = receive$ be in Act_{ag_A} in Υ^{Pr}_{IS}. Let $\overline{a} \in Act$ be a joint action such that $\overline{a}_{ag_A} = a_1$ and $\overline{a}_{Env} = transmit(ag_A, t, x)$, $x \in \mathcal{R}_t$. We say action e is in the relation *$\tilde{v}' \subseteq Ev^{Pr} \times Act$ with the joint action $\overline{a}$, $e\,\tilde{v}'\,\overline{a}$, if:*

$$\begin{cases} e\,\tilde{u}'\,a_1 \text{ with } a_1 \in P_{ag_A}(g_{ag_A}),\ g \in G & (1) \\ ((s[e\rangle s') \text{ and } (s\,\tilde{p}\,g) \text{ and } (s\,\tilde{p}'\,g)) \Rightarrow E_{ag_A}(g_{ag_A}, \overline{a}) = g'_{ag_A} & (2) \end{cases}$$

For $e \in Ev^{Pr}$ and $a \in Act$, if $e\,\tilde{v}'\,a$ we say that the event e is in the $\tilde{v}'$-relation *with the joint action a.*

Definition 4.3.20 takes the $\tilde{u}'$-relation to the operational level. A receive event and a joint action are in the $\tilde{v}'$-relation if not only are they identical up to renaming, but they are effectively applied at states which are in the $\tilde{p}$-relation and the $\tilde{p}'$-relation. The notions introduced by Definition 4.3.20 are used to express more easily the evolution of the relations between states over computations, i.e., avoid verbose proofs.

The following lemma proves that the receive events in M_{CAPSL} and receive actions in M_{IS} indeed preserve the $\tilde{p}$-relation and the $\tilde{p}'$-relation between states. In Appendix B we give the informal explanations of this proof. Below, we present the lemma formally.

Lemma 4.3.21 *Let Pr be an RT protocol and M^{Pr}_{CAPSL} be its bounded protocol model. Let Υ^{Pr}_{IS} be obtained from M^{Pr}_{CAPSL} via algorithm tr. Let M^{Pr}_{IS} be the unwinding of the Υ^{Pr}_{IS}. Let $\sigma(A\text{-}role) \in \Sigma$, $\sigma(A) \neq \mathbf{I}$, $s, s' \in S$, $e = (\omega|_A, \sigma, i)$ be a receive event in M^{Pr}_{CAPSL} such that $act(e) = A?B : (M)t$ and $s[e\rangle s'$. Let $t \in Msg$, $x \in \mathcal{R}_t$, $\sigma^+ \in \Sigma^+$, $\sigma^+ \cong \sigma$, $\sigma^+(A\text{-}role) \mapsto_{\Upsilon_{IS}} ag_A$, $g, g' \in G$ and $g\overline{a}g'$ be the application of a joint action $\overline{a}$ in M^{Pr}_{IS} such that $e\,\tilde{v}'\,\overline{a}$, where $\overline{a}_{ag_A} = receive$ and $\overline{a}_{Env} = transmit(ag_A, t, x)$. Then, $s'\,\tilde{p}\,g'$ and $s'\,\tilde{p}'\,g'$.*

Proof Recall that to prove $s'\,\tilde{p}\,g'$, we need to prove that parts (1)–(3) in the definition of the $\tilde{p}$-relation hold for s' and g' (i.e., Definition 4.3.2).

Let $s_0[\xi\rangle s[e\rangle s'$, where ξ is an arbitrary run in M_{CAPSL}, $s_0 \in S_0$ and $act(e) = A?B : (M)t$ as in the hypothesis. If $s[e\rangle s'$, then by the definition of enabling of events in M^{Pr}_{CAPSL}, $\xi \supseteq^{\bullet} e$. This triggers that t_0 is substituted by $\sigma(t_0)$, i.e., $[\sigma(t_0)/t_0]$, for all $t_0 \in OSub(t) \cap OwnedAtoms^A \cap OSub(analz(s_A))$. In other words, since the event e is under a substitution σ suitable for e, then all atomic terms of t previously transmitted along the run ξ have been substituted by σ in M^{Pr}_{CAPSL} $(*)$.

Given the way algorithm *tr* constructs Υ_{IS} from M^{Pr}_{CAPSL}, i.e., from the events along the runs of M^{Pr}_{CAPSL} with $\sigma^+ \cong \sigma$, and given that $s\,\tilde{p}\,g$, then $(*)$ implies that $out\_match^{\mathbb{I}^{\sigma^+}}(t, x, g_{ag_A}@i)$ returns *true*. Hence, g does get updated under σ^+ to $g^{'}$ in M^{Pr}_{IS} when s moves at s' under σ in M^{Pr}_{CAPSL}, for $\sigma^+ \cong \sigma$.

Now, recall that the state update in M_{CAPSL} dictates $s^{'}_A = s_A \cup \{t\}$ and the state update in Υ_{IS} dictates $set^{\mathbb{I}^{\sigma^+}}(t, g_{ag_A}@i)$ $(**)$.

For part (1) of $s^{'}\,\tilde{p}\,g^{'}$ to hold, we need that: $[\sigma(t_0)/t_0] \in analz(\sigma(s^{'}_A \cup Gen)) \Rightarrow [\Sigma^+(t_0)/t_0] \in g^{'}_{ag_A}$, for $t_0 \in Atoms^A$ arbitrary.

Then, let $t_0 \in Atoms^A$ arbitrary such that $[\sigma(t_0)/t_0] \in analz(\sigma(s^{'}_A \cup Gen))$, for $Gen \subseteq Generated^A$. This is equivalent to saying that $[\sigma(t_0)/t_0] \in analz(\sigma(s_A \cup \{t\} \cup Gen))$, for $Gen \subseteq Generated^A$. Since $s\,\tilde{p}\,g$, the only interesting case to discuss is when t_0 is analysable due to the recent receival of t (i.e., $t_0 \in i\text{-}LearnedAtoms^A$). So, consider the set Gen' of a minimal generation such that $[\sigma(t_0)/t_0] \in analz(\sigma(s_A \cup \{t\} \cup Gen^{'})) \setminus analz(\sigma(s_A \cup Gen^{'}))$ $(***)$. As we deal here with RTP and $t_0 \in i\text{-}LearnedAtoms^A$ then $t_0 \in OSub(t)$ (i.e., all messages upon receival are analysable down to atomic parts and we are under $(***)$). But this implies that t_0 has been set at $g^{'}_{ag_A}$ through $set^{\mathbb{I}^{\sigma^+}}(t, g_{ag_A}@i)$. Hence, $[\Sigma(t_0)/t_0] \in g^{'}_{ag_A}$. Thus, we have part (1) of $s^{'}\,\tilde{p}\,g^{'}$.

The part (2) of $s^{'}\,\tilde{p}\,g^{'}$ holds trivially as the intruder does not gain new data through this event/joint action nor does it perform any action of type analz or synth. For the same reason, it is trivial that $s^{'}\,\tilde{p}'\,g^{'}$. ∎

Lemma 4.3.21 completes the picture of the previous results and shows that receive events/actions preserve the "good" relations between states in the two models as well.

All in all, in Section 4.2.1 we have given an algorithm called *tr* that produces Υ^{Pr}_{IS} from M^{Pr}_{CAPSL}. The unwinding of the model produced has the initial states in the $\tilde{p}$-relation with the initial states of M^{Pr}_{CAPSL}. In this section we have shown that the events in M_{CAPSL} and actions or sequences of actions in M_{IS} are such that they preserve the $\tilde{p}$-relation between states in the two models.

This entails that the ability of the intruder to analyse and synthesise terms is preserved along corresponding fragments of computations in the two models. Figure 4.8 summarises these results. This will be useful when we have to choose states that have homomorphic properties, in order to prove that the $\tilde{p}$-relation induces a homomorphism between the two models.

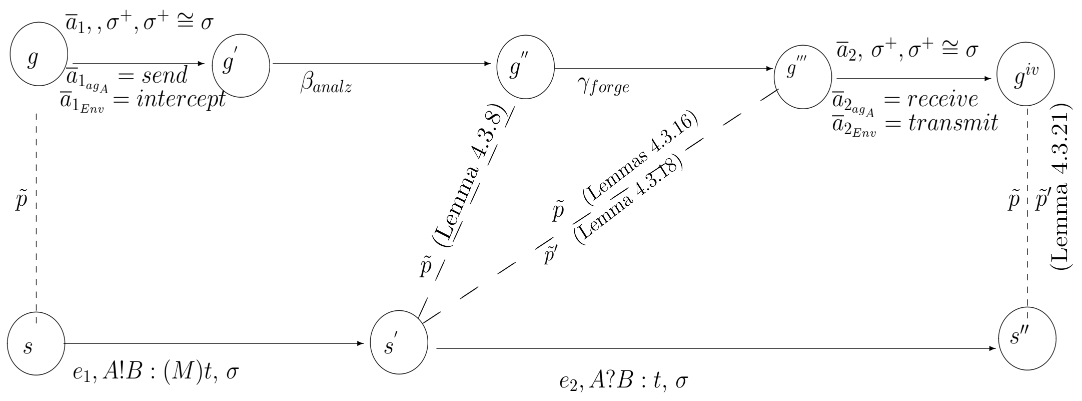

$\overline{a}_1, , \sigma^+, \sigma^+ \cong \sigma$
$\overline{a}_{1_{ag_A}} = send$
$\overline{a}_{1_{Env}} = intercept$
β_{analz}
γ_{forge}
$\overline{a}_2, \sigma^+, \sigma^+ \cong \sigma$
$\overline{a}_{2_{ag_A}} = receive$
$\overline{a}_{2_{Env}} = transmit$
g
g'
g''
g'''
g^{iv}
s
s'
s''
$\tilde{p}$
(Lemma 4.3.8)
(Lemmas 4.3.16)
(Lemma 4.3.18)
$\tilde{p}'$
(Lemma 4.3.21)
$e_1, A!B : (M)t, \sigma$
$e_2, A?B : t, \sigma$

Figure 4.8 Preserved Relations Between States in Computations of M_{CAPSL} and M_{IS}

In Appendix B we use the relations and lemmas given so far to prove by induction that that the $\tilde{p}$-relation is preserved all along respective computations of M_{CAPSL} and M_{IS}. The relations between events and actions are thereby used as well, to avoid verbose proofs.

4.4 A Homomorphism from M_{CAPSL} to M_{IS}

We will show that the $\tilde{p}$-relation, induced by the algorithm tr, defines a homomorphism between the models involved. Then, we will use this to prove that M_{CAPSL} validates $\rho_{CAPSL}(r)$ if and only if the model M_{IS} validates f, for all $f \in \rho_{IS}(r)$, for $\rho_{CAPSL}(r) \asymp \rho_{IS}(r)$ and r an atomic `CAPSL` goal of initial secrecy or agreement on initials.

We begin by introducing the transition systems underlying the two models.

Definition 4.4.1 (Transition Systems for M_{CAPSL}) *Let Pr be an RT protocol and M^{Pr}_{CAPSL} be its M_{CAPSL} model under a set of role-substitutions Σ_{Pr}. The tuple $TS^{Pr}_{CAPSL} = (S, Ev^{Pr}, \bigcup_{e \in Ev^{Pr}} \xrightarrow{e})$ denotes the* transition system underlined by *M^{Pr}_{CAPSL}. In TS^{Pr}_{CAPSL}, S is the set of states under Σ_{Pr},*

Ev^{Pr} is the set of all the labels of all (T,k)-bounded events, i.e., each label e unambiguously denotes the event $e \in Ev^{Pr}$ and $\bigcup_{e \in Ev^{Pr}} \xrightarrow{e}$ is a set of binary relations on S such that $s, s' \in S$, $s \xrightarrow{e} s'$ if $s[e\rangle s'$ as per M^{Pr}_{CAPSL}.

To define the transition system for M_{IS}, we first distinguish certain types of joint actions in the M_{IS} model. In more detail, we will first outline those actions that operate towards creating the closure under $synth^{\mathbb{I}}$ and $analz^{\mathbb{I}}$.

Definition 4.4.2 (τ_1-actions in M_{IS}) *A joint action a is a τ_1-action if $E(g,a) = g'$ implies that $analz^{\mathbb{I}^{[x'/t']}}(g_{Env}.analz\_log, g_{Env}.values\_log, t') = true$, for $t' \in Composites$, $x' \in \mathcal{R}_{t'}$ (i.e., $(t', x') \notin g_{Env}.analz\_log$).*

By Definition 4.4.2, a joint action a in M_{IS} is a τ_1-action if a_{Env} contains a component of type *analz* which effectively analyses $[x'/t']$ from g, leading to $g^{'}$. In other words, the closure of g under $analz^{\mathbb{I}}$ has not yet occurred if a τ_1 action is being applied.

Definition 4.4.3 (τ_2-actions in M_{IS}) *A joint action a is a τ_2-action if $E(g,a) = g'$ implies that $synth^{\mathbb{I}}(g_{Env}, t, x) = true$, for $t \in Msg$, $x \in \mathcal{R}_t$ and (i.e., $(t,x) \notin g_{Env}.synth\_log$).*

By Definition 4.4.3, a joint action of M_{IS} is a τ_2-action if a_{Env} contains a component of type $forge$ which effectively synthesise $[x/t]$ at g, leading to $g^{'}$. In other words, the closure of g under $synth^{\mathbb{I}}$ has not yet occurred if a τ_1 action is being applied.

Let $Act^{\tau} \subsetneq Act$ be the subset of all τ_i-actions of M_{IS}, for $i \in \{1,2\}$. Let the set $Act^{'} = Act \setminus Act^{\tau}$ denotes the actions which are not τ_i-actions of M_{IS}, for $i \in \{1,2\}$. For any $i \in \{1,2\}$, τ_i-actions are generically referred to as *τ-actions.*

Having formally differentiated the τ-actions from other actions, we can now define the transition systems for M_{IS}.

Definition 4.4.4 (Transition Systems for M_{IS}) *Let Pr be an RT protocol, M^{Pr}_{CAPSL} be its bounded protocol model under a set of role-substitutions Σ_{Pr} and Υ^{Pr}_{IS} be obtained via algorithm tr from M^{Pr}_{CAPSL}. Let M^{Pr}_{IS} be the unwound model implied by the interpreted system Υ^{Pr}_{IS}. The tuple*

$TS_{IS}^{Pr} = (G, Act, \bigcup_{a \in Act'} \xrightarrow{a} \cup \xrightarrow{\tau}, V)$ *denotes the* transition system underlined by M_{IS}^{Pr}, *where G is the set of global states as given by Σ_{Pr}^{+}, Act' is the set of all the labels of all joint actions except Act^{τ}, all τ-actions in M_{IS}^{Pr} are labelled τ, $\bigcup_{a \in Act'} \xrightarrow{a} \cup \xrightarrow{\tau}$ is a set of binary relations on G, for $g, g' \in G, x \in Act$, $g \xrightarrow{x} g'$ if $E(g, x) = g'$ in Υ_{IS}^{Pr} and V is the set of atomic propositions originally present in Υ_{IS}^{Pr}.*

In Definitions 4.4.5 and 4.4.6 we are going to revisit the definition of the two models. This revisiting will ease the handling of the transition relation particularly in the case of M_{IS} (i.e., given that in M_{IS}, we encounter analysis and synthesis series of Dolev-Yao actions of variable length).

Definition 4.4.5 (Revisited M_{CAPSL} Model) *Let Pr be an RT protocol and M_{CAPSL}^{Pr} be its bounded protocol model, as before. Then, M_{CAPSL}^{Pr} can be expressed as the tuple $M_{CAPSL}^{Pr} = (S, Ev^{Pr}, \{R_e\})$, with $\{R_e\} = \xrightarrow{e}$ as per the definition of TS_{CAPSL}.*

Definition 4.4.5 essentially states that the transition relation R_e in M_{CAPSL} is simply a computational step in the model.

We will now use the notion of τ-actions to outline the analz-forge sequences of the M_{IS} model, introduced in Section 4.3. Therefore, let $(\xrightarrow{\tau})^*$ denote any number of τ-actions.
We use $(\xrightarrow{\tau})^*$ and the transition systems TS_{IS} to revisit the M_{IS} model (originally given in Definition 3.3.21).

Definition 4.4.6 (Revisited M_{IS} Model) *Let Pr be an RT protocol, M_{CAPSL}^{Pr} be its bounded protocol model and Υ_{IS}^{Pr} be obtained via algorithm tr from M_{CAPSL}^{Pr}, as before. Let M_{IS}^{Pr} be the unwound model implied by the interpreted system Υ_{IS}^{Pr}. Then, the M_{IS}^{Pr} can be expressed as the tuple $M_{IS}^{Pr} = (G, Act, \{R_a\}, V)$, with $\{R_a\} \subseteq (\xrightarrow{\tau})^* \xrightarrow{a} (\xrightarrow{\tau})^*$ and V the set of atomic propositions in Υ_{IS}^{Pr}.*

By Definition 4.4.6, several τ-actions can be conceptually collapsed into one relational step between the states of M_{IS}.

Now, let $(\xrightarrow{\tau})^n$ denote either a complete analz-forge series (i.e., intercepting followed by a complete *analz* sequence and a complete *forge* sequence) or no τ-actions.

Definition 4.4.7 (Stuttering M_{IS} Model) *Let Pr be an RT protocol, M^{Pr}_{CAPSL} be its bounded protocol model and Υ^{Pr}_{IS} be obtained via algorithm tr from M^{Pr}_{CAPSL}, as before. Let M^{Pr}_{IS} be the unwound model implied by the interpreted system Υ^{Pr}_{IS}. The stuttering M_{IS} model is expressed by the tuple $\overline{M}^{Pr}_{IS}$ =$(G', Act, \{\overline{R}_a\}, V)$, with $G' \subsetneq G$, $\{\overline{R}_a\} \subseteq (\xrightarrow{\tau})^n \xrightarrow{a} (\xrightarrow{\tau})^n \subseteq G' \times G'$ and V the set of atomic propositions in Υ^{Pr}_{IS}.*

In Definition 4.4.7 we embedded explicit computational steps in M_{IS} into a transition relation that takes into account the *complete* intercept-analz-forge series in M_{IS}. The $\overline{M}^{Pr}_{IS}$ model is more compact than the M^{Pr}_{IS} model; the $\overline{M}^{Pr}_{IS}$ model collapses a *complete* intercept-analz-forge series into one computational step.

Note that $\overline{R}_a \subseteq R_a$. The relation R_a in M_{IS} allows for any number of analysis and/or synthesis actions, whereas the relation $\overline{R}_a$ refers to the special case where a complete sequence of these is applied.

From here onwards, we will refer to the models M_{CAPSL}, M_{IS}, $\overline{M}_{IS}$ as introduced in Definitions 4.4.5–4.4.7, respectively.

We will now prove that the relation $\tilde{p} \subseteq S_{M_{IS}} \times G_{M_{IS}}$ induces a homomorphism between M_{CAPSL} and M_{IS} and an epimorphism between M_{CAPSL} and $\overline{M}_{IS}$. To achieve that, we begin by Lemma 4.4.8 which shows that any state in S is $\tilde{p}$-related to some state in G.

Lemma 4.4.8 *Let Pr be an RT protocol, M^{Pr}_{CAPSL} be its bounded protocol model and Υ^{Pr}_{IS} be obtained via algorithm tr from M^{Pr}_{CAPSL}, as before. Let M^{Pr}_{IS} be the unwound model of the interpreted system Υ^{Pr}_{IS}.The relation $\tilde{p} \subseteq S_{M^{Pr}_{CAPSL}} \times G_{M^{Pr}_{IS}}$ is total.*

Proof The above means that $\forall s \in S$, $\exists g \in G$ such that $s\,\tilde{p}\,g$.

If s is an arbitrary initial state resulted from the instantiations in Ev^{Pr}, i.e., $s = s_0$, $s_0 \in S_0$, then algorithm *tr* creates g_0 is such that $s_0\tilde{p}_0g_0$. And, as per Section 4.3, $\tilde{p}_0 \subseteq p$.

By Theorem B.3.1, every state s along a run in M_{CAPSL} is in $\tilde{p}$-relation with some state g along a computation in M_{IS}.

Then the lemma is proven. ∎

In the next theorem we prove that the relation $\tilde{p} \subseteq S \times G$ defines a homomorphism between the M^{Pr}_{CASPL} and the M^{Pr}_{IS} models and and an epimorphism between M_{CAPSL} and $\overline{M}_{IS}$.

Theorem 4.4.9 *Let Pr be an RT protocol, M^{Pr}_{CAPSL} be its bounded protocol model and Υ^{Pr}_{IS} be obtained via algorithm tr from M^{Pr}_{CAPSL}, as before. Let M^{Pr}_{IS} be the unwound model of the interpreted system Υ^{Pr}_{IS}.*

The relation $\tilde{p} \subseteq S_{M^{Pr}_{CAPSL}} \times G_{M^{Pr}_{IS}}$ induces a homomorphism between the model M^{Pr}_{CAPSL} and the model M^{Pr}_{IS}.

The relation $\tilde{p} \subseteq S_{M^{Pr}_{CAPSL}} \times G_{M^{Pr}_{IS}}$ induces an epimorphism (i.e., surjective homomorphism) between the model M^{Pr}_{CAPSL} and the stuttering model $\overline{M}^{Pr}_{IS}$.

Proof To prove the theorem, we need to show the following:

1. for $s_1, s_2 \in S$ with $s_1 R_e s_2$ there exist $g_1, g_2 \in G$ such that ($s_1 \,\tilde{p}\, g_1$ and $s_2 \,\tilde{p}\, g_2$ and $g_1 R_a g_2$);

2. for all $g \in G'_{\overline{M}^{Pr}_{IS}}$, there exists $s \in S$ such that $s \,\tilde{p}\, g$.

We start by proving Point 1 above.

Assume $s_1, s_2 \in S$ arbitrary and $s_1 R_e s_2$, hence there exists $e \in Ev^{Pr}$ such that $s_1[e\rangle s_2$.

• If $s_1 \in S_0$, then by the construction in *tr*, there exists $g_1 \in G_0$ such that $s_1 \,\tilde{p}_0\, g_1$ and, by $\tilde{p}_0 \subset p$, it follows that $s_1 \,\tilde{p}\, g_1$.

Recall the working hypothesis, i.e., $s_1 R_e s_2$. So, there is an event e applied at s_1 to produce $s_1[e\rangle s_2$. Assume that this event e is a send event in M_{CAPSL} under some instantiation σ, i.e., $act(e) = A!B : (M)t$ for some $A, B \in Ho, t \in \mathcal{T}$, $M \subset Generated^A$. Take $\overline{a} \in Act_{M_{IS}}$ such that $e \,\tilde{v}\, \overline{a}$ (such an action $\overline{a}$ exists, by the construction in algorithm *tr* and as Lemma 3.3.17 has shown). Recall that, in M_{IS}, a linear series of full analysis and a full synthesis are enforced (by *tr*) after the application of $\overline{a}$. Then, let $\beta_{analz}\gamma_{forge}$ be a complete intercept-analz-forge sequence such that $g_1\overline{a}\beta_{analz}\gamma_{forge}g$. By the definition of the R_a relation, $g_1 R_a g$. And, by Lemma 4.3.16, $s_2 \,\tilde{p}\, g$. $(*)$

We are looking for a state g_2 such that $g_1 R_a g_2$ and $s_2 p g_2$. Choose this state g_2 to be g in $(*)$.

So, we have found $g_1, g_2 \in G$ such that $s_1 \,\tilde{p}\, g_1$ and $s_2 \,\tilde{p}\, g_2$ and $g_1 R_a g_2$, for $s_1, s_2 \in S$ and $s_1 \in S_0$. $(\circledast)$

• If $s_1 \in S \setminus S_0$, then it means that there exists ξ a M^{Pr}_{CAPSL} run constructed as in algorithm *A*1 [188] and $s_0[\xi\rangle s_1$, for some $s_0 \in S_0$.

By the construction in algorithm tr, there exists $g_0 \in G$ such that $s_0\,\tilde{p}_0\,g_0$. Then, by Theorem B.3.1 there exists α a computation in M^{Pr}_{IS}, there exists $g \in G$ such that $g_0 \alpha g_1$ and $s_1\,\tilde{p}\,g$, $s_1\,\tilde{p}'\,g$. $(**)$

We are looking for a state $g_1 \in G$ such that $s_1\,\tilde{p}\,g_1$. Choose this state g_1 to be g in $(**)$.

So, we have found g_1 as desired. It remains to find a state $g_2 \in G$ that complies with the demands. For $s_1 R_e s_2$ in the working hypothesis, assume first that it is a send event which was applied at s_1 to produce $s_1[e\rangle s_2$. The argument for choosing g_2 as desired is analogous with the case $s_1 \in S_0$ above.

For $s_1 R_e s_2$ in the working hypothesis, assume now that the event e applied at s_1 to produce $s_1[e\rangle s_2$ is a receive event. Then, by the construction of algorithm tr, there exists $\overline{a} \in Act_{M_{IS}}$ such that $e\,\tilde{u}'\,\overline{a}_{ag_A}$ (i.e., $\overline{a}_{ag_A}$ is identical to e up to renaming of values and it is possible in Υ_{IS} at g). Let $g' \in G$ such that $g_1 \overline{a} g'$. Given that $e\,\tilde{u}'\,\overline{a}_{ag_A}$, $s[e\rangle$, $g_1 \overline{a} g'$ and $s\,\tilde{p}\,g$, by Lemma 4.3.21, it follows that $s_2\,\tilde{p}\,g'$ $(* * *)$.

We are looking for a state g_2 such that $g_1 R_a g_2$ and $s_2\,\tilde{p}\,g_2$. Choose g' in $(* * *)$ to be this state g_2.

So, we have found $g_1, g_2 \in G$ such that $s_1\,\tilde{p}\,g_1$ and $s_2\,\tilde{p}\,g_2$ and $g_1 R_a g_2$, for $s_1, s_2 \in S \setminus S_0$. $(\circledast\circledast)$

With $(\circledast)$ and $(\circledast\circledast)$, we have proven Point 1 in this theorem, i.e., we have proven that the relation $\tilde{p} \subseteq S_{CAPSL} \times G_{IS}$ induces a homomorphism between the models M^{Pr}_{CAPSL} and M^{Pr}_{IS}.

We proceed to prove Point 2, i.e., that this homomorphism is in fact an epimorphism between M^{Pr}_{CAPSL} and the stuttering model $\overline{M}^{Pr}_{IS}$.

Let $g \in G'$.

For $g \in G_0$, by the construction of algorithm tr, there exists $s_0 \in S_0$ such that $s_0\,\tilde{p}_0\,g_0$. Moreover, $\tilde{p}_0 \subseteq \tilde{p}$. Hence, there exists $s \in S$, $s = s_0$ such that $s\,\tilde{p}\,g$. $(*)$

Let us now deal with the case where g is a non-initial state in M_{IS}, i.e., $g \in G' \setminus G_0$.

Let $s_0 \in S_0$. The algorithm tr will construct $g_0 \in G_0$ such that $s_0\,\tilde{p}\,g_0$. $(\mathbf{1})$

Let α be some arbitrary computation in M_{IS} such that $g_0 \alpha g$. Let the arbitrarily obtained $g \in G$ be the non-initial state in M_{IS} under consideration.

Let $g_1, \ldots, g_n$ be the states along α such that $g_0 \overline{R}_a\, g_1$, $g_i \overline{R}_a g_{i+1}$ and $g_{n-1} \overline{R}_a g$, with $i = \overline{1, n-2}$. Recall that $\overline{R}_a = (\xrightarrow{\tau})^n \xrightarrow{a} (\xrightarrow{\tau})^n$, where $(\xrightarrow{\tau})^n$ means either no τ-actions or a complete intercept-analz-forge sequence of them.

Consider the computation fragment $g_i\overline{R}_a g_{i+1}$ in the aforementioned α.

If a is a sending action, then let η be an complete intercept-analz-forge sequence. By construction of algorithm tr, there exists $e_1 \in Ev^{Pr}$ such that $e_1\,\tilde{u}\,a$. (**2**)

If a is a receiving action, then by construction of algorithm tr there exists $e_2 \in Ev^{Pr}$ such that $e_2\,\tilde{u}'\,a$. (**3**)

Starting with $g_0\overline{R}_a g_1$ and continuing with all $g_i\overline{R}_a g_{i+1}$ along α, "replace" η with e_1 in (**2**) and the receiving a with e_2 in (**3**). Start the application of the so-obtained events at s_0 in (**1**) and apply them in order (i.e., these events will be enabled at the resulting M_{CAPSL} states, due to the way algorithm tr obtained M_{IS} actions out of M_{CAPSL} events). Like this, we obtain a run ξ in M_{CAPSL}. Let $s \in S$ such that $s_0[\xi\rangle s$. By Section 4.3, $e_1\,\tilde{v}^+\,\eta$ and $e_2\,\tilde{v}'\,a$, for each e_1, e_2, a, η as above. Then, since we start with $s_0\,\tilde{p}\,g_0$, the lemmas in Section 4.3 guarantee that $s\,\tilde{p}\,g$.

So, for an arbitrary $g \in G' \setminus G_0$ we can always "reverse engineer" the reasoning behind the algorithm tr, taking into consideration the complete intercept-analz-forge series in $\overline{M}_{IS}$. These will lead us to finding $s \in S$ such that $s\,\tilde{p}\,g$. $(**)$

By $(*)$ and $(**)$, we prove Point 2 above.

By proving Point 1 above, we have shown that $\tilde{p} \subseteq S_{M_{CAPSL}} \times G_{M_{IS}}$ induces a homomorphism from M^{Pr}_{CAPSL} to M^{Pr}_{IS}. Then, by proving Point 2 above we have shown that this homomorphism is actually an epimorphism from M^{Pr}_{CAPSL} to $\overline{M}^{Pr}_{IS}$. ▮

Theorem 4.4.10 *Let Pr be an RT protocol, M^{Pr}_{CAPSL} be its bounded protocol model, Υ^{Pr}_{IS} be the output of algorithm tr given M^{Pr}_{CAPSL}. Let M^{Pr}_{IS} be the unwinding of the interpreted system Υ^{Pr}_{IS}.*

Let r be an atomic `CAPSL` *goal such that $\rho_{CAPSL}(r) \asymp \rho_{IS}(r)$.*

The model M^{Pr}_{CAPSL} validates $\rho_{CAPSL}(r)$ if and only if M^{Pr}_{IS} validates all formulations of $\rho_{IS}(r)$.

Proof The statement we need to prove can be formally rewritten as:
for all $r \in G$ such that $\rho_{CAPSL}(r) \asymp \rho_{IS}(r)$,
M^{Pr}_{CAPSL} validates $\rho_{CAPSL}(r)$ if and only if $M^{Pr}_{IS} \models f$, for all $f \in \rho_{IS}(r)$.

Let r be expressed by "`SECRET t`", "`Holds A:t`", for A and t arbitrary in the given context. Synonymously, let r be an initial secrecy goal.

"$\Rightarrow$:" Assume that M^{Pr}_{CAPSL} validates $\rho_{CAPSL}(r)$; we need to prove that $M^{Pr}_{IS} \models f$, for all $f \in \rho_{IS}(r)$.

The fact that M^{Pr}_{CAPSL} validates $\rho_{CAPSL}(r)$ is synonymous to saying that any run of M^{Pr}_{CAPSL} is non-leaky [188]. Hence, for $s_0 \in S_0$,$s \in S$, for any arbitrary run ξ in M^{Pr}_{CAPSL} with $s_0[\xi\rangle s$, it is the case that $t \in analz(s_A) \setminus analz(s_{\mathbf{I}})$. (**hyp**)

Let us assume by reductio ad absurdum that there exists $f \in \rho_{IS}(r)$ such that $M^{Pr}_{IS} \not\models f$. (**sup**)

Recall that the formulations of $\rho_{IS}(r)$ are:

- (S_1)= $AG(g_{Env}.values\_log.t \neq g_{ag_A}.t)$, for any ag_A corresponding to the arbitrary principal A in the `CAPSL`-formulation of the goal r (through algorithm tr, $\sigma^+(A\text{-}role) \mapsto_{\Upsilon_{IS}} ag_A$, $\sigma^+ \cong \sigma$);
- $(S2)$=$AG(\neg K_{Env}(Holds^{\mathbb{I}^{\sigma^+}}(A,t)))$, for $\sigma^+ \cong \sigma$, σ for A-*role*, A being the principal in the `CAPSL`-formulation of the goal r.

Following (**sup**), let us assume first that $(S1)$ is refuted in M^{Pr}_{IS}. Given the formulation of $(S1)$, it follows that there exists a path $\pi = \pi_1 \ldots \pi_n$ in M^{Pr}_{IS} such that there exists i, $i \in \overline{1,n}$ such that $\pi_i = g$ and $g_{Env}.values\_log.t = g_{ag_A}.t$. (**i**)

Since complete intercept-analz-forge series of actions are enforced in M_{IS} and given the persistence of the value-setting in M_{IS}, without loss of generality, we can take the state g in (**i**) to be a final state of a complete intercept-analz-forge sequence in M_{IS} (i.e., $g \in G'_{\overline{M}_{IS}}$).

Since the relation $\tilde{p}$ is an epimorphism between M_{CAPSL} and $\overline{M}_{IS}$, it follows that there exists $s \in S$ such that $s\,\tilde{p}\,g$. From (**i**) and the definition of the $\tilde{p}$-relation, it follows that there exists $s \in S$ such that $t \in analz(s_{\mathbf{I}})$. Given how the algorithm tr has created $g \in G$ and actions in M^{Pr}_{IS}, there exists a run containing s in M^{Pr}_{CAPSL}. Since $t \in analz(s_{\mathbf{I}})$, this run is leaky. Hence, this contradicts the working hypothesis (**hyp**). Hence, the supposition of $(S1)$ being refuted in M^{Pr}_{IS} cannot hold. (**ii**)

Following (**sup**), let us assume now that $(S2)$ is refuted in M^{Pr}_{IS}. Given the formulation of $(S2)$, it follows that there exists a path $\pi = \pi_1 \ldots \pi_n$ in M^{Pr}_{IS} such that there exists i, $i \in \overline{1,n}$ such that $\pi_i = g$ and there exists $g' \in G$, $g' \sim_{Env} g$ such that $Holds^{\mathbb{I}^{\sigma^+}}(A,t)$ is true at g_{Env}. This would be the case if $atoms\_log$ contained (t, m, ag_A), for some $m \in Msg$. By the semantics of $analz^{\mathbb{I}}$, this implies that there exists $g'' \in G$ with $g'\overline{R}_a g''$ and $g' R_a g''$ such that $g''.values\_log.t = ag_A.t$. (**iii**)

Since $\tilde{p}$ is an epimorphism between M_{CAPSL} and $\overline{M}_{IS}$, for g'' in (**iii**) there exists a state $s \in S$ such that $s\,\tilde{p}\,g''$. By (**iii**) and the definition of the $\tilde{p}$-relation, then $t \in analz(s_{\mathbf{I}})$. Hence, there exists a run containing s in M^{Pr}_{CAPSL}. Since $t \in analz(s_{\mathbf{I}})$, this run is leaky. Hence, this contradicts the working hypothesis (**hyp**). Hence, the supposition of $(S2)$ being refuted in M^{Pr}_{IS} cannot hold. (**iv**)

Therefore, by (**ii**) and (**iv**), we proved the direct implication of this theorem.

"$\Leftarrow$:" Assume that M^{Pr}_{CAPSL} refutes $\rho_{CAPSL}(r)$; we need to prove that there exists $f \in \rho_{IS}(r)$ such that $M^{Pr}_{IS} \not\models f$.

The fact that M^{Pr}_{CAPSL} refutes $\rho_{CAPSL}(r)$ means that there exists a run of M^{Pr}_{CAPSL} which is leaky [188]. Hence, for $s_0 \in S_0$, $s \in S$, there exists ξ in M^{Pr}_{CAPSL} with $s_0[\xi\rangle s$ such that $t \in analz(s_{\mathbf{I}})$. (**hyp2**)

For convenience, let us assume the non-trivial case of $\xi = \xi' e$, for ξ' a run and $e \in Ev^{Pr}$ an event. The trivial case when ξ is a single event is treated in the same way. Further, let $s_1 \in S$ such that $s_0[\xi\rangle s_1[e\rangle s$.

So, $s_1 R_e s$. And as $\tilde{p}$ is a homomorphism between the two models involved, it follows that there exist g_1 and g_2 such that $s_1\,\tilde{p}\,g_1$ and $s\,\tilde{p}\,g_2$ and $g_1 R_a g_2$. (**v**)

From (**v**), (**hyp2**), $s\,\tilde{p}\,g_2$ and the definition of the $\tilde{p}$-relation, it follows that $g_{2_{Env}}.values\_log.t = ag_A.t$, for ag_A such that $\sigma^+(A\text{-}role) \mapsto_{\Upsilon_{IS}} ag_A$, $\sigma^+ \cong \sigma$. (**vi**)

From (**v**) and (**vi**), it follows that there is a path π in M^{Pr}_{IS} such that g_2 is on π. From (**vi**) and the semantics of CTLK on M_{IS}, it follows that $M^{Pr}_{IS} \not\models f$, where $f \in \rho_{IS}(r)$ and f is the $(S1)$ formulation of the `CAPSL` goal r.

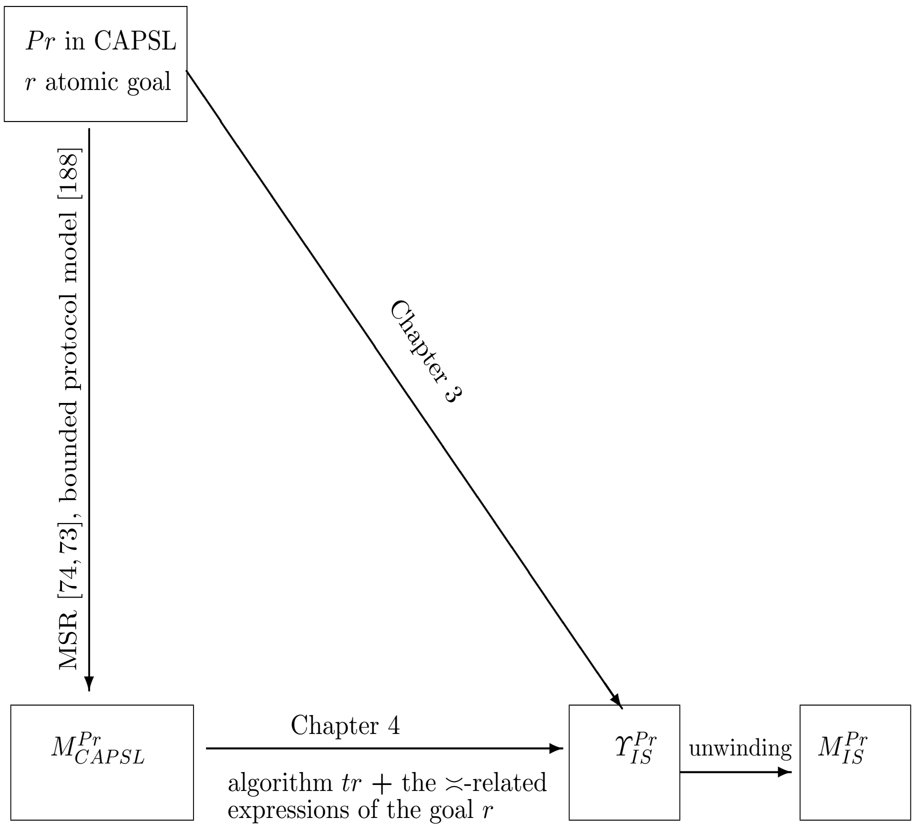
Therefore, by the above and (**hyp2**), we proved the inverse implication of this theorem.

Another possibility for $\rho_{CAPSL}(r) \asymp \rho_{IS}(r)$ is that of r being a goal of agreement on initials. This proof is similar to the one above. ∎

By Theorem 4.4.9 and Theorem 4.4.10 the aim of this chapter is attained. The latter shows that our MAS formalism is aligned in the satisfaction of traditional goals with a standard semantics for security protocols.

Another proof of preservation of satisfaction/refutation of $\asymp$-related formulations of atomic goals between the M^{Pr}_{CAPSL} and its M^{Pr}_{IS} image through algorithm tr is given in Appendix B. That proof is not based on homomorphisms between the underlying structures, but it inductively shows that the $\tilde{p}$ relation is preserved all along respective computations in the two models. Then, if algorithm $A1$ [188] is used in M_{CAPSL} to decide the satisfaction/refutation of an M_{CAPSL} formulation of a goal, the output of algorithm $A1$ holds for the $\asymp$-related Υ_{IS} formulation of the goal also.

Figure 4.9 outlines the important facts and results obtained about Υ_{IS}, our MAS-based model for security protocol verification.


Pr in CAPSL
r atomic goal
MSR [74, 73], bounded protocol model [188]
Chapter 3
M^{Pr}_{CAPSL}
Chapter 4
algorithm tr + the $\asymp$-related expressions of the goal r
Υ^{Pr}_{IS}
unwinding
M^{Pr}_{IS}

$$\begin{tikzcd}
M^{Pr}_{CAPSL} \arrow[rr, "\substack{\tilde{p}\text{ homomorphism}\\ \text{(Theorem 4.4.9)}}", "\substack{\tilde{p}\text{ preserved over computations}\\ \text{(Theorem B.3.1)}}"'] & & M^{Pr}_{IS}
\end{tikzcd}$$

$$\begin{tikzcd}
M^{Pr}_{CAPSL} \arrow[rr, dash, "\substack{M_{CAPSL}\text{ validates }\rho_{CAPSL}(r)\text{ iff }M_{IS}\text{ validates}\\ \text{all }f,\ f\in\rho_{IS}(r),\text{ with }\rho_{CAPSL}(r)\asymp\rho_{IS}(r)}", "\text{(Theorems 4.4.10 and B.3.2)}"'] & & M^{Pr}_{IS}
\end{tikzcd}$$

Figure 4.9 The Model M_{IS} and Traditional Semantics for Security Protocols

In Appendix B we use a relation slightly weaker than the $\tilde{p}$-relation to prove that the M^{Pr}_{CAPSL} model and the M^{Pr}_{IS} model are actually stuttering[4] equivalent [37]. This leads to an alternative way to showing the if M^{Pr}_{CAPSL} validates/refutes secrecy and/or agreement requirements of protocol Pr so does the tr-image model M^{Pr}_{IS}.

Re-evaluation of the M_{IS} Model

In Chapter 3, page 115, we evaluated the potential of our MAS model M_{IS}. In light of the results hereby presented, we reassess some of that evaluation and discuss the significance of this chapter.

In this chapter, by Theorem 4.4.10, we have proven that M^{Pr}_{CAPSL} validates secrecy and agreement requirements of protocol Pr if and only if the unwound, tr-correspondent model M^{Pr}_{IS} does so too. This ensures that the model we proposed in Chapter 3 is aligned with the mainstream semantics for security protocols with respect to the analysis of atomic `CAPSL` goals. This was possible partially due to Theorem 4.4.9 that showed the existence of an homomorphism between M_{CAPSL} and M_{IS}.

As an alternative, using the lemmas of this chapter, Theorem B.3.2 proves by structural induction on an arbitrary computation of M_{CAPSL} that if M^{Pr}_{CAPSL} validates/refutes secrecy and agreement requirements of protocol Pr so does the tr-correspondent model M^{Pr}_{IS}. Also, in Theorem B.4.3, the systems are proven to be stuttering equivalent. Figure 4.9 depicts these facts.

Given the guarantees of well-foundedness in our approach with respect to traditional `CAPSL` semantics, the next chapters will describe implementations of automatically generating the Υ_{IS} formalisation from a `CAPSL` protocol description. Further, this will enable us to show the practical use of our MAS formalism in analysing security protocols in traditional ways but also in novel settings driven by its temporal-epistemic foundations.

This chapter and Chapter 3 together also show that M_{IS} is more expressive that M_{CAPSL}, i.e., there many formulations of security requirements in Υ_{IS} which are novel and have no correspondence in the standard trace-semantics for protocol executions. In fact, some formulations of the `CAPSL` goals, e.g., the unlinkability property expressed in the $(S2)$ formulation of secrecy, can be rather likened to observational equivalence[5] properties used in security verification formalisms based on

[4]For a summary on stuttering equivalence, see Appendix B, page B.1.2.

[5]See Chapter 1, page 52.

process algebra. Whilst process algebra approaches to cryptography verification do not inherently refer to `CAPSL`-described protocol and are mostly used to verify classes of protocols other than authentication protocols, the relations between such formalisms and our MAS approach would still appeal to investigation. In fact, a proof of correspondence between certain epistemic formulation of security goals in our Υ_{IS} model and formulations based on observational equivalence in applied pi models is the subject of future line of work.

To sum up, some of the temporal-epistemic formulae expressing security requirements are validated in our MAS formalisation whenever their corresponding formulations are validated in trace-based semantics for `CAPSL`-described protocols (and vice-versa). These facts offer well-founded grounds to Chapter 5, where we present a toolkit for the automatic generation of Υ_{IS} MAS models from high-level `CAPSL` protocol descriptions.

Nevertheless, some of our temporal-epistemic formulae expressing security requirements seem to be more likened to applied pi [2] approaches (see Chapter 2 for notions on applied pi). These facts will be further discussed in Chapter 7.

Moreover, there are some Υ_{IS} formulations of security requirements which are completely novel and essentially epistemic, e.g., those involving group knowledge operators in Example 3.3.26. Chapter 6 will take such novel epistemic expressions of security properties further and show the new insight they bring into security verification and theoretical intrusion detection.

Chapter 5

`PD2IS` — An Attack-Finding Toolkit for Receiver-Transparent Protocols and Receiver-Transparent Reducible Protocols

Motto: "The value of an idea lies in the using of it."
(Thomas Edison)

In this chapter we model check multiagent system models for multi-session execution of authentication and key-establishment protocols against temporal-epistemic specifications of their requirements. The models and the specifications are formalised as per Chapter 3, i.e., Υ_{IS} and ρ_{IS} respectively. Their expression is given in `ISPL` *and it is verified with the* `MCMAS` *model checker. To achieve the above, we design and employ a model-generation toolkit called* `PD2IS`*. Its input is a* `CAPSL` *description for a receiver-transparent, authentication and key-establishment protocol Pr together with a user-defined protocol instantiation. The output of* `PD2IS` *consists of* `ISPL` *files describing the multiagent system model Υ_{IS}^{Pr} (as per Chapter 3) and/or other related models. In the resulting* `ISPL` *files,* `PD2IS` *also generates all the temporal-epistemic formulae included in the ρ_{IS} expressions of the* `CAPSL` *goals of protocol Pr. We use* `PD2IS` *to generate MAS models for the multi-*

session execution of several security protocols drawn from well-established repositories [49, 119]. As **PD2IS** *is linked to the model checker* **MCMAS**, *verification of the models is consequently performed. We report and discuss the verification results. The implementation of* **PD2IS** *follows the algorithm tr in Chapter 4; hence, the legitimacy of our tool is ensured by the proofs in Chapter 4. Part of the material in this chapter was presented in [27, 26].*

Section 5.1 introduces the notion of *receiver-transparent reducible protocols*. These are receiver-opaque protocols which can be easily converted into receiver-transparent protocols. Section 5.1 presents the procedures involved in this conversion and the implications they raise. In Chapter 3, the Υ_{IS} formalism was introduced to model only receiver-transparent protocols. By introducing receiver-transparent reducible protocols, Section 5.1 extends the applicability of the Υ_{IS} protocol-model to include also the class of receiver-opaque protocols reducible to receiver-transparent protocols. Section 5.2 presents a series of optimisations on the Υ_{IS} protocol-models. These optimisations increase the efficiency of the practical verification of such Υ_{IS} protocol-models. Therefore, we draw a taxonomy of increasingly efficient MAS protocol-models based on the original Υ_{IS} semantics. Following the ideas in algorithm *tr* in Chapter 4, we developed the PD2IS (*Protocol Descriptions to Interpreted Systems*) tool; it generates the aforementioned taxonomy of models as ISPL programs. Correctness guarantees for using PD2IS to verify MAS models for CAPSL-described security protocols are founded on the theoretical results presented in Chapter 4. Section 5.3 exposes the architectural and implementation details of PD2IS. Section 5.4 evaluates the performance of PD2IS and MCMAS. The criteria considered in the evaluation are:

- the efficiency of PD2IS in generating different optimised/(un-)optimised Υ_{IS} protocol-models;
- the significance of the results obtained by PD2IS and MCMAS, as an attack-finding and as a protocol verification toolkit; other performance parameters (e.g., state-space of unwound models, verification time, etc.) are also discussed here.
- the way in which PD2IS and MCMAS compare to other state-of-the-art tools in the field.

For ease of presentation, Table 5.1 provides a terminology-box containing all the distinct optimisations and/or classes of Υ_{IS} models that we use with PD2IS and that we discuss in this chapter.

Concept	**Notation**	**Explanation**
(loose) scenario	Sc	a partial protocol instantiation in which principals are mapped to participant/names (e.g., $A \rightarrow alice$, $A \rightarrow alice2$, $B \rightarrow bob$, $B \rightarrow greg$);
fixed scenario	$i, i \in \{1, \ldots n\}$	a partial protocol instantiation in which principals are mapped to participant/names; additionally, all communication partners are assigned (e.g., $(A, alice)$ "talks to" (B, bob), $(A, alice2)$ "talks to" $(B, greg)$, (B, bob) "talks to" $(A, alice2)$, etc.)
model for a loose scenario	Υ_{IS}	the IS-based model presented in Chapter 3
un-constrained model for a fixed scenario i	un-constrained $\Upsilon_{IS_{fixed}(i)}$	the IS-based model presented in Chapter 3 where the initial states are set so that they reflect the communication setting in the a fixed scenario i
constrained model for a fixed scenario i	constrained $\Upsilon_{IS_{fixed}(i)}$	the IS-based model presented in Chapter 3 where the initial states are set so that they reflect the communication setting in the a fixed scenario i and where the ranges of terms are restricted according to the possibilities allowed by the fixed communication setting in the scenario i

Table 5.1 `PD2IS`-driven Terminology Box

5.1 Receiver-Transparent Reducible Protocols

In Chapter 3 we defined the classes of receiver-transparent and receiver-opaque protocols. We introduced these notions driven by a need to systematise classes of protocols for which we can create bespoke, efficient models for IS-based verification. In this sense, the Υ_{IS} was introduced as a protocol-model for receiver-transparent protocols, i.e., protocols where the purported receivers can "peel off" all levels of encryption down to the atomic parts of the messages received (see Definition 3.1.3). However, in certain cases the Υ_{IS} protocol-model can also be applied to receiver-opaque protocols, i.e., protocols where the purported receivers might be unable to decrypt a message down to its atomic parts (see Definition 3.1.5). This will become clear by the notions exposed in the current section.

This section introduces the notion of *receiver-transparent reducible protocols*. These are receiver-opaque protocols which can be easily converted into receiver-transparent protocols. Thus, the Υ_{IS} protocol-model can be applied not only to RTP but also to ROP that are RT-reducible. The tool presented in Section 5.3 generates `ISPL` versions of the Υ_{IS} protocol-model for both RTP and ROP that are RT-reducible.

On the `CAPSL` descriptions of certain receiver-opaque protocols we can operate some syntactic modifications in order to transform the protocols into receiver-transparent protocols. Moreover, such modifications have negligible impact on the MAS denotations of the protocol executions, as the following will show.

To illustrate, first consider an excerpt of the description of the KSL [111] protocol:

Example 5.1.1 (An Excerpt of the Description of the KSL Protocol)

```
1.    A  ->  B  :   Na, A
2.    B  ->  S  :   Na, A, Nb, B
3.    S  ->  B  :   {Nb, A, Kab}Kbs, {Na, B, Kab}Kas
4.    B  ->  A  :   {Na, B, Kab}Kas, {Tb, A, Kab}Kbb, Nc, {Na}Kab
5.    A  ->  B  :   {Nc}Kab
```

KSL is a receiver-opaque protocol: in rule 3, S sends to B the composite $\{Na, B, Kab\}Kas$ that B cannot decrypt. However, by inspection of the protocol description, note that rules 3 and 4 can be rewritten as in Example 5.1.2.

Example 5.1.2 (A Modification to the Description of the KSL Protocol)

```
3.    S  ->  B  :   {Nb, A, Kab}Kbs
3'.   S  ->  A  :   {Na, B, Kab}Kas
4'.   B  ->  A  :   {Tb, A, Kab}Kbb, Nc, {Na}Kab
```

Through the transformation in Example 5.1.2, S sends $\{Na, B, Kab\}Kas$ directly to A, therefore B does not receive opaque composites from S any longer. Also, in the amended step $4'$, B does not need to send $\{Na, B, Kab\}Kas$ to A, as S already did send $\{Na, B, Kab\}Kas$ to A in step $3'$.

If the transformation in Example 5.1.2 is applied to the set of rules 1–5, then the new rules describe a protocol which is receiver-transparent. Hence, the original protocol is reducible to a receiver-transparent protocol, in the sense formalised below.

We now proceed to refine our original clusters of receiver-transparent and receiver-opaque. Definition 5.1.3 formalises the notion of opaque protocols reducible to a receiver-transparent protocol.

Definition 5.1.3 (RT Reducible Protocol) *A receiver-opaque protocol Pr is* reducible to a receiver-transparent protocol *(*RT reducible*) if for any $A \in Ho$, for all rules $r \in Rules^A$ of the form $r = i.B \rightarrow A : t_1, t_2$, where $i > first(Steps^A)$, $B \in Ho$ and there exists $t_2' \in composites(t_2)$ and $t_2'' = OSub(t_2')_{last(t_2')}$ such that $t_2'' \notin i\text{-}LearnedAtoms^A \cup OwnedAtoms^A$, it is the case that:*

1. *there exists $C \in Ho$ such that $t_2'' \in i\text{-}LearnedAtoms^C \cup OwnedAtoms^C$ and*
2. *there exists the rule $r' = j.A \rightarrow C : \overline{X}, t_2$, for some $j \in Steps, j > i$ and $\overline{X} \subseteq T$.*

To obtain the RT protocol correspondent to the RO protocol *Pr, the rules*

$$r = i.B \rightarrow A : \ t_1, t_2 \tag{5.1}$$

$$r' = j.A \rightarrow C : \overline{X}, t_2 \tag{5.2}$$

are rewritten to set of rules

$$r_{new} \quad = i.B \rightarrow A : t_1 \tag{5.3}$$

$$r_{interm} = i'.B \rightarrow C : t_2 \tag{5.4}$$

$$r'_{new} \quad = j.A \rightarrow C : \overline{X} \tag{5.5}$$

Definition 5.1.3 expresses the following. The RO protocol Pr is RT reducible if for any principal A that receives a pair (t_1, t_2) at step i such that A cannot analyse composite t_2 down to its atoms, there exists a principal C that will later receive t_2 from A and, moreover, C is able to analyse t_2 down to its atomic parts even at the current step i. Then, the sender B can send t_2 directly to

C (i.e., r_{interm}) and, at the later stage, A sends to C all of the originally intended terms, except the composite t_2 (i.e., $\overline{X}$).

Note that, in Definition 5.1.3, the replacement of rules denoted r, r' with their counterparts r_{new}, r_{interm}, r'_{new} is carried out for all opaquely received messages of the kind described. Therefore, by the definition of RT protocols, the obtained protocol is indeed an RT protocol. For chain-authentication protocols (e.g., the Otway-Bull protocol [39], shown in Example 3.1.4), the technique can be applied sequentially, until an RT protocol is obtained.

In an execution of the RT protocol obtained from Pr via the transformations in Definition 5.1.3, a *C-agent* possesses at step j the same atomic terms as it would possess at step j in an execution of the original RO protocol Pr. If a *B-agent* is able to construct the term $(t1, t2)$ at step i, by persistence of values for already set atoms, the *B-agent* is able to compose t_1 at step i and t_2 at the later, newly inserted step i'. For an *A-agent*, the actual setting of atomic terms does not change (i.e., in the execution of the original protocol, the agent would only set concrete values for atoms in the composite t_1, and this is also the case in the modified protocol). A similar discussion can be made with respect to the Dolev-Yao agent. His synthesis and analysis triggered by the term t_2 are delayed by one step (i.e., from i to i'). All the rest of the protocol semantics stays unchanged. Hence, the only amendments to be made refer to steps.

Note that since the *A-role* was not able to analyse the composite t_2 down to its atomic parts, the intruder could have inserted a type-flawed term at step i. By the modifications explained, the *A-role* participant is no longer purported to receive t_2 at step i. In this sense, such reductions can lead to eluding some type-flaw attacks[1]. Nevertheless, given the discussions above, the intruder could still insert that type-flawed term at step i'. The attack mounted in this fashion would however have a different trace. Furthermore, the intruder can manipulate the newly introduced step i' to mount an attack where he impersonates B, in a way that was not possible in the original RO protocol Pr. With `PD2IS`, we verify ROP which are reducible to RTP. In the light of the above, if an attack is found on an RO protocol Pr by verification of the RT correspondent protocol of Pr, a trace-inspection of the attack counterexample shown could be used to place the results in the context of the original RO protocol Pr.

[1]See Chapter 2, page 35.

We do not apply similar techniques to cases where the opaque term t_2 is within a nested encryption (e.g., $B \rightarrow A : \{t_1, t_2\}_{k_A}$). A protocol with this kind of term-nesting is the Woo-Lam protocol (shown in Example 2.2.2). In such cases, it would be hard to evaluate the preservation of the original denotation of the protocol-executions after reducing it to an RT protocol. The class of such receiver-opaque protocols which do not reduce to receiver-transparent protocols by the method exposed in Definition 5.1.3 are discussed in Chapter 7 and in Appendix C.

In this section we have presented a systematic way to modify the descriptions of certain RO protocols to render RT protocols. These modifications will allow us to verify ROP using our Υ_{IS} model for RTP. However, these amendments of `CAPSL` descriptions imply certain changes in the meaning of the protocols' execution. In that sense, we only apply RT-reduction to RO protocols where these implied changes appear negligible. In Chapter 7 we will verify ROP by using bespoke models for RO protocols. Our practice of verifying ROP both in this fashion and with such bespoke models for ROP has indeed shown attack-trace modifications in the case of RT-reduced models. In the same time, the verification parameters (e.g., size of state-space, verification time) exhibited by a Υ_{IS}^{Pr} model for a RT-reducible protocol Pr were shown to be much smaller that those exposed by a model for the un-modified ROP Pr. Chapter 7 and Appendix C detail this issue. In that sense, it is motivating to verify ROP which are RT-reducible protocols using the Υ_{IS} model for RTP. All things considered, the tool presented in the following sections deals with RT protocols and with RO protocols which are RT reducible.

5.2 Optimised Υ_{IS} Models for Protocols Scenarios

This section presents the optimisations we bring to the Υ_{IS} model in order to maximise the model checking performance. Advantages and disadvantages to these techniques will be discussed. The tool presented in the following section generates IS-based models following all these optimisations.

5.2.1 Fixed Protocol Scenarios

In this section we introduce the notion of *fixed protocol scenario*, e.g., a protocol instantiation that explicitly designates the communication partners. This section also gives the initial motivation that fixing protocol scenarios is a path to follow in order to optimise Υ_{IS} for practical model checking purposes.

In Chapter 2 we summarised that there are two dichotomous approaches to formal-methods verification of security protocols. One assumes a bounded number of protocol-sessions under analysis, whereas the other assumes an unbounded such number. The first assumption is the most prevalent, as a bounded number of protocol-sessions usually entails the decidability of several security-related problems [188, 175]. Unfortunately, computation traces present in an unbounded protocol model might be eluded in the corresponding protocol model for a bounded number of protocol sessions. However, practice has proven [138, 178] that protocol attacks are usually exhibited on models for few concurrent sessions. An example of such practice is the successful use of Lowe's small systems [138] (recalled in Chapter 2, page 48). In Chapter 2, page 33, we explained the notion of protocol scenario: a number of protocol sessions described by the participant-names instantiating the principals, i.e., $(A : alice1)$, $(A : alice2)$, $(B : bob1)$, $(B : bob2)$, etc. Approaches assuming a bounded number of protocol-sessions [138, 9, 19, 163, 181, 53] often fix the communication partners implied by a scenario. Let Sc be the aforementioned scenario: $(A : alice1)$, $(A : alice2)$, $(B : bob1)$, $(B : bob2)$, etc. Stating that, e.g., $(A : alice1)$ "talks to" $(B : bob1)$, $(B : bob1)$ "talks to" $(A : alice2)$, $(A : alice2)$ "talks to" $(B : bob2)$, etc., denotes fixing the communication partners in the scenario Sc. By a *fixed protocol scenario* we mean a fixed communication setting of this kind. By contrast, a *loose scenario* or, simply, a *scenario* denotes an instantiation of the principals with participant-names where the communication partners are not fixed.

We will now give an intuition into why models for loose scenarios can prove to be computationally expensive when model checked. In Chapter 3, the Υ_{IS}^{Pr} model formalises the execution for a loose scenario for Pr, where Pr is a `CAPSL` described RT protocol. In this model the number of role-instantiations considered is bounded, but it is usually exponential in the size of the protocol. Consequently, the Υ_{IS}^{Pr} formalism uses very large ranges for the terms (see Chapter 4, page 123, for

$min\_size(\mathcal{R}_X)$ for a sort X). Since Υ_{IS}^{Pr} describes a loose scenario, the initial states are not fixed with respect to some pre-established communication setting (e.g., in the possible initial states of any A-*agent* ag_A^{σ}, the variable $ag_A^{\sigma}.PartnerB$ ranges over the entire set $\mathcal{R}_{Ho\cup\mathbf{I}}$). Therefore, the size of the initial-state space in Υ_{IS}^{Pr} can be very large (even for a small size protocol). It follows that model checking the Υ_{IS}^{Pr} system, given in Chapter 3, can often be infeasible in practice.

We will now suggest the main reason for which models for fixed scenarios perform better than models for loose scenarios in the practice of model checking. Let i denote a fixed scenario for a protocol Pr and $\Upsilon_{IS_{fixed}(i)}^{Pr}$ denote the MAS model for its corresponding execution. An $\Upsilon_{IS_{fixed}(i)}^{Pr}$ model has the same semantics as Υ_{IS}^{Pr}, except for the fact that the initial states of $\Upsilon_{IS_{fixed}(i)}^{Pr}$ will be fixed according to the initial communication setting in the scenario i. Therefore, the number of possible initial states in $\Upsilon_{IS_{fixed}(i)}^{Pr}$ is exponentially smaller than the number of possible initial states of Υ_{IS}^{Pr} (i.e., the decrease is by a factor of $C_{|Ag|}^2$). It follows that the total number of reachable states of $\Upsilon_{IS_{fixed}(i)}^{Pr}$ is much smaller than that of Υ_{IS}^{Pr}. Consequently, model checking a $\Upsilon_{IS_{fixed}(i)}^{Pr}$ model for a fixed scenario i of Pr is more efficient than model checking a large Υ_{IS}^{Pr} model. Nevertheless, there are certain trade-offs implied by model checking a $\Upsilon_{IS_{fixed}(i)}^{Pr}$ model for a fixed scenario i of Pr instead of model checking the Υ_{IS}^{Pr} model; these will be presented in Section 5.2.3.

To optimise the model checking security protocol with `MCMAS`, we often use models for fixed scenarios. The tool presented in the following sections generates models for both fixed and loose scenarios. However, there is a number N of fixed scenarios of Pr which would describe all the communication patterns implied by a loose scenario of Pr (i.e., $N > 1$). Therefore, a reasonable counterpart of the analysis of a Υ_{IS}^{Pr} model for a loose scenario would be to verify all $\Upsilon_{IS_{fixed}(i)}^{Pr}$ models for a fixed scenario i of Pr, for all $i \in \{1, \ldots, N\}$. Thus, in the following, Section 5.2.2 will investigate how to enumerate such N fixed scenarios needed to counterbalance the possible communication patterns implied by a loose scenario. The tool presented in the following sections implements the techniques to be presented in Section 5.2.2.

5.2.2 Enumerating the Fixed Scenarios for a Protocol Pr

We begin with estimating the number of all possible fixed scenarios spawned by our method of instantiating protocols. Let $A \in Ho$ be an arbitrarily fixed principal in the signature/description of Pr. Assume that $\mathcal{R}_{Ho\cup\mathbf{I}} = \{name_1, \ldots, name_k\}$ is the range for the names to be given to the principals of Pr and to the intruder, under a set of role-substitutions Σ_{Pr}. A participant representing A would have $|Ho| - 1$ communication partners (i.e., we assume the habitual case where any role is engaged with all the other roles). Then, there are $|\mathcal{R}_{Ho\cup\mathbf{I}}|^{|Ho|-1}$ possibilities to designate the communication partners of the A-role participant. Since a participant of the *A-role* could have any name in $\mathcal{R}_{Ho\cup\mathbf{I}}$ and since A was drawn arbitrarily from Ho, it follows that there are $p = |\mathcal{R}_{Ho\cup\mathbf{I}}| \times |Ho| \times |\mathcal{R}_{Ho\cup\mathbf{I}}|^{|Ho|-1}$ communication settings to be chosen in this way. We draw these from the entire space of $\mathcal{R}_{Ho\cup\mathbf{I}}$ and $|\mathcal{R}_{Ho\cup\mathbf{I}}|$=$k$. Therefore, there are C_k^p fixed scenarios to be considered in this fashion (starting from a `CAPSL` description of Pr and a set of substitutions mapping the principals over $\mathcal{R}_{Ho\cup\mathbf{I}}$).

In the set of C_k^p fixed scenarios, we have included some that are equivalent up to the renaming of the communication partners, i.e., they are symmetric. For instance, if in a fixed scenario Sc we uniformly interchange $name_i \in \mathcal{R}_{Ho\cup\mathbf{I}}$ with $name_j \in \mathcal{R}_{Ho\cup\mathbf{I}}$, then the scenario obtained is equivalent up to this renaming to the original scenario Sc. Observe that such an interchange corresponds to a transposition in the group of permutations over the set $\mathcal{R}_{Ho\cup\mathbf{I}}$. For the set of fixed scenarios that are equivalent up to a transposition or a product of transpositions over $\mathcal{R}_{Ho\cup\mathbf{I}}$, we need to count just one scenario (i.e., a representative of the class). If we do so, we need to re-evaluate how many fixed scenarios for Pr there are to be considered.

In general, a group G acts on a set X by applying an internal operation to its elements. For example, the permutation group $G_1 = \{(abcd), (bacd), (abdc), (badc)\}$ operates on the elements a and b over the set $\{a, b\}$ and on the elements c and d over the set $\{c, d\}$. Formally, a group can operate on any particular element x over a fixed set. This set is called the *orbit* of the element. In this context, we recall the *Cauchy-Frobenius lemma* which gives a method of counting orbits in an arbitrary group G. Let G be a finite group that acts on a set X. For each $g \in G$, let X^g denote the set of elements in X that are fixed by g. Then, the Cauchy-Frobenius lemma gives the following formula for finding the number of orbits, denoted $|X/G|$, as: $|X/G| = \frac{1}{|G|}\sum\limits_{g\in G}|X^g|$.

To consider only the set of symmetry-free fixed scenarios, we are then looking for the number of orbits in the group of transpositions over the set $R_{Ho\cup\mathbf{I}}$. There are $C_2^{|\mathcal{R}_{Ho\cup\mathbf{I}}|}$ elements of this group, hence $|G|$=$C_2^{|\mathcal{R}_{Ho\cup\mathbf{I}}|} = C_2^k$. If one such transposition is fixed then the total of $p = |\mathcal{R}_{Ho\cup\mathbf{I}}| \times |Ho| \times$

$|\mathcal{R}_{Ho \cup \mathbf{I}}|^{|Ho|-1}$ possibilities of choosing the names of the communication partners drops, by a factor of C_2^k, from p to p'. Then, the number of resulting scenarios is $C_{k/2}^{p'}$. By Cauchy-Frobenius lemma applied to the group of transpositions over the set $R_{Ho \cup \mathbf{I}}$, it follows that $N = \frac{\sum_{g \text{transposition}} \alpha_k \cdot C_{k/2}^{p'}}{C_2^k}$, $\alpha_k = 0$ if k is odd and $\alpha_k = 1$ if k is even.

If we allow for symmetry, the number N of fixed scenarios can very big indeed. For a protocol with two roles, e.g., $|Ho| = 2$, and considering only two names for the participants, e.g., $k = 2$, there are $N{=}C_k^p{=}C_2^{3 \cdot 2 \cdot 3^3}{=}66$ fixed scenarios. Out of these N scenarios, some are symmetric. Instead if we select only the symmetry-free set of fixed scenarios, we only need to consider 36 fixed scenarios. However, if we raise the range of names to $k = 3$ and keep $|Ho| = 2$, then the number of fixed, non-symmetric scenarios reaches the order of $3 \cdot 10^4$.

In the first case of $N = C_k^p$ scenarios, we also counted scenarios in which, e.g., *alice* as an A-role "talks to" *alice* as a B-role, for $A \neq B$ (i.e., *alice* "talks to" herself). To capture certain attacks, we will not dismiss this kind of communication setting.

The above shows how many fixed scenarios are needed to cover the space of scenarios over $\mathcal{R}_{Ho \cup \mathbf{I}}$. In the rest of the section, we discuss the case of fixed scenarios covering other ranges for the participant-names.

Note that part of the space for the fixed scenarios can be pruned by explicitly giving Σ_{Pr}. For instance, if $ag_1.id$ is already fixed to $\sigma(A)$, for some $\sigma \in \Sigma_{Pr}$, then the number p drops by a factor of $|\mathcal{R}_{Ho \cup \mathbb{I}}|$. Another way to reduce number of the fixed scenarios is to enforce the fact that each particular role is mapped into a different range (e.g., for any $X, Y \in Ho$, $X \neq Y$, $\mathcal{R}_X \neq \mathcal{R}_Y$). These respective ranges need not necessarily be disjoint, but each should restrict the domain over which some principal A ranges (i.e., it is to be enforced that $|\mathcal{R}_A| < |\mathcal{R}_{Ho \cup \mathbf{I}}|$). Such a restriction is reasonable: e.g., the servers of a Swiss bank will have IP ranges different from those of their American clients.

Enumerating all symmetry-free fixed scenarios means generating all symmetry-free combinations over $\mathcal{R}_{role_1} \times \ldots \times \mathcal{R}_{role_n}$ (irrespective of whether $\mathcal{R}_{role_i}$ is $\mathcal{R}_{Ho \cup \mathbf{I}}$ or a more restricted range). Given the set $\mathcal{R}_{role_i}$ for each $i = \overline{1, n}$, a trivial way to achieve the above is to generate the generalised Cartesian product $\mathcal{R}_{role_1} \times \ldots \times \mathcal{R}_{role_n}$ and then eliminate the ordered n-tuples that are symmetric.

In the tool presented in the following section, we implement all the aforementioned methods of generating fixed scenarios. The procedures to generate all ordered n-tuples and all symmetry-free ordered n-tuples are presented in Appendix A, Section A.2.

Other reduction techniques for protocol models, looking at the symmetry of the actual protocol data, have been recently advanced [56]. We differentiate ourselves from those in that we hereby discuss how protocol scenarios impact the size of the protocol models and we do not make an in-depth enquiry into the symmetry of the protocol.

A more generic method (un-tailored for a specific protocol model) for calculating the number of possible protocol scenarios is presented in [62].

The previous section has explained why fixed scenarios are preferable to loose scenarios when modelling protocol for practical verification. In this section we have investigated how many fixed scenarios spawn a protocol instantiation. We have evaluated how many of these are symmetry-free, i.e., worth analysing in practice. At a high-level, we have presented the methods of enumerating/generating all such scenarios. The tool we use for model-generation, `PD2IS`, implements these methods. Section 5.3 will detail this. In the following section, we proceed to explain the following: *1)* how to modify the Υ_{IS} model for loose scenarios to encode fixed protocol scenarios; *2)* how to constrain these models in order to increase the performance of model checking these models; *3)* which are the trade-offs implied by the aforementioned, practice-driven constraints.

5.2.3 Constrained Υ_{IS} Models for Fixed Scenarios

In this section we show means of increasing the efficiency of the verification process by applying further constraints to the Υ_{IS} models for fixed scenarios. The resulting trade-offs are also reported. The tool we developed to generate of protocol-models produced all flavours of models constrained in the fashion presented in this section.

Let $\Upsilon^{Pr}_{IS_{fixed(i)}}$ be an Υ^{Pr}_{IS} model for a fixed scenarios i of a given protocol Pr, for $i \in \{1, \dots, N\}$. As before, let $\Upsilon^{Pr}_{IS_{fixed(i)}}$ be identical to Υ^{Pr}_{IS} except for the fixing of the communication partners in the initial states of $\Upsilon^{Pr}_{IS_{fixed(i)}}$ according to the scenario i. We will refer to such models as *unconstrained Υ_{IS} for fixed protocol scenarios.* Then, the parallel composition of verifying all $\Upsilon^{Pr}_{IS_{fixed(i)}}$, i.e., for all $i \in \{1, \ldots, N\}$, will cover the same state-space of the original, general $\varUpsilon_{IS}^{Pr}$. However, some generality is lost. For instance, some formulae given by the expression $\rho_{IS}(p)$ will be trivially validated in $\varUpsilon_{IS_{fixed(i)}}^{Pr}$ (e.g., the formulation for aliveness of *bob* if we fix $j.PartnerB = bob$ in the initial states of agent j). Furthermore, by using models for fixed scenarios, the results of verifying $\varUpsilon_{IS_{fixed(i)}}^{Pr}$ against the distributed knowledge of all agents with the same identity will not usually imply the same results holding on $\varUpsilon_{IS}^{Pr}$ (i.e., in some formulae in Section 3.3.3, page 115, D_{Gr} is used inside formulations of goals, where Gr is the group of all agents ag with the same $ag.id$).

Verifying $\varUpsilon_{IS_{fixed(i)}}^{Pr}$ is more efficient than verifying $\varUpsilon_{IS}^{Pr}$. However, we already exemplified some of the compromises in sequentially verifying all $\varUpsilon_{IS_{fixed(i)}}^{Pr}$. If we are ready to accept such trade-offs, we can optimise our models further. As we will show in the following, these optimisations will improve the performance of the model checking and will still find protocol attacks. These optimisations imply more constraints on the models for fixed scenarios and bring us closer to analysing MAS homologues of Lowe's small systems. For instance, consider each $\varUpsilon_{IS_{fixed(i)}}^{Pr}$ to be an $\varUpsilon_{IS}^{Pr}$ model, not only its initial states fixed, but obtained under a smaller instantiation. More precisely, if in the scenario i, a series of participants are not part of the communication, all of their role-substitutions can be dismissed. Then, the ranges of atomic-terms, the ranges of messages, the set of possible actions, etc., become smaller. Hence, the reachable state space of $\varUpsilon_{IS_{fixed(i)}}^{Pr}$ becomes smaller. We will expose further optimisations along these lines. For illustration purposes, let $\varUpsilon_{IS}^{NSPK}$ denote the $\varUpsilon_{IS}$ model for $NSPK$. In $\varUpsilon_{IS}^{NSPK}$, $ag_A.N_B$ ranges over $\mathcal{R}_{\mathcal{N}}$ (i.e., since $N_B \in LearnedAtoms^A$, ag_A could accept any value-nonce for N_B from any *B-role* player). Now, let i be a fixed scenario for an execution of the NSPK protocol that stipulates an *A-role* participant *alice* "talking to" a *B-role* participant *bob*. Then, in the initial states of $\varUpsilon_{IS_{fixed(i)}}^{NSPK}$, the assignment $ag_A.PartnerB := bob$ is declared, where $ag_B.id = bob$ for some ag_B. Therefore, in $\varUpsilon_{IS_{fixed(i)}}^{NSPK}$ we can restrict the range of $ag_A.N_B$ from $\mathcal{R}_{\mathcal{N}}$ to $\{ag_B.N_B\} \cup \sigma_{\mathbf{I}}(\mathcal{N})$. This is reasonable, as in the fixed communication setting above, ag_A can only receives values for N_B from ag_B or possibly from the intruder. Given a fixed scenario, its $\varUpsilon_{IS}$ model restricted in this way is denoted as the *constrained $\varUpsilon_{IS}$ model.*

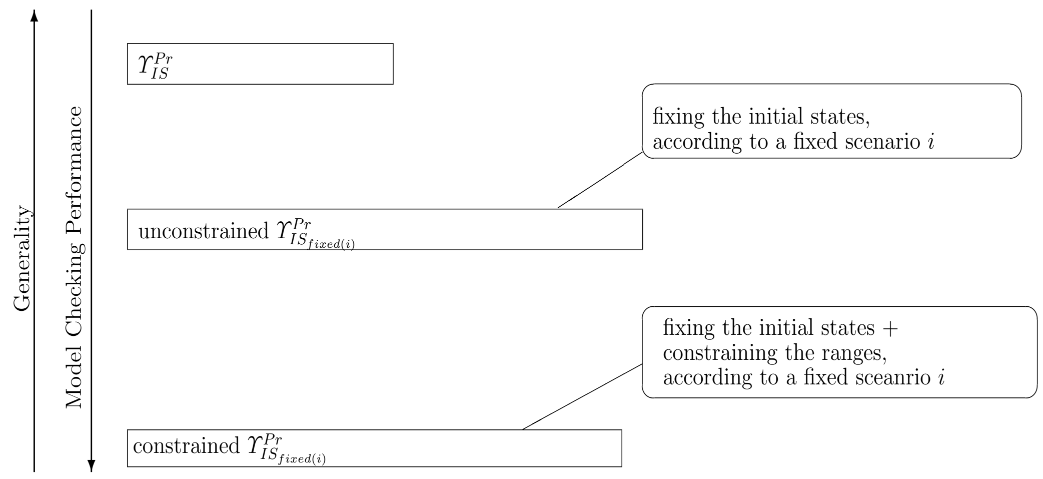
Let φ be a generic CTLK formula expressed through ρ_{IS} in $\varUpsilon_{IS}^{Pr}$. It is possible that the formula φ does not make sense in the constrained $\varUpsilon_{IS_{fixed(i)}}^{Pr}$ model for some fixed scenario i (i.e., due to the pruning of the state-space of $\varUpsilon_{IS}^{Pr}$, it can be that φ is not well-founded with respect to the smaller set of variables and their respective ranges employed in $\Upsilon^{Pr}_{IS_{fixed}(i)}$). If φ is well-founded in $\Upsilon^{Pr}_{IS_{fixed}(i)}$, then the aforementioned trade-offs in verifying φ are applicable in this case too. The next section will show a partial remedy to this issue.

We have suggested how verifying different flavours of $\Upsilon^{Pr}_{IS_{fixed}(i)}$ can be more efficient than verifying Υ^{Pr}_{IS}. However, the number N of fixed scenarios is exponential in the size of the protocol Pr. Then, verifying all N models $\Upsilon^{Pr}_{IS_{fixed}(i)}$, for all $i = \overline{1, N}$, remains expensive. To diminish the number of systems to verify, we can sequentially verify models corresponding to increasingly larger scenarios until an attack is found. For instance, we start with a set of constrained models, each corresponding to a fixed, non-symmetric scenario over a set of participant-names of size l. If no attack is found on any of the models for scenarios of size l, we then verify each constrained model corresponding to a fixed non-symmetric scenarios over a set of names of size $l + 1$, etc. We start with a size l that is large enough to ensure that initially the agents do not know the communication schema (see Chapter 4, page 124). In general, if $|Ho| = 2$ then the size of the range $\mathcal{R}_{Ho \cup \mathbf{I}}$ should be at least 4 (i.e., one name for each honest principal, one name for the intruder and one name for obfuscating the initial setup, such that some ag_A does not trivially know the identity of the intruder).

For invariant properties[2] expressed in CTLK, we can try to optimise the protocol analysis without trading off on the verification of knowledge properties as above. This alternative method is to verify the large Υ^{Pr}_{IS} models for a loose scenario of Pr by using a distributed version of `MCMAS` [108]. This branch of `MCMAS` will parallelise the model Υ^{Pr}_{IS} by using criteria other than the spawning of sub-models for all fixed scenarios. Then, it will concurrently verify each such parallel share. In later sections, we will report on using this method to analyse Υ^{Pr}_{IS} models for loose scenarios of Pr.

To conclude, in this section we have presented means of optimising the Υ_{IS} model in order to maximise the model checking performance. The Υ_{IS} formalism introduced in Chapter 3 mirrored a model for loose protocol scenarios (i.e., protocols where principals are instantiated to a potentially exponential number of participants). In turn, this section has advanced un-constrained models for fixed scenarios and constrained models for fixed scenarios. Figure 5.1 synthesises the discussions above, i.e., going from general to specific in modelling security protocols executions in a MAS set-

[2]An invariant property expressed in CTLK has the modality AG at the root of the parse tree.



Generality
Model Checking Performance
Υ_{IS}^{Pr}
fixing the initial states, according to a fixed scenario i
unconstrained $\Upsilon_{IS_{fixed(i)}}^{Pr}$
fixing the initial states + constraining the ranges, according to a fixed sceanrio i
constrained $\Upsilon_{IS_{fixed(i)}}^{Pr}$

Figure 5.1 General and Optimised Υ_{IS} Models Generated by `PD2IS`

ting. Our model-generation tool, `PD2IS`, produces all these flavours of protocol-models, i.e., the Υ_{IS} formalisation for loose protocol scenarios, plus all the implied constrained and un-constrained models for fixed protocol scenarios. With `MCMAS`, we verify all the corresponding `ISPL` files or a part of them until an attack is found. Amongst other performance evaluations, the next sections will compare the results of generating/analysing the most specific with the results of generating/analysing the most general of these models.

5.3 Automatic Compilation of Protocol Scenarios into Interpreted Systems

In this section we detail on the toolkit `PD2IS` (`P`rotocol `D`escriptions to `I`nterpreted `S`ystems) which we developed to generate MAS models upon the semantics described in Chapter 3 and MAS models optimised as discussed in Section 5.2. In brief, our toolkit generates the Υ_{IS} model for a loose scenario of some protocol Pr, the constrained and unconstrained $\Upsilon_{IS_{fixed(i)}}^{Pr}$ models for all fixed scenarios i of protocol Pr. The protocol Pr is a receiver-transparent or a receiver-transparent reducible protocol, given in `CAPSL`. `PD2IS` produces the aforementioned IS-based models as `ISPL` programs. The section proceeds as follows: *1)* some high level details about the characteristics of

PD2IS are given; *2)* details of the architecture and implementation of PD2IS are provided. The following section evaluates the methodology of verifying protocols with PD2IS and MCMAS.

The input to PD2IS is a file designating a CAPSL protocol description together with some additional parameters. These parameters describe either a loose protocol scenario or a fixed protocol scenario. Let us consider first the case where the input describes a loose protocol scenario. Then, the input contains the principal instantiations (e.g., participant-names for instantiating the principals are given) and some bounds on the size of the scenario to be considered (e.g., the maximum number of A-agents, etc.). In this case, PD2IS can generate the ISPL file for a generic $\varUpsilon_{IS}^{Pr}$ as per Chapter 3 (i.e., calculating all minimal ranges, etc.). Alternatively, PD2IS can generate all possible constrained and un-constrained $\varUpsilon_{IS_{fixed}}^{Pr}$ models for the fixed scenarios spawning the loose scenario. Let us consider now the case where the input describes a fixed scenario (i.e., not only a principal-instantiation, but also a fixed, initial communication setting). Then, PD2IS will then generate the constrained and the un-constrained $\varUpsilon_{IS_{fixed}}^{Pr}$ models for the fixed scenarios.

PD2IS systematically generates the set of propositions and formulae corresponding to the ρ_{IS} expressions of the CAPSL goals. PD2IS automatically inserts these temporal-epistemic formulae in the ISPL file for the model under generation. MCMAS is called for each ISPL file produced by PD2IS. MCMAS returns the calls either by certifying that the specifications are satisfied or by returning detailed counterexamples. These are used by PD2IS to report details of the attack found on the protocol (i.e., the failure of one or more of formulae corresponding to the goals). We proceed with details about the architecture and implementation of PD2IS.

5.3.1 PD2IS: Architecture

From an architectural point of view PD2IS comprises six modules: `formalisation`, `parser`, `unmarshaller`, `data-setup`, `producer` and `utils`. The architecture of PD2IS is shown in Figure 5.2; it follows the Model-View-Controller (MVC) design pattern [88] as sketched below:

1. processedInput $\leftarrow$ Formalisation(userInput)

2. resultsDataAccess $\leftarrow$ Parser(processedInput);

3. Unmarshaller.unmarshall(resultsDataAccess);

4. resultsFromBusinessLogic ← DataSetup.setup(...);

5. isplOutput ← Producer(resultsFromBusinessLogic)

The module `formalisation` is composed of two sub-modules: the `description` sub-module and the `scenario-generator` sub-module. The `description` sub-module consists of a collection of `XML` schemas (i.e, `XSD` files) that encode the protocol signature and the term algebra as given in Chapter 3 (i.e., variables and their ranges, typed and untyped messages, principals, and goals for any generic RT protocol). The input is given through an `XML` file designating the `CAPSL` description to be parsed and describing a scenario. In Section 5.3.2 we give an example of the `XML` schema for an atomic term of a signature/description and an example of such an `XML` input file. The routines in the `scenario-generator` sub-module scan part of the description file and input parameters and generate the data structures for (all) the (fixed) scenarios and the formulae to be checked. The `parser` module scans the `CAPSL` description and, using information provided by the `scenario-generator`, it populates more detailed data structures. At the same time, it outputs (i.e., marshals) `XML` files compliant with the schemas provided for any protocol signature. These `XML` files describe at a lower level the previously generated scenarios and the specifications to be generated/verified. More precisely, abstract, succinct representations of instantiated roles, messages, atoms etc., are given through these `XML` files. They draw the relation between a protocol scenario and a protocol-execution model. An example of such an `XML` file is provided in Section 5.3.2. The modules above, from the point of view of the MVC pattern, are comprised in the *controller* part of the application.

The `unmarshaller` module then converts (i.e., unmarshals) the `XML` files generated by the `parser` module into `JAVA` objects and populates the data structures describing the semantics to be adopted. Finally, the `data-setup` module processes all these data structures in order to produce the full IS semantics described in Chapter 3. Thus, it is the `data-setup` module that implements most of the systematisation behind the translation from a standard protocol model into our multiagent system formalism (i.e., the algorithm *tr* in Chapter 4). As far as the MVC pattern is concerned, the `unmarshaller` and the `data-setup` modules constitute the *model* part of the application.

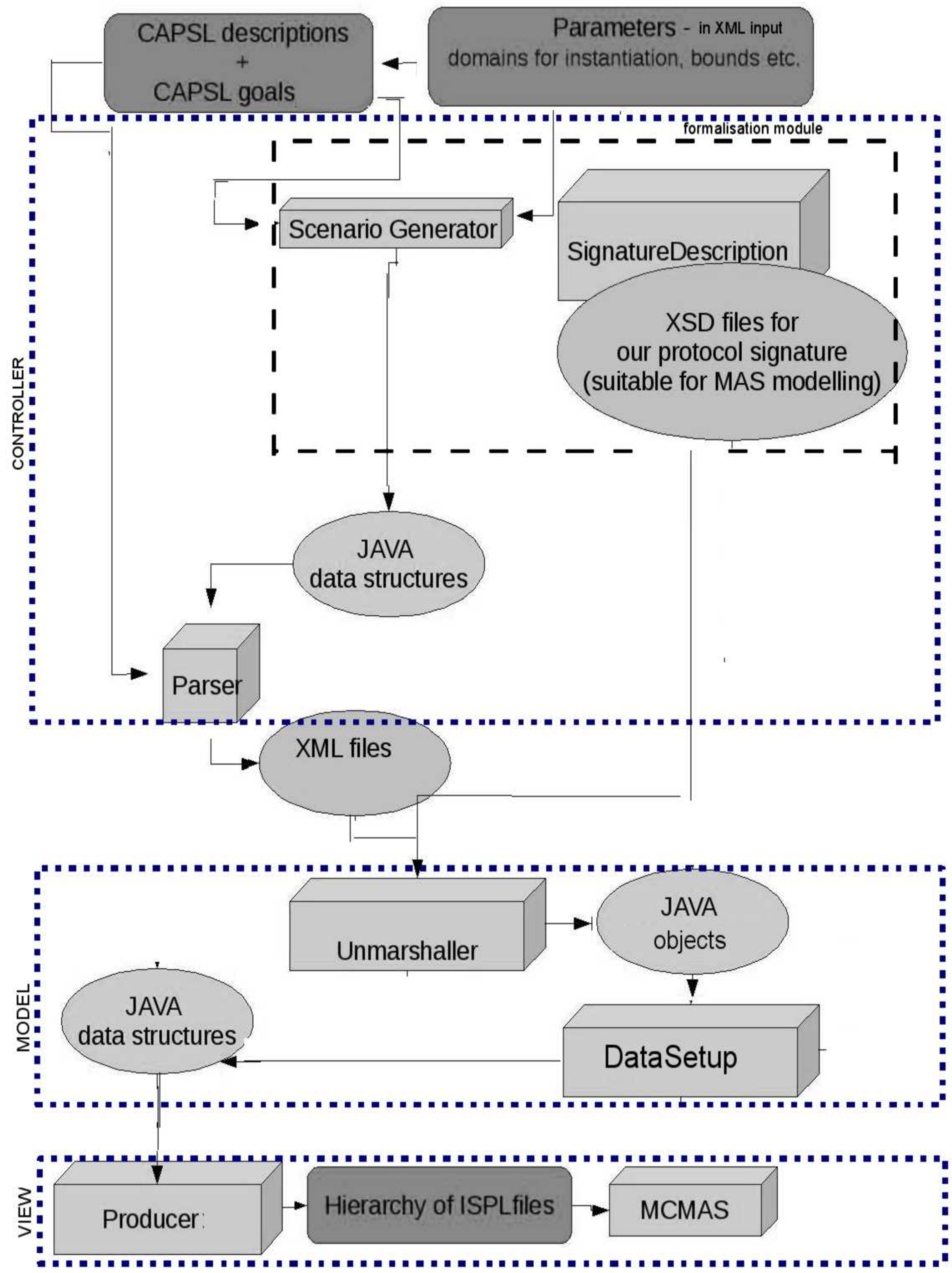


Figure 5.2 The MVC Architecture of PD2IS

Then, the `producer` module simply outputs the resulting structures into the `ISPL` format (i.e., it is the *view* part of the MVC design).

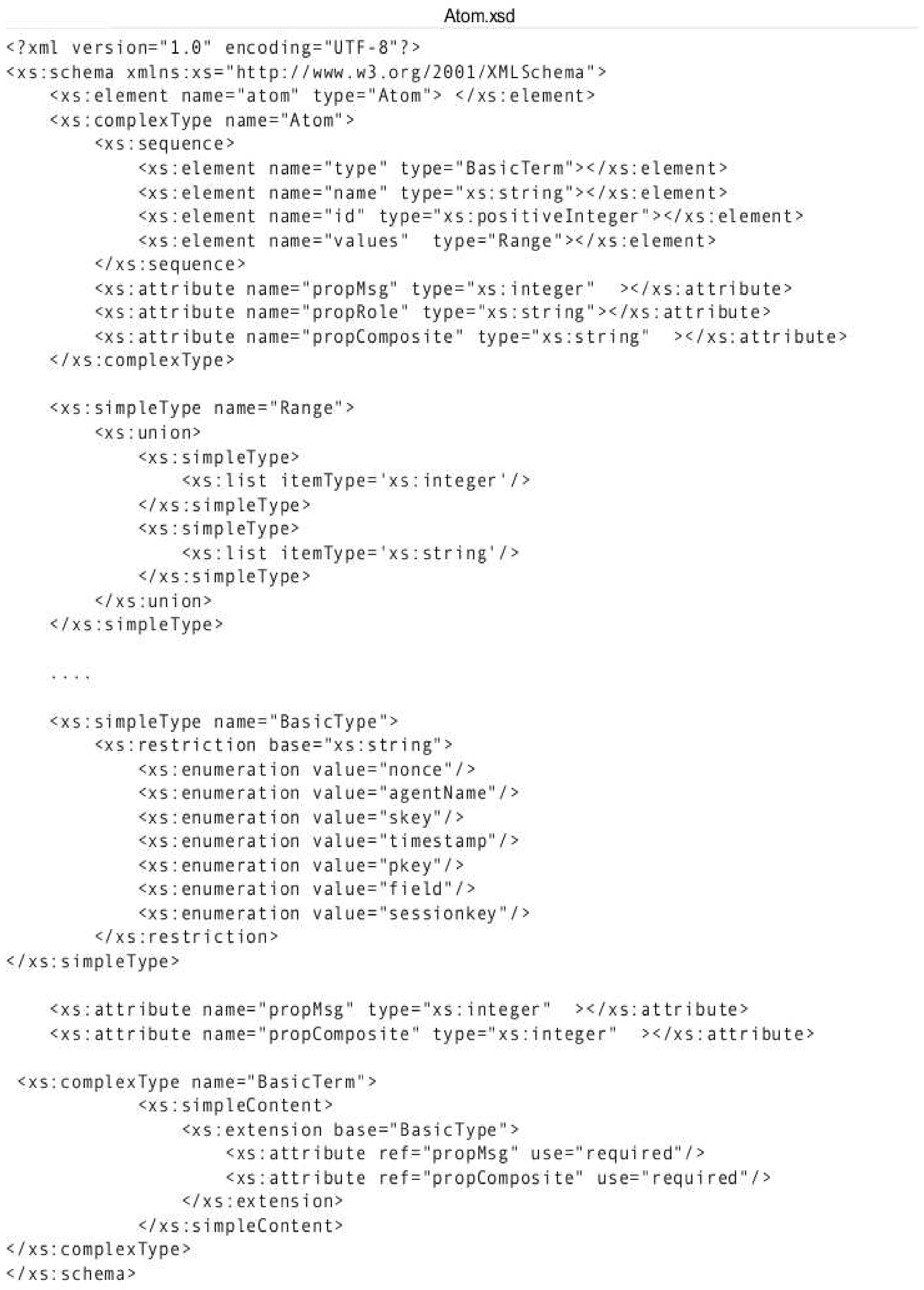
The `utils` module contains classes which are helpers to the implementation of the other modules.

By using the MVC pattern in the architecture of `PD2IS`, we created an expandable and reusable platform. For instance, we use an intermediate format to describe the semantics of the models under generation, i.e., the `XML` files generated. By implementing different `data-setup` and `producer` modules we can reuse the information in these `XML` files to compile into different protocol semantics and different low-level languages (other than our MAS-based semantics and `ISPL`, respectively). Conversely, by replacing the `parser` module and keeping the rest in place, we can get a method of compiling IS models into `ISPL` starting from security protocols expressed in other high-level languages but `CAPSL` (e.g., `HLSPL`). The architecture of `PD2IS` is decoupled not only in an MVC style, but also such that it mirrors the methodology of generating Υ_{IS} model as described in Chapter 3 and Chapter 4 (i.e., algorithm tr).

5.3.2 `PD2IS`: Implementation Details

`PD2IS` is coded in JAVA 5 and it is available open-source at [28]. We now present the implementation details behind each module of `PD2IS`.

The `description` sub-module contains `XML` schemas (i.e., `XSD` files) underlying a signature similar to the one presented in Chapter 3. In `PD2IS`, we are dealing with a larger signature than the signature $\mathcal{S}$ used in Chapter 3 and Chapter 4. In that respect, the set S of sorts used in `PD2IS` is extended to contain timestamp, field, generalised timestamp, session-key, etc. The set of cryptographic primitives supported for compilation with `PD2IS` is also larger than the one presented in Chapter 3 and Chapter 4. In the excerpt of an `XSD` schema given in Figure 5.3, we can observe this extension of the signature $\mathcal{S}$ within the `PD2IS` toolkit (see the `XSD`-element called *BasicType*). We note that an atomic term is nevertheless described within `PD2IS` exactly as in Chapter 3. Hence, an atomic term is characterised by a type (e.g., sort), a name (e.g., the symbolic term) and a possible range. In `PD2IS`, each atomic term is also associated with an identifier (i.e., in the excerpt from

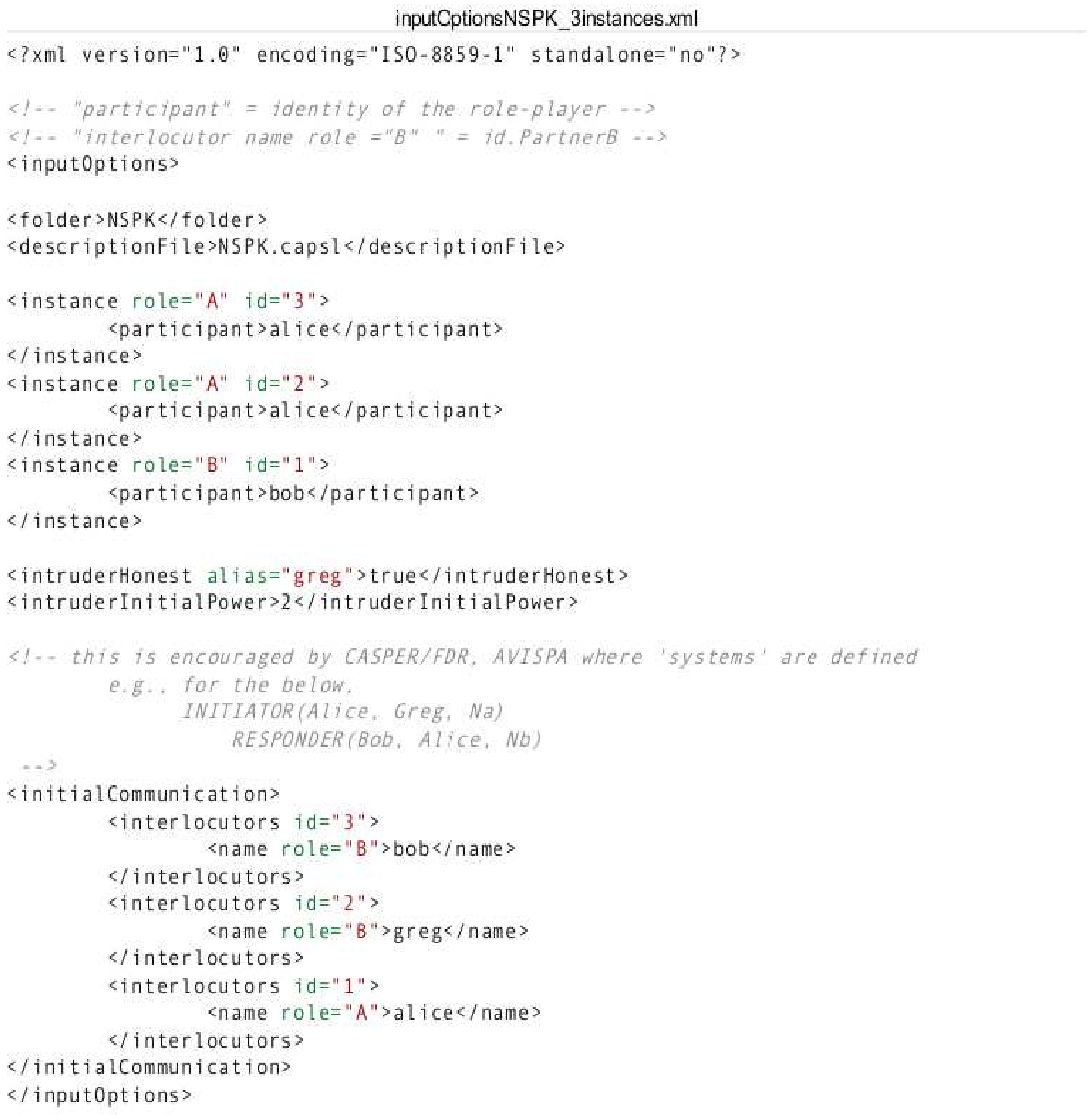


Atom.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:element name="atom" type="Atom"> </xs:element>
    <xs:complexType name="Atom">
        <xs:sequence>
            <xs:element name="type" type="BasicTerm"></xs:element>
            <xs:element name="name" type="xs:string"></xs:element>
            <xs:element name="id" type="xs:positiveInteger"></xs:element>
            <xs:element name="values"  type="Range"></xs:element>
        </xs:sequence>
        <xs:attribute name="propMsg" type="xs:integer"  ></xs:attribute>
        <xs:attribute name="propRole" type="xs:string"></xs:attribute>
        <xs:attribute name="propComposite" type="xs:string"  ></xs:attribute>
    </xs:complexType>

    <xs:simpleType name="Range">
        <xs:union>
            <xs:simpleType>
                <xs:list itemType='xs:integer'/>
            </xs:simpleType>
            <xs:simpleType>
                <xs:list itemType='xs:string'/>
            </xs:simpleType>
        </xs:union>
    </xs:simpleType>

    ....

    <xs:simpleType name="BasicType">
        <xs:restriction base="xs:string">
            <xs:enumeration value="nonce"/>
            <xs:enumeration value="agentName"/>
            <xs:enumeration value="skey"/>
            <xs:enumeration value="timestamp"/>
            <xs:enumeration value="pkey"/>
            <xs:enumeration value="field"/>
            <xs:enumeration value="sessionkey"/>
        </xs:restriction>
</xs:simpleType>

    <xs:attribute name="propMsg" type="xs:integer"  ></xs:attribute>
    <xs:attribute name="propComposite" type="xs:integer"  ></xs:attribute>

 <xs:complexType name="BasicTerm">
            <xs:simpleContent>
                <xs:extension base="BasicType">
                    <xs:attribute ref="propMsg" use="required"/>
                    <xs:attribute ref="propComposite" use="required"/>
                </xs:extension>
            </xs:simpleContent>
</xs:complexType>
</xs:schema>
```

Figure 5.3 An Excerpt of the `XSD` Schema for Atomic Terms

Atom.xsd given in Figure 5.3, note the field called *id*).

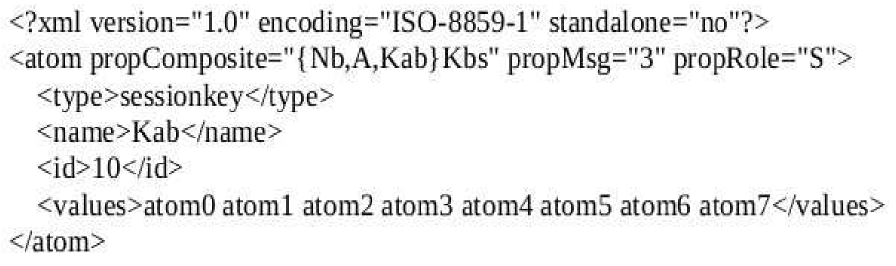

inputOptionsNSPK_3instances.xml

```xml
<?xml version="1.0" encoding="ISO-8859-1" standalone="no"?>

<!-- "participant" = identity of the role-player -->
<!-- "interlocutor name role ="B" " = id.PartnerB -->
<inputOptions>

<folder>NSPK</folder>
<descriptionFile>NSPK.capsl</descriptionFile>

<instance role="A" id="3">
        <participant>alice</participant>
</instance>
<instance role="A" id="2">
        <participant>alice</participant>
</instance>
<instance role="B" id="1">
        <participant>bob</participant>
</instance>

<intruderHonest alias="greg">true</intruderHonest>
<intruderInitialPower>2</intruderInitialPower>

<!-- this is encouraged by CASPER/FDR, AVISPA where 'systems' are defined
        e.g., for the below,
              INITIATOR(Alice, Greg, Na)
                  RESPONDER(Bob, Alice, Nb)
 -->
<initialCommunication>
        <interlocutors id="3">
                <name role="B">bob</name>
        </interlocutors>
        <interlocutors id="2">
                <name role="B">greg</name>
        </interlocutors>
        <interlocutors id="1">
                <name role="A">alice</name>
        </interlocutors>
</initialCommunication>
</inputOptions>
```

Figure 5.4 An Example of User-Input to PD2IS

We now explain the input given to PD2IS; Figure 5.4 gives an example of a input file (describing a fixed scenario for the NSPK protocol). It indicates the CAPSL protocol description to be parsed (see the elements *folder* and *descriptionFile* in the XML file in Figure 5.4). Through the elements called *instance*, the user gives a scenario, but not a fixed one (i.e., up to some bound, the user provides participant-names for each role). Optionally, the user can associate a number with each instance (in Figure 5.4, see the attribute *id*); this identifier is used mainly in specifying a fixed scenario, if necessary. If the `XML` element *intruderHonest* is present in the input file, then the Dolev-Yao insider is named according to the `XML` attribute *alias* of this element. To avoid trivial knowledge of, e.g., the intruder's identity, the range of the honest participants will be larger than the one provided by the user. The entry called *intruderPower* states how many values for a nonce/short-term key are to be assigned to the Dolev-Yao insider in the model. As mentioned before, the intruder does not get more power if he is assigned more than one nonce-value; this is because one nonce $n_{\mathbf{I}}$ can generate $\{n_{\mathbf{I}}\}_{K_{\mathbf{I}}}$, $\{\{n_{\mathbf{I}}\}_{K_{\mathbf{I}}}\}_{K_{\mathbf{I}}}$, ..., etc. However, attacks of shorter traces are possible if more data is provided to the intruder from the initial setup. The `XML` input file can terminate here. In this case, `PD2IS` can generate: *1)* the `ISPL` file for the Υ_{IS}^{Pr} model corresponding to the loose scenario depicted so far and *2)* the `ISPL` files for constrained and un-constrained models $\Upsilon_{IS_{fixed}}^{Pr}$ for the fixed scenarios that can be unwound for this input (e.g., for the input in Figure 5.4, `PD2IS` will consider all scenarios of size 2 over a corresponding range of names and all scenarios of size 3 under the corresponding range of names). If the input file ends here, then the user will be prompted to direct `PD2IS` to generate the models either according to option 1 or according to option 2 above. If the input file does continue, then it means that the user is providing a fixed scenario as input (i.e., under the element called *initialCommunication*, the user gives the communication partners of the instances already stipulated). Then, `PD2IS` will generate the `ISPL` files for constrained and un-constrained models for the fixed scenario given as input.

We proceed with details of the implementation of `PD2IS` modules other than the `description` module. The `parser` module scans the `CAPSL` description indicated in the input file. Figure 5.5 gives one example of an `XML` file generated by the `parser` module and compliant to the schema *Atom.xsd* presented in Figure 5.3. It exemplifies a small part of the results produced by `PD2IS` when parsing the description for the KSL [111] protocol description. Figure 5.5 shows that, in the KSL protocol, there is an atom Kab that is a session-key and that appears in composite $\{Nb, A, Kab\}Kbs$ of message number 3. Through the `XML` attribute called *propRole*, the file *Atom.xsd* states that Kab is an atom owned by the S-role (i.e., $Kab \in OwnedAtoms^S$). Within the implementation, Kab will be identified as atom 10 (i.e., according to element *id*). The session-key Kab will range in the



```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no"?>
<atom propComposite="{Nb,A,Kab}Kbs" propMsg="3" propRole="S">
   <type>sessionkey</type>
   <name>Kab</name>
   <id>10</id>
   <values>atom0 atom1 atom2 atom3 atom4 atom5 atom6 atom7</values>
</atom>
```

Figure 5.5 An XML File for an Atomic Term in the KSL Protocol

model to-be-generated over the set $\{atom0, \ldots, atom7\}$ (hence, the XML element called *values*).

Whilst working with an extended cryptographic signature, `PD2IS` does not assume full-typing (as in Chapter 3). Instead it implements tagging schemes similar to those in [102], where certain different sorts and/or composites can bear the same type-tag (e.g., nonces and short-term keys). Thus, the `parser` module maps nonces and short-term keys over the same ranges. This is due to the necessity of capturing type-flaw attacks (e.g., the intruder might insert into a fraudulent message a value that he learned from a nonce, as if it were a short-term key). In the `data-setup` module the semantics of *synth* (as per Section 3.3.1) is generalised to support this extension.

As part of the signature extension, the toolkit supports atoms of sort *timestamp*. Such variables are mapped over bounded-integers. In order to support timestamps, in the `data-setup` module we implement a MAS semantics which extends the one in Chapter 3. In brief, an extra variable named *clock* ranging over bounded-integers is added to the local state of each agent whose role views the manipulation of timestamps. These variables are incremented synchronously for all agents at each joint action which is not of type *analz* or *synth*. The semantics of matching (i.e., *out_match*) is extended as follows: a timestamp is acceptable if it is no “older” than x-units of times in comparison with the value of the variable *clock*. The threshold x for accepting timestamps varies with the size of the protocol and the size of the scenarios. For `MCMAS`-wise optimisation, we also model timestamps and clocks ranging over an enumeration type (i.e., we use $\{number1, number2, \ldots, numberN\}$, for N a fixed integer) instead of the built-in `bounded integer` type. Firstly, in such a modelling of timestamps, the underlying BDDs are smaller. Secondly, it enables us to use generalised timestamps in our framework; we model a generalised timestamp as a variable ranging over the Cartesian product of the ranges for nonces and the enumeration-range for timestamps. Making the agent-names and their respective public keys range over the same set of values is another optimisation that we implemented in PD2IS (i.e., to reduce the state-space triggered by the PD2IS-produced models).

In the evolution function of an agent ag_A, it is often the case of comparing $ag_A.t = Environment.t$ (e.g., for the implementation of *out_match* where ag_A is checking that what she received is in accordance with something she previously held). Also, the denotations of predicate symbols usually contain comparisons of the kind $ag_A.t = ag_B.t$. Note however that the ranges for $Environment.t$, $ag_A.t$ and/or $ag_B.t$ can be different (especially when dealing with constrained models for fixed protocol scenarios). We have therefore extended MCMAS to support comparison between terms that do not have the same range, but one is ranging over a subset of the codomain of the other. In doing so, we have tackled some of the inconveniences in using constrained models for fixed protocol scenarios.

In the case of generating generic models Υ_{IS}^{Pr} for loose scenarios and/or un-constrained model for fixed scenarios, the procedures in the `data-setup` module that generate ranges for composites and messages can be expensive. Given the specific ranges for atomic-terms, these procedures generate (as explained in Section 5.2.2) either all the ordered n-tuples (i.e., the generalised Cartesian products) or only the symmetry-free ordered n-tuples. For a medium size protocol (e.g., 5 rules) and a loose scenario of size greater than 6, practice showed that it is possible that the JAVA Virtual Machine exceeds its memory limits. For generic models for large size loose scenarios/protocols, we have consequently implemented the writing to disk of the ordered n-tuples generated (i.e., in order to store the generalised Cartesian product, the `data-setup` module does write some data to disk instead of the original case where it uses it all to populate JAVA objects). At ISPL generation time, we retrieve the data from the disk. This method lengthens the model-generation time. Though implemented we rarely faced the need to use this intricate generation method. This is because attacks are usually found by verifying models for loose scenarios and/or un-constrained model for fixed scenarios of small sizes (e.g., scenarios implying 3-4 agents). For the most intricate protocols tested, the generation of these models is a matter of maximum 3 minutes.

The bottleneck of generating all the possible-message space in the case of generic models Υ_{IS}^{Pr} for loose scenarios and/or un-constrained model for fixed scenarios is therefore another argument to use constrained models for fixed fixed scenarios. In the case of the latter, the generation does not require writing to disk and it is a matter of few seconds.

We will now present aspects of the implementation of the generation of formulae. This line follows in itself the MVC design pattern too. The `parser` module processes `CAPSL` goals into assertion objects. The hierarchy of assertions is implemented through interfaces and classes in the `utils` module. Assertions follow the grammar of the atomic and complex `CAPSL` goals. Thus, assertions can be simple and/or of different levels of nesting. They are also split in atomic and epistemic. The most general assertion class (called *Assertion*) is characterised by a type, a subject role, an object role and a data array (e.g., `PRECEDES A:B | na` yields an assertion of type *precedes*, with A as the subject role, B as the object role and the data array filled with the atom na). Given the instantiations, the `data-setup` module processes assertions per role into predicates per agents. For one assertion, there could be several predicates admissible in the translation. This is aligned with the hierarchy for formulae corresponding to expression-relation ρ_{IS} in Chapter 3 . To these sets of predicates, we add others which correspond to Gollman's goals (see [93] or Chapter 2, page 49). We systematically separate formulae into CTL and CTLK reachability, reactivity and safety (see Chapter 2, page 26). The `producer` module will finally translate these into CTLK formulae expressed in the `ISPL` syntax.

We supply some excerpts from the `ISPL` programs generated by the `producer` model in `PD2IS` from an NSPK protocol scenario. The first code-sample is a very simplified and commented version of the code generated for an A-agent. It can be seen that agents' local variables encode *stores*; the agents' actions and local protocols contain instantiated *send* and *receive* actions; the agents' local evolutions are described by appropriate matching preconditions and setting postconditions.

Example 5.3.1 *Simplified ISPL code for an A-agent*

```
Agent ag_A -- Encodes an NSPK instance (alice, A-role)
 Vars: --Encodes <stores>
   -- The Id of the agent ag_A is stored in A:
   A: {alice};
   --The communication partner B is fixed:
   B: {bob1, bob2, greg1, greg2}
   Na, Nb: {r1,r2,...};
   Step: {0,1,2,3};
 end Vars
```

```
Protocol: --Encodes <A-role>
  --Step 0:
  Na=X and Step=0: {send_enc_alice_X_pubkey_bob1, send_enc_alice_X_pubkey_bob2, .... };
  --A rule as above for each X in the nonce range {r1,r2,...}

  --Step 1:
  Step=1: {receive};

  --Step 2:
  Na=X and Nb=Y and Step=2: {send_enc_X_Y_pubkey_bob1, send_enc_X_Y_pubkey_bob1, .. };
  --A rule as above for all X,Y in the nonce range {r1,r2,...}

  --Step 3:
   Step=3: {empty};
end Protocol
Evolution:
  --Step 0 and step 2:
  Step=Step+1
  if
  Action=send_X and Env.Action=intercept_X;
  --A rule as above for each message X

  --Step 1:
  Step=Step+1  and
  Nb=Y --<SET> assigns nonce Y to X
  if
  Action=receive_enc_Na_Nb_pubkey_alice and
  Env.Action=transmit_enc_X_Y_pubkey_alice and
  Na=X; --<OUT_MATCH> checks the consistency of Na
  --A rule as above for all X,Y in the nonce range {r1,r2,...}

  --Step 3
  -- No update to local state
end Evolution
```

The local variables of the Environment cited below denote the list *POOL* described in Chapter 3. The Environment's *protocol section* and *evolution section* encode an optimisation of the Dolev-Yao deductions described in Chapter 3 (e.g., restricted protocol function upon the knowledge-set, analysis carried out only with the view of composing messages due at the next step, etc.). Below we give a simplified version of an `ISPL` code snippet for the Environment agent in a model for an NSPK scenario.

Example 5.3.2 *A simplified fragment for the Environment agent in* `ISPL`

```
Vars:
 knows_X:boolean;   -- Represents whether nonce X is in the knowledge set
 --A line as above for each X in the nonce range {r1,r2,...}
 ...
end Vars
Protocol:
 --Transmit actions enabled when nonce X is in the knowledge-set:
 knows_X: {transmit_enc_alice_X_pubkey_bob1, transmit_enc_alice_X_pubkey_bob2, ...};
 --A rule as above for each X in the nonce range {r1,r2,...}
 ...
end Protocol
Evolution:
 --DY decomposition upon intercept of enc_A_Na_pubkey_B:
 knows_X = true
 if
 Action=intercept_enc_alice_X_pubkey_greg1 and
 ag_A.Action=send_enc_alice_X_pubkey_greg1
 --A rule as above for each X in the nonce range {r1, r2, ...}
 --and for each A-agent ag_A
 ...
end Evolution
```

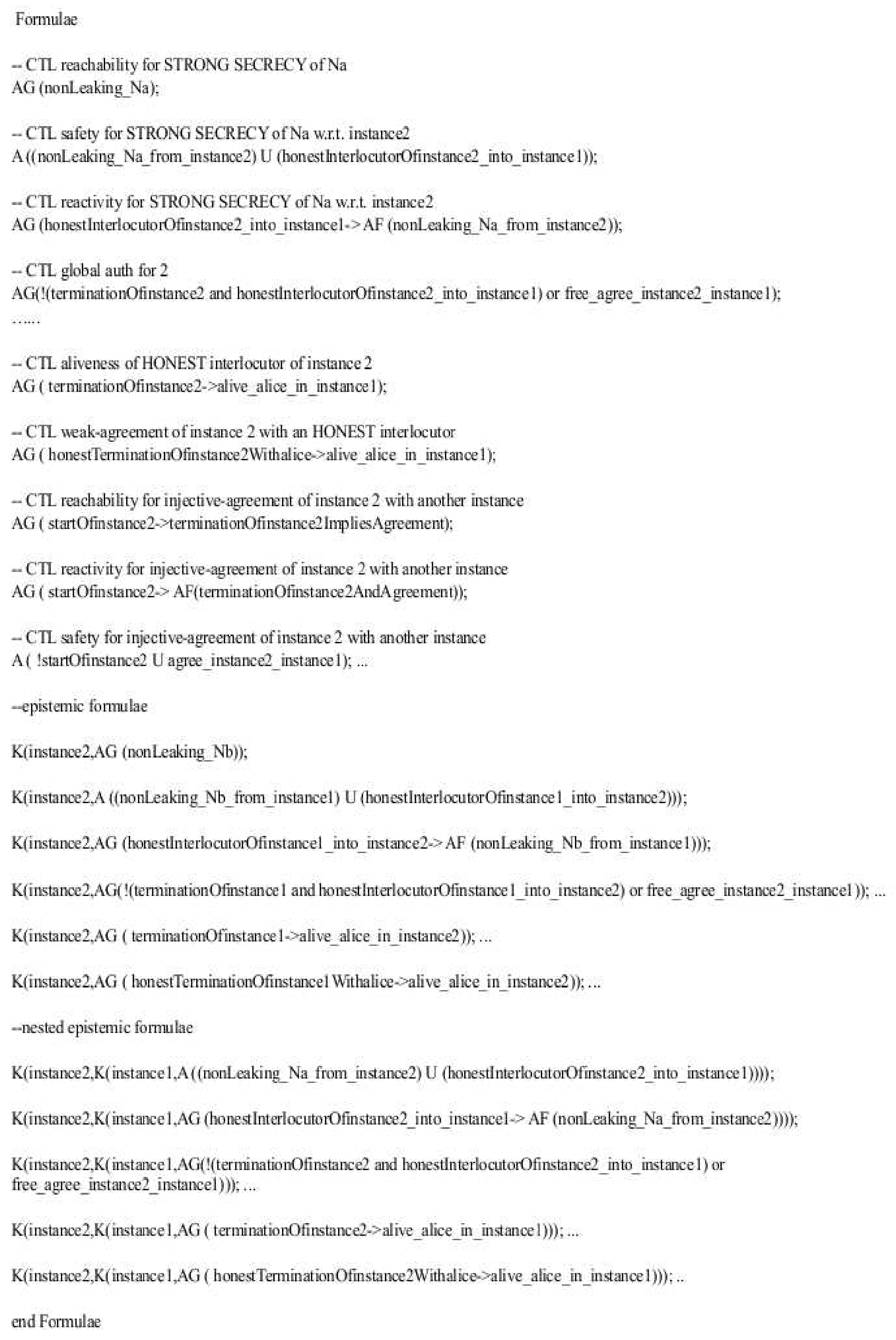
The actual ISPL files produced by PD2IS are less intuitive than the ones presented above as they are heavily optimised to reduce the state-space of the generated model. For instance, a lot of the communication process is simplified by the use of ISPL local observable and observable variables. This makes it possible for some of the ISPL code corresponding to the evolution function of the Dolev-Yao agent to be placed within the honest agents' evolution lines, triggering the direct setting of variables in the respective agents.

A snippet of the formulae generated in ISPL for an NSPK scenario is shown in Figure 5.6.

To sum up, in Sections 5.3.1 and 5.3.2 we have presented the architectural and the implementation details of the PD2IS toolkit. In summary, the PD2IS toolkit takes CAPSL protocol description, processes it into IS formalisations based on the Υ_{IS} formalism and produces (several) corresponding optimised ISPL files ready to be verified by MCMAS.

5.4 PD2IS-based Protocol Verification

In this section we evaluate our MAS-based methodology to verify security protocols. In other words, this section presents and discusses the experiments of automatic protocol-model generation



```
Formulae

-- CTL reachability for STRONG SECRECY of Na
AG (nonLeaking_Na);

-- CTL safety for STRONG SECRECY of Na w.r.t. instance2
A ((nonLeaking_Na_from_instance2) U (honestInterlocutorOfinstance2_into_instance1));

-- CTL reactivity for STRONG SECRECY of Na w.r.t. instance2
AG (honestInterlocutorOfinstance2_into_instance1-> AF (nonLeaking_Na_from_instance2));

-- CTL global auth for 2
AG(!(terminationOfinstance2 and honestInterlocutorOfinstance2_into_instance1) or free_agree_instance2_instance1);
......

-- CTL aliveness of HONEST interlocutor of instance 2
AG ( terminationOfinstance2->alive_alice_in_instance1);

-- CTL weak-agreement of instance 2 with an HONEST interlocutor
AG ( honestTerminationOfinstance2Withalice->alive_alice_in_instance1);

-- CTL reachability for injective-agreement of instance 2 with another instance
AG ( startOfinstance2->terminationOfinstance2ImpliesAgreement);

-- CTL reactivity for injective-agreement of instance 2 with another instance
AG ( startOfinstance2-> AF(terminationOfinstance2AndAgreement));

-- CTL safety for injective-agreement of instance 2 with another instance
A ( !startOfinstance2 U agree_instance2_instance1); ...

--epistemic formulae

K(instance2,AG (nonLeaking_Nb));

K(instance2,A ((nonLeaking_Nb_from_instance1) U (honestInterlocutorOfinstance1_into_instance2)));

K(instance2,AG (honestInterlocutorOfinstance1_into_instance2-> AF (nonLeaking_Nb_from_instance1)));

K(instance2,AG(!(terminationOfinstance1 and honestInterlocutorOfinstance1_into_instance2) or free_agree_instance2_instance1 )); ...

K(instance2,AG ( terminationOfinstance1->alive_alice_in_instance2)); ...

K(instance2,AG ( honestTerminationOfinstance1Withalice->alive_alice_in_instance2)); ...

--nested epistemic formulae

K(instance2,K(instance1,A ((nonLeaking_Na_from_instance2) U (honestInterlocutorOfinstance2_into_instance1))));

K(instance2,K(instance1,AG (honestInterlocutorOfinstance2_into_instance1-> AF (nonLeaking_Na_from_instance2))));

K(instance2,K(instance1,AG(!(terminationOfinstance2 and honestInterlocutorOfinstance2_into_instance1) or
free_agree_instance2_instance1))); ...

K(instance2,K(instance1,AG ( terminationOfinstance2->alive_alice_in_instance1))); ...

K(instance2,K(instance1,AG ( honestTerminationOfinstance2Withalice->alive_alice_in_instance1))); ..

end Formulae
```

Figure 5.6 An Excerpt of the CTLK Formulae for a Fixed NSPK Scenario

and temporal-epistemic model checking carried out with the PD2IS toolkit and the MCMAS model checker, respectively.

To begin with, we recall that PD2IS and MCMAS can be used to generate and verify: *1)* the Υ_{IS}^{Pr} model for a loose protocol scenario (i.e., a principal-instantiation of exponential dimensions in the size of the protocol) the protocol Pr; *2)* the $\Upsilon_{IS_{fixed(i)}}^{Pr}$ model for a fixed scenario i of the protocol Pr. In the first case, given the generality of the problem approached, PD2IS and MCMAS are to be considered a *protocol analysis methodology*. In the second, given the model-abstractions operated within $\Upsilon_{IS_{fixed(i)}}^{Pr}$, PD2IS and MCMAS are to be considered an *attack-finding methodology*. Amongst other aspects, in this section we are going to evaluate these both facets incorporated in PD2IS and MCMAS.

We explain the structure of this section, with respect to evaluating our methodology.

- Section 5.4.1 evaluates the significance of the results obtained by PD2IS and MCMAS when used as an attack-finding methodology. We draw several well known authentication and key-establishment protocols from customary repositories [49, 119]. Models for numerous fixed scenarios of each protocol are automatically generated and checked against automatically produced temporal-epistemic formulation of their respective goals. Well-established settings and novel settings (e.g., leaked keys) are explored in the verification process. We report and discuss both the attacks found and the parameters (e.g., state-space of unwound models, verification time, etc.) of the verification lead. To emphasise our AI-inspired methodology, the last part of Section 5.4.1 evaluates separately the results concerning the satisfaction/refutation of essentially epistemic goals.

- Section 5.4.2 evaluates the performance of PD2IS and MCMAS when used as a general protocol verification methodology. In that sense, models for large, loose protocol scenarios are generated and verified. At the same time, Section 5.4.2 draws a comparison between the performance of PD2IS and MCMAS in generating/verifying these models for loose scenarios and the models for fixed scenarios (discussed previously, in Section 5.4.1).

- Section 5.4.3 compares the performance in attack-finding for security protocols of PD2IS and MCMAS versus other state-of-the-art tools in the field.

5.4.1 `PD2IS`: Attack-finding Evaluation

In our framework, an *attack* on a protocol Pr is a trace of Υ_{IS}^{Pr} or $\Upsilon_{IS_{fixed(i)}}^{Pr}$ exhibiting the failure of a CTLK formula included in the expression ρ_{IS} of a security goal of the protocol Pr, where i is some fixed scenario of Pr. In practice, an attack on Pr is found if `MCMAS` outputs a counterexample trace exhibiting a CTLK formula, one of the ρ_{IS}-expressions of a security goal of the protocol Pr failing.

To evaluate the `PD2IS` tool in a systematic way, we ran tests on protocol descriptions from the `CAPSL`-version of the Clark-Jacob's library [49] and the SPORE library [119]. In addition to the atomic `CAPSL` goals considered traditionally, we added a number of complex `CAPSL` goals to each `CAPSL` input file. Specifically, we included complex authentication goals with up to two levels of nesting of knowledge operators (as per Example 3.2.1 and Example 3.2.2). Table 5.2 reports on the experimental results of verifying several `PD2IS`-generated constrained models for fixed scenarios of the protocols in the libraries mentioned above.

The first column in the table specifies the protocol being checked; KSL1 and KSL2 stand for variants of the Kerberos protocol described in [111] and [136] respectively; these are RO protocols which are RT reducible, whereas the others are RT protocols. The second column indicates whether we assume that keys can be leaked ("learned") to the intruder. This is in order to capture known-key attacks (as reviewed in Chapter 2, page 35). The third column reports the number of atomic `CAPSL` goals for which `MCMAS` found an attack. The fourth gives the average verification time that `MCMAS` took to verify a single `ISPL` file (i.e., one single, constrained model for a fixed scenario) while searching for possible attacks. Differently from other approaches, in our approach we systematically generate the `ISPL` files corresponding to each fixed protocol scenario possible under some given principal-name instantiation. In this case, the table reports the results obtained through the verification of constrained models for fixed scenarios with up to 3 agents per protocol role, generated with `PD2IS`. Each of these files were passed to `MCMAS`, one by one, until either an attack on an atomic goal was found or all the generated files had been checked. The last column reports the total time used by `PD2IS` and `MCMAS` while performing these checks sequentially. Note that in principle the various `ISPL` files could be checked in parallel thereby reducing the total verification

Protocol	Learned Keys	Attacks	Avg. time (secs)	Total time (secs)
ISO1PUCCF	off	none	1	23
	on	1	1	3
ISO2PUCCF	off	none	2	42
	on	2	2	6
ISOSK1PU	off	none	2	46
	on	1	1	2
ISOSK2PU	off	none	3	63
	on	2	2	8
ISOSK3PM	off	none	4	80
	on	4	3	13
AndrewRPC	-	2	4	60
NSPK	-	1	1	19
WideMouthFrog	-	1	7	56
KSL1	-	1	7	55
KSL2	-	1	12	183

Table 5.2 Attack-finding with `PD2IS`

time.

By inspecting row 7 of Table 5.2, we observe that `MCMAS` verified 19 `PD2IS`-generated models each corresponding to a NPSK fixed scenario until the Lowe attack [135] was exhibited. The average time of verifying these models was just over one second. Therefore, we conclude that `PD2IS` and `MCMAS` can be used as an attack-finding toolkit with fast response times. In practice, most protocol verification tools do not run averaged tests of these kind. In turn, scenarios prone to attack are modelled and verified individually. In that context, the model corresponding precisely to the fixed scenario for Lowe attack [135] is instantaneously generated by `PD2IS` and it is verified by `MCMAS` in just under one second (we used `tstime` to measure the performance). Comparably promising times are shown also in the case of the more complicated, RT-reducible protocols like KSL [111] (see the last two rows of Table 5.2).

Table 5.2 does not report the size of the unwound models (e.g., state-space, number of BDD variables used by `MCMAS`). This is because in the case of attack-finding on models for fixed scenarios these were found unproblematic. Along the same lines, the `PD2IS` generation of these models is instantaneous. In turn, models for unfixed scenarios are larger and therefore exhibit a greater complexity in generation and/or verification. Section 5.4.2 will detail these models and the statistics relating them to `PD2IS` and `MCMAS`.

As Table 5.2 shows, several attacks were found using `PD2IS` and `MCMAS`. A large number of the counterexamples thus exhibited symbolised replay, impersonation and interleaving attacks, all of which were known to exist for the protocols considered (for the Andrew Secure RPC, NSPK, KSL, Wide Mouth Frog, Woo-Lam protocols). `MCMAS` also reported the shortest path depicting these attacks. This intuitively certifies the methodology and its implementation. In Figure 5.7, we present the minimal representation of the famous Lowe attack [135] on NSPK, as it is found with `PD2IS` and `MCMAS`. On the timed-model, for KSL, we have captured the attack on the freshness of timestamps, originally found in [136], by pen-and-paper methods. For the ISO protocols analysed, testing the setting of leaked long-term keys is innovative (i.e., not usually explored by other attack-finding methodologies). In this setting, we have therefore found some new binding[3] attacks. This confirms the difference that we originally underlined, at a theoretical level, between our modelling

[3]See Chapter 2, page 35.

and the LDYIS [125] (see Chapter 1, page 292).

Irrespective of the attractive verification results for atomic goals, as previously remarked, our methodology focuses on verifying specifications containing knowledge operators. To illustrate this point, Table 5.3 presents the results for two NSPK scenarios checked against the epistemic CAPSL goals in Example 3.2.1 and Example 3.2.2. As reported in the table, the epistemic CAPSL goals both hold unless the scenario includes a "corrupt insider", i.e., unless the intruder initially knows the private key of some principal whom other principals trust. This follows, of course, our intuition. The two epistemic CAPSL goals considered are automatically compiled into temporal-epistemic specifications as described in Example 3.3.25 and Example 3.3.26 respectively.

Also note that adding further levels of nesting of knowledge modalities may falsify an initially true CAPSL goal. For example, the experiments confirm that in the case of the ISO1PUCCF protocol while the first-level epistemic goal from Table 5.3 holds, the second-level goal from Table 5.3 fails.

Insider	$\texttt{Knows}\,B : \texttt{holds}\,A : \texttt{Na}$	$\texttt{Knows}\,A : \texttt{Knows}\,B : \texttt{holds}\,A : \texttt{Na}$
No	True	True
Yes	False	False

Table 5.3 Checking Epistemic Goals for NSPK models

5.4.2 Model Evaluation: Υ^{Pr}_{IS} vs. $\Upsilon^{Pr}_{IS_{fixed}}$

In the previous section we have evaluated the PD2IS toolkit with respect to generating and verifying models for fixed protocol scenarios. In this section the emphasis is placed on generating/verifying models for loose protocol scenarios. We will also summarise the performance difference between verifying models for loose protocol scenarios (i.e., Υ^{Pr}_{IS}) and verifying constrained models for fixed protocol scenarios (i.e., $\Upsilon^{Pr}_{IS_{fixed(i)}}$, for i being some fixed protocol scenario). We now proceed to setting the details for the aforementioned comparison.

On the one hand, we consider models for fixed protocol scenarios (i.e., like previously, in Section 5.4.1). We only consider constrained models (i.e., where not only the initial setup is fixed

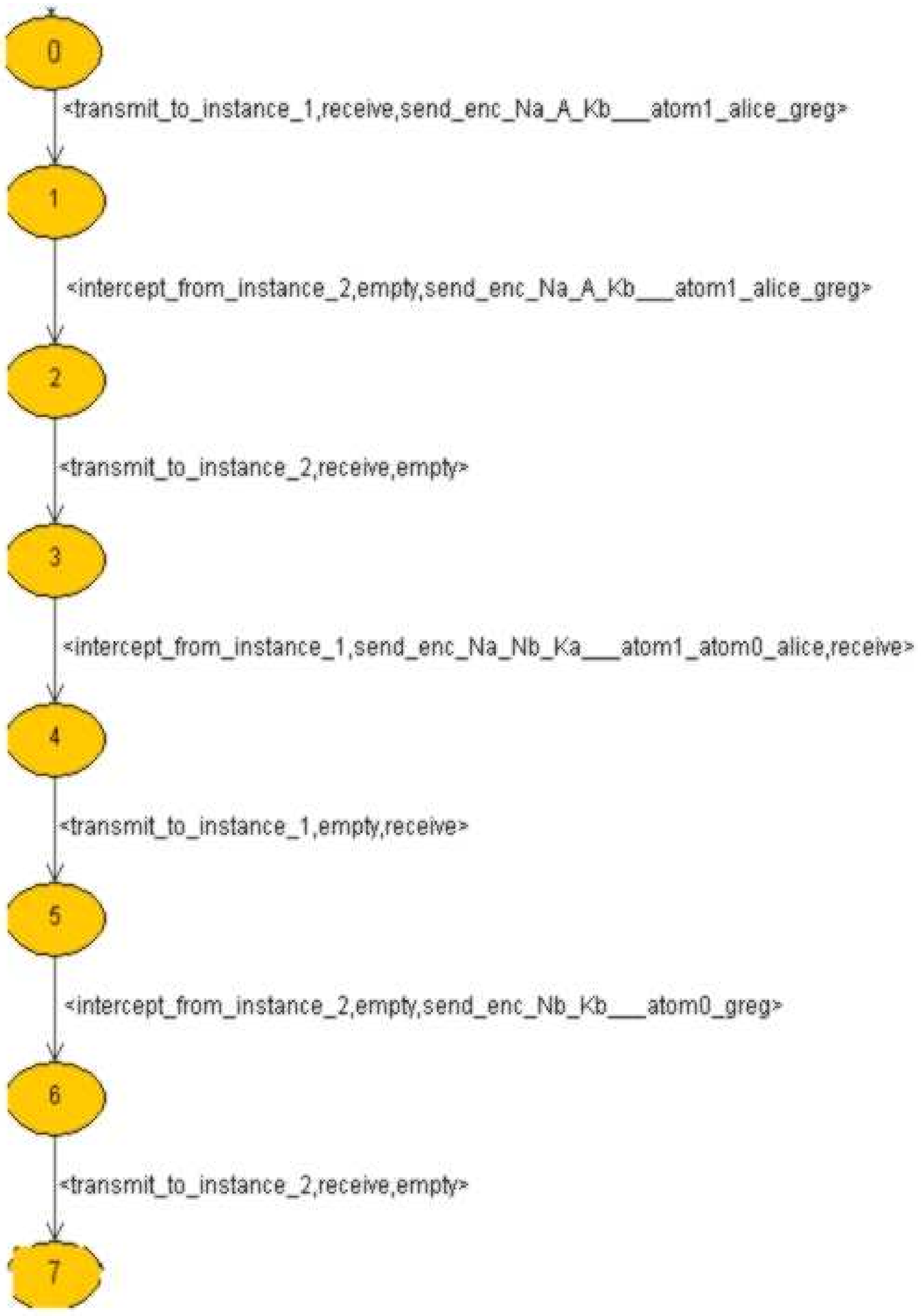


Figure 5.7 Lowe's Attack [135] on NSPK found with PD2IS and MCMAS

according to the scenario, but the ranges of terms are heavily restricted). In the case of protocols where we are looking for a particular attack that does not require the intruder to generate data (e.g., the Lowe attack [135] on NSPK), we construct the constrained Υ_{IS} models assigning no atoms in the initial state setup to the Dolev-Yao intruder (i.e., in the `XML` input file, as exemplified in Figure 5.4, we make the value of the element *intruderPower* equal 0). This reduces further the state-space of constrained models.

On the other hand, we consider generic Υ_{IS}^{Pr} models for loose scenarios. In particular, we consider a generic model Υ_{IS}^{NSPK} for NSPK, instantiating three agents and the intruder. No range-related restrictions or initial setup constraints are applied to this Υ_{IS}^{NSPK}. In terms of generating the model, `PD2IS` requires approximately 6 seconds. The Υ_{IS}^{NSPK} is still reasonably small in comparison to models that we usually generate/modify (e.g., 6000 `ISPL` lines). Therefore, `PD2IS` performs well in generating all this model employing just the stack (i.e., without writing model-related data to disk).

We now report on large models for loose scenarios that still raise no difficulty for `PD2IS` to generate using just the stack (i.e., using no intermediate serialisation to disk of data-objects). For comparison purposes, we will hereby refer to those protocols for which the models for fixed scenarios have already been discussed in Section 5.4.1. In Table 5.4 we give a guideline regarding the size of the generic IS programs for the loose scenarios generated/verified with `PD2IS` of these protocols. In row 4 of Table 5.4, we can see that the largest of these interpreted systems is generated by `PD2IS` in less than two minutes (i.e., including the actual writing to disk of the corresponding `ISPL` file).

Protocol	Size of Scenario	Size of Generic IS Model/ `ISPL` file (number of `ISPL` lines generated)
ISO1PUCCF	2 instances + active intruder	18,105
ISO2PUCCF	2 instances + active intruder	18,105
ARSPC	3 instances + active intruder	36,234
KSL	2 instances + active intruder	299,269

Table 5.4 Size of Generic IS Models for Loose Scenarios

We conclude that, for models for loose scenarios, `PD2IS` generates large size `ISPL` programs (i.e., 3×10^5 number of `ISPL` lines) with no exceptional difficulty (i.e., in less than two minutes and having a maximum of 2GB of memory available).

To complete the picture, we will now refer to the parameters yielded by the verification of systems for loose scenarios. In Table 5.5 we show the difference in the statistics produced when analysing a generic model for a loose scenario of NSPK and verifying the constrained model for a fixing of that scenario of NSPK. In more detail, the third column of the table refers to a generic model for a loose scenario of NSPK where: *alice* instantiates B, *alice* also instantiates A, B is substituted to *bob* and the intruder is represented by *greg*. The second column of the table reports on the verification of a constrained model for the fixing of the aforementioned scenarios, such that intruder "speaks to" *alice* as B, the intruder "speaks to" *bob* as B and *bob* as B "speaks to" *alice* as A.

Characteristics of the Unwound Model	Fixed Scenario	Loose Scenario
Reachable State Space	39	1.38254e+10
`MCMAS` Reported Time for Verification	1s	104s
Number of BDD Variables Used by `MCMAS`	9501536	57808032
Number of Formulae Verified	10	63

Table 5.5 Verifying Fixed vs. Loose Scenarios for NSPK with 3 Honest Participants

In Table 5.5, column 3, we observe that a Υ_{IS}^{NSPK} for a loose NSPK scenario unwinds an actual model with more than 10^{10} reachable states. In constrast, in the second column, we observe that a constrained model for a fixed NSPK scenario mirroring the Lowe attack [135] unwinds a model with just 39 reachable states. The third row of Table 5.5 shows the number of BDD[4] variables used by `MCMAS` to store and verify the unwound models. A considerable difference is recorded in this case as well, between the model for the loose scenarios and the same size constrained model for a fixed scenario. As we have anticipated in Section 5.3, some of the formulae automatically generated in the generic model Υ_{IS}^{NSPK} for a loose NSPK (i.e., 63 as per the fourth line of Table 5.5) are not well-founded in the constrained model for a fixed NSPK scenario. Hence, only 10 formulae

[4]See Chapter 2 for details on BDDs.

between the two models can be effectively compared. This outlines the two-folded usage of PD2IS: an attack-finding tool (i.e., when testing few formulae on a constrained model for a fixed, well-determined scenario) and a protocol verification methodology (i.e., when analysing more formulae for generic large models).

To evaluate our techniques further, we used the distributed version of MCMAS [108]. In MCMAS [108] the parallelisation is performed by criteria other than spawning the fixed scenarios of a loose scenario. So, this method of verifying Υ_{IS}^{Pr} is an alternative to using models for fixed protocol scenarios. The CUDD calls to re-ordering BDD variables exit with an exception for some of our large models (i.e., for bigger loose scenarios for NSPK, or for loose scenarios of other protocols). Nonetheless, we were able to verify several of our models. In particular, we used MCMAS [108] to verify the model Υ_{IS}^{NSPK} for a loose scenario for NSPK against a CTLK formula due to capture Lowe's attack. We concluded that using distribution through MCMAS [108] does not outperform the security-tailored method of enumerating models for fixed scenarios (i.e., the results presented in Section 5.4.1).

5.4.3 Performance Evaluation: PD2IS vs. Other Tools

In this section we report a comparison between the performance of our PD2IS-based methodology and other state-of-the-art tools in protocol analysis.

The recorded verification times for constrained models for fixed protocol scenarios are not too dissimilar to those produced by currently leading toolkits (run on comparable software and hardware configurations). Moreover, verification results in the literature are most often reported only for one fixed scenario for each given protocol. By contrast, we tested a set of such scenarios and we report the average verification measures yielded by these sequential tests.

We draw a comparison between the performances of PD2IS in tandem with MCMAS and those of other state-of-the-art tools when verifying fixed scenarios of NSPK. Some of the data used in this comparison was drawn from [62]. We report in Figure 5.8. OFMC [19] and Sat-MC [58] (mentioned in Figure 5.8) are tools also based on bounded-number-of-session model checking and they are two of the back-ends in the state-of-the-art AVISPA [9] platform. OFMC [19] is a symbolic (i.e., not ground), on-the-fly model checker. Therefore, it usually outperforms our results when analysing

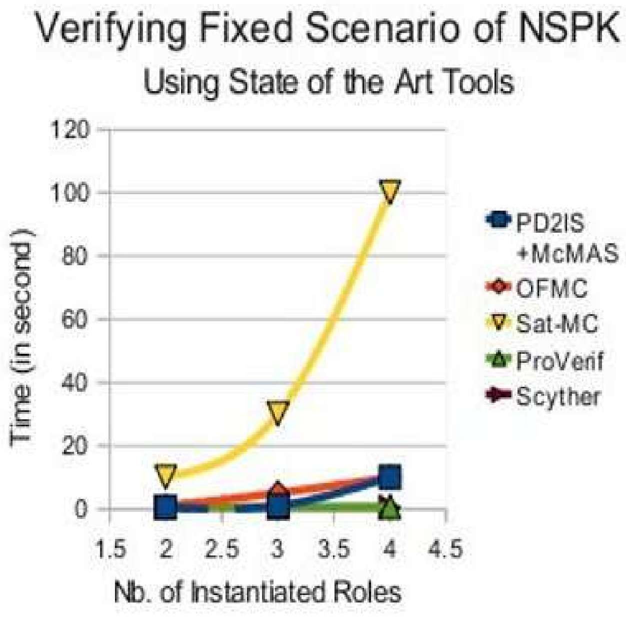


Figure 5.8 Performance Comparison: `PD2IS` and `MCMAS` vs. other tools

larger protocols and/or large scenarios. The last two tools appearing in the comparison, Scyther [61] and ProVerif [23], deal with an unbounded number of protocol session. Their underlying techniques are different and, as we can notice, the time does not vary much with the number of sessions to be verified. However, we recall from Chapter 2 that in the case of certain protocols, their analysis might not ever halt.

We believe the promising results in verifying constrained models for fixed protocol scenarios are due to a combination of factors. These include the implementation of the Dolev-Yao analysis/synthesis optimised for `ISPL` syntax, the underlying efficiency of `MCMAS`, the efficient translation optimised for immediate decoding of messages by the receiver in the RT protocols, and the generation of `ISPL` code in which variable types and ranges were optimised for `MCMAS`. Further, recall that the initial state space and the ranges were constructed to minimise the resulting state space. Lastly, as we said in Section 5.3.2, while we do not employ full-typing, we assume some of the tagging schemes in [102], which –in the implementation– limit the size of the Dolev-Yao inference system.

The machine used in the implementation of `PD2IS` and in the evaluation of the model checking experiments is a dual-core PC with 3 GiB of RAM which has an Intel(R) Core(TM)2 Duo CPU at 2.26GHz; it runs a 64-bit `Open Suse 10.3` operating system, with a 2.6.31.5 kernel. The `MCMAS` used in the experiments is the 0.8.3 branch, where some data range-driven modifications were operated (as we have explained in page 200). `MCMAS` 0.8.3 is linked against the 2.4.2-exp `CUDD` release.

Section 5.4 has evaluated the following aspects in our AI-inspired methodology of verifying security protocols:

1. in Section 5.4.1, the performance of `PD2IS` as an attack-finding methodology (using constrained models for fixed scenarios of for RT and RT-reducible protocols) ;

2. in Section 5.4.1, the testing of those formulations of security goals which are aligned with traditional approaches, as well as the testing of novel, essentially epistemic formulations of security goals;

3. in Section 5.4.2, the performance in generating different flavours of the Υ_{IS} specification in `ISPL` and of analysing the consequent M_{IS} unwound models (i.e., generic vs. (un)-constrained models for loose and fixed scenarios, respectively);

4. in Section 5.4.2, the performance of `PD2IS` as general protocol verification methodology (using generic Υ_{IS} specifications for loose scenarios of for RT and RT-reducible protocols);

5. in Section 5.4.3, the comparison in performance between `PD2IS` and `MCMAS` and other state-of-the-art tools in protocol verification.

To conclude, the MAS models for loose scenarios can be viewed as a means to a general protocol verification technique. Models for fixed scenarios are approximations of the generic MAS models for loose scenarios. Much of the security community analyses such models for fixed scenarios (including the state-of-the-art AVISPA toolkit [9]). To generalise and systematise the approach, we generate all possible fixed scenarios up to some bound and sequentially verify them until an attack is found or the set is exhausted. Employing fixed scenarios nevertheless describes a more specific protocol analysis technique: an attack-finding methodology. As an attack-finding toolkit, our `PD2IS` software does perform comparably to other state-of-the-art tools in the field.

In this chapter we showed a fully automatic methodology to verify a large class of the authentication and key-establishment protocols (i.e., the RT and RO which are RT reducible protocols). This methodology is AI-inspired and it is based on the systematic modelling of RT protocols as MAS models presented in Chapter 3. The correctness guarantees for using our toolkit to verify MAS models for `CAPSL`-described security protocols are founded on the theoretical results presented in Chapter 4. We have presented `PD2IS`, a tool that compiles fixed and loose scenarios for such protocols into IS-based, MAS models of their executions. `PD2IS` also translates the protocols' security goals into a taxonomy of temporal-epistemic specifications. Given the characteristics of the models generated, the `PD2IS` toolkit can be used as both a general protocol verification methodology and an attack-finding methodology. The toolkit starts from a `CAPSL` protocol description and produces (a series of) `ISPL` files; these are MAS models of the execution scenarios for the protocol given as input. The model checker `MCMAS` is called for verification. Therefore, `PD2IS` can be viewed as an implementation of the methodology advanced in Chapter 3, through using the algorithms and guarantees offered by Chapter 4. With it, we have presented what is the first fully automatic and systematic methodology of analysis of MAS models for authentication and key-establishment protocols against temporal-epistemic specifications of their security requirements. The models formalised in Chapter 3 and automatically generated in Chapter 5 are not designed to support epistemic modalities other than those mentioned, e.g., no particular attention is given to "everyone knows" or "common knowledge" modalities [80]. For the models to soundly support such modalities, additional care may need to be taken in the initialisation process, i.e., the generating of protocol scenarios. We add that such modalities are not of interest for the security requirements that we analyse (hence, they are not part of the specification languages we use). However, if there is interest in certain protocols such as contract signing, where such modalities could come into place, then a bespoke evaluation of the scenario generation procedure is advisable.

In later chapters we will present extensions of the `PD2IS` toolkit to cope with more classes of CTLK formulae (i.e., formulae that express properties other than standard security requirements) as well as other tools, similar in nature, but aimed at handling other classes of protocols (particularly, e-voting protocols and, more generally, RO protocols non-reducible to RT protocols).

Chapter 6

Detectability of Security Protocols Failures

Motto: "No man is wise enough by himself."
(Plautus)

If a security requirement is violated in one of the protocol sessions, then this exhibits a protocol failure*. In this chapter we use the M_{IS} protocol model to introduce the notion of* detectability *of properties related to protocol failures, i.e., groups of agents acknowledging facts related to protocol failures. We introduce a taxonomy of detectability specifications expressed in temporal-epistemic logic. We illustrate the practical relevance of detectability in a case study applied to a variant of the Kerberos protocol. We show an extension of the* `PD2IS` *toolkit to cater for model-checking automatically generated MAS models for security protocols against specifications of detectability. We report the experiments carried out on several well known protocols. Part of the material in this chapter was presented in [32].*

6.1 Protocol Failures in M_{IS}

In Chapter 2, page 33, the security requirements of protocols were informally presented. Then, in Chapter 3, Section 3.3.3., we introduced the formalisation of these requirements in the M_{IS}

model. In this section we are going to refine this formalisation of security requirements. We are going to distinguish between those requirements that are explicitly stipulated in a `CAPSL` description and those underlying requirements that follow implicitly from the description. This fine-grained distinction will lead to differentiating between completed protocol attacks and protocol failures leading to an attack.

Security requirements are most often formalised as simple invariants (e.g., state predicates). In the following, consider the M_{IS} model for multi-session protocol executions, as introduced in Chapter 3. To ground the discussion, let us assume that each security requirement of interest appears as an atomic proposition $p \in PV$ in the M_{IS} protocol model. We emphasise however that the detectability specifications introduced in the next section equally apply to more complex, temporal-epistemic expressions of security requirements for privacy, receipt-freeness, etc.

Example 6.1.1 (Authentication Requirement) *A standard formulation for an authentication requirement is: whenever an agent i has performed (at least) N execution steps, there exists a different agent j that agrees with agent i on the values of variables in a selected set Ξ. We denote this as:*

$$auth@N(i, \Xi) = i.Step \geq N \rightarrow \bigvee_{j \in Ag \setminus \{i\}} Agree(i, j, \Xi),$$

where j ranges over agents different from i and $Agree(i, j, \Xi)$ abbreviates $\bigwedge_{X \in \Xi} (i.X = j.X)$, as introduced in Section 3.3.3. We implicitly assume that Ξ includes variables for the intended communication partners. For example, in a model for NSPK, $auth@3(i, \{N_A\})$ abbreviates $auth@3(i, \{A, B, N_A\})$. The notation $auth@3(i, \{N_A\})$ denotes that once all three execution steps of agent i have been completed, agent i and its alleged communication partner j possess the same value for the nonce N_A.

Recall from Section 5.4.1 that an *attack* against a security requirement p is an execution trace of M_{IS} that refutes p. We say that an *attack on p* has occurred whenever p fails in the above sense.

Example 6.1.2 (Attack on Authentication) *An attack on the authentication requirement $auth@N(i, \Xi)$ has occurred if there is an execution modelled in M_{IS} where agent i has completed N*

steps and there is no other agent j that agrees with i on the variables Ξ:

$$i.Step \geq N \wedge \neg \bigvee_{j \in Ag \setminus \{i\}} agree(i, j, \Xi)$$

Intended Goals and Pre-goals. For each protocol a set of security requirements is explicitly specified at design time (and they appear in the `CAPSL` description of the protocol). These stipulate the *intended goals* of the protocol. In this sense, an attack on any of these is an attack on the protocol. Typically, the intended goals of a protocol concern the final state of agents in some execution, e.g., the requirement $auth@3(i, \{N_A, N_B\})$ in a model for NSPK. This is indeed the view that we have taken towards the analysis of goals (i.e., finding of attacks) in all previous chapters.

However, intended goals are achieved gradually throughout a protocol execution. If an intended goal holds at a final step of an agent in an execution, it is expected that meaningful parts of that goal held at some intermediate steps of the agent, i.e., previously in that execution. We use *pre-goals* to denote (implicit) requirements that express parts of intended goals considered at intermediate execution steps. We illustrate below a possible method of deriving the pre-goals given the protocol description (and the intended goals).

Example 6.1.3 (Derivation of Pre-goals) *Assume that the set of intended goals includes the authentication requirement $auth@N(i, \Xi)$. We derive a pre-goal $auth@N'(i, \Xi')$ for each $N' \leq N$ and each subset $\Xi' \subseteq \Xi$ of variables which ought to be set in the local states of agent i at step N'.*

According to the formalisations in Chapter 3 and Chapter 4, the set Ξ' in Example 6.1.3 is given by N'-$LearnedAtoms^A \cap OwnedAtoms^A$, i.e., Ξ'=N'-$LearnedAtoms^A \cap OwnedAtoms^A$. So, obtaining this set is a simple manipulation of the data-structures used in algorithm tr, Chapter 4. In fact, the `PD2IS` toolkit processes these sets to obtain the Υ_{IS} formalisation and the CTLK formulae for protocol requirements. For example, assume that $auth@3(i, \{N_A, N_B\})$ is the intended goal of NSPK. Then $auth@2(i, \{N_A\})$ is a pre-goal for the protocol derived as above, e.g., $N_A \in \{N_A, N_B\}$ is a protocol variable to be "learned"/set at step 2 in the role of A. The pre-goal $auth@2(i, \{N_A\})$ denotes the following: once the second step of agent i has been completed, the agent is expected to share the nonce N_A with its intended communication partner.

We will assume that if a pre-goal fails it is impossible that the protocol will eventually complete successfully. In more detail, for a protocol with a set Γ of intended goals and one pre-goal p we assume that:

$$AG\,(\neg\, p \rightarrow\ AG\,(end \rightarrow \neg\Gamma)) \tag{6.1}$$

where the predicate *end* expresses that the agents have completed all their protocol steps and the formula $\neg\Gamma$ denotes that at least one requirement in Γ fails. In light of property (6.1) it seems reasonable to consider also refutations of pre-goals as protocol failures.

In this section we have refined our original formalisation of security requirements, in the sense that we have distinguished between intended goals and pre-goals. In a nutshell, intended goals are those requirements which refer to the end of a protocol execution and are usually stipulated by the specifier. In turn, pre-goals are logical consequences of the intended goals and are supposed to hold at intermediate points in a protocol execution.

6.2 Detectability of Protocol Failures in M_{IS}

In mainstream verification of security protocols the analysis ends once an attack on an intended goal has been uncovered and the protocol is dismissed as faulty. In practice though, the protocol may be kept deployed even after attacks have been found. For instance, IPSec, WEP, GSM, DNS and SSH1 or even Kerberos have all been shown to be susceptible to attacks, but are still widely used. Often, the cost of updating the deployed system might outweigh the risk of attacks [186] (e.g., especially when part of the protocol is deployed in hardware such as smart cards, mobile phone chips, etc.).

Once a deployed security protocol is found to be open to attacks a new question arises: are attacks *detectable*? Of course, attacks on agent i are by their very nature undetectable by agent i alone. However, attacks may leave a group Gr of agents with sufficient collective information to infer that an attack has taken place. In the following, we discuss the relevant groups Gr, candidates for collectively detecting an attack.

Detectability-Relevant Groups of Agents. Depending on a particular protocol scenario, the following groups of agents distinguish themselves in their ability to detect an attack:

- the group of agents that represent the same participant and instantiate the same protocol-role, e.g., (some of) the simultaneous client-sessions of *alice* in an SSH execution;
- the group of agents that represent the same participant, e.g., (some of) the sessions of *alice*, either as sender or as receiver, in an NSPK execution;
- the group of agents that instantiate the same protocol-role, e.g., (some of) the server's sessions in a Kerberos execution;
- the group of agents that represent different participants, e.g., in an e-commerce protocol execution, (some of) the sessions of *alice* together with (some of) the sessions of *bob*;
- the group of agents that instantiate different protocol-roles, e.g., in an e-banking protocol execution, (some of) the sessions of the server together with (some of) the sessions of the client.

Some of the groups above more than others seem the natural candidates to be considered as sets of agents which share information in order to detect an attack. A first compelling candidate is the group Gr of agents that represent the same participant. This group models the concurrent protocol sessions run by the same user on the same machine. But, one might wish to consider only the subgroup of Gr where agents represent the same participant and play the same protocol role. The group may model a software client that collects information from its different sessions (e.g, a user running two client sessions of the SSH protocol, each in a different terminal). In fact, one may even look at a group of agents that represent different participants (e.g., *alice* and *bob* are users in the same local network and share information easily). The last kind of group is most relevant from the point of view of a protocol attacker. The attacker would ideally be interested in a guarantee that he will remain undetected no matter what information is subsequently exchanged between any honest participants. Indeed, attacks that are in principle detectable by a certain group can lead to retaliations undesired by the attacker. As argued in [91], this fact can act as deterrence.

In this section we use temporal-epistemic logic to formalise subtle differences in the ability of a group Gr to detect attacks and other protocol failures. Are attacks eventually detectable on all possible future paths, just on some paths or on none? Are attacks detectable without the intruder knowing this? Can attack-related failures be immediately detected and can attacks therefore be detected prior to their actual completion? In future sections the meaning of these specifications will be explicitly denoted in the M_{IS} modelling and their corresponding formulae will be checked against such modellings.

Attack Detectability

We consider attack detectability specifications of the form:

$$AG\,(\beta \rightarrow \Box\,\beta) \tag{6.2}$$

where the *modality* $\Box$ is generated by the grammar:

$$\Box \ ::= \ D_{Gr} \mid AF\,D_{Gr} \mid EF\,D_{Gr} \mid \neg\Box.$$

In the schema (6.2), Gr denotes a (detectability-relevant) group of agents and the *condition* β expresses that a state "undesirable" to the group Gr has been reached. In the most basic case, $\beta ::= \neg p$. Then, condition β states that there has been an attack on an intended security goal p. In the following, we will also consider more complex conditions β.

We say that condition β is *instantly detectable by the group Gr* if whenever β holds the group Gr knows this:

$$AG\,(\beta \rightarrow D_{Gr}\,\beta) \tag{6.3}$$

We say that condition β is *eventually detectable by the group Gr* if whenever β holds the group Gr will eventually know this:[1]

$$AG\,(\beta \rightarrow AF\,D_{Gr}\,\beta) \tag{6.4}$$

[1]It is a customary assumption in the Dolev-Yao model that the intruder is able to block messages. However, this is not reasonable for all protocols, e.g., wireless protocols. Thus, we sometimes restrict the path quantifiers to paths that are *fair* in the sense that agents complete all their protocol steps.

We say that condition β is *possibly detectable by the group Gr* if whenever β holds it is possible that the group Gr will eventually know this:

$$AG\,(\beta \rightarrow EF\, D_{Gr}\, \beta) \tag{6.5}$$

We say that β is *instantly undetectable by the group Gr* if whenever β holds the group Gr does not know this:

$$AG\,(\beta \rightarrow \neg D_{Gr}\, \beta) \tag{6.6}$$

We say that β is *possibly undetectable by the group Gr* if whenever β holds it is possible that the group Gr will never know this:

$$AG\,(\beta \rightarrow \neg AF\, D_{Gr}\, \beta) \tag{6.7}$$

We say that β is *forever undetectable by the group Gr* if whenever β holds the group Gr will never know this:

$$AG\,(\beta \rightarrow \neg EF\, D_{Gr}\, \beta) \tag{6.8}$$

Specifications (6.3)–(6.8) exhaust the modalities $\Box$ up to logical equivalence. As Figure 6.1 shows, the logical-strength decreases in specifications (6.3) to (6.5) while it increases in (6.6) to (6.8). Moreover, for detectability specifications (6.3)–(6.5) the logical-strength decreases with larger groups Gr. On the other hand, for (un)detectability specifications (6.6)–(6.8) the logical-strength decreases with smaller groups Gr.

Simple Attack-Detectability. The most basic conditions β are of type $\beta ::= \neg\,\Gamma$. This denotes that some security requirement p from a selected set Γ of requirements fails. In specifications, we identify the set Γ with the conjunction of the requirements included in the set, i.e., if $\Gamma = \{p_1, \ldots, p_n\}$ then, in specifications, $\Gamma{:=}p_1 \wedge \ldots \wedge p_n$. The detectability schema (6.2) then becomes:

$$AG\,(\neg\,\Gamma \rightarrow \Box\,\neg\,\Gamma) \tag{6.9}$$

stating that attacks on the security requirements Γ are instantly/eventually/possibly detectable or instantly/possibly/forever undetectable by the group Gr.

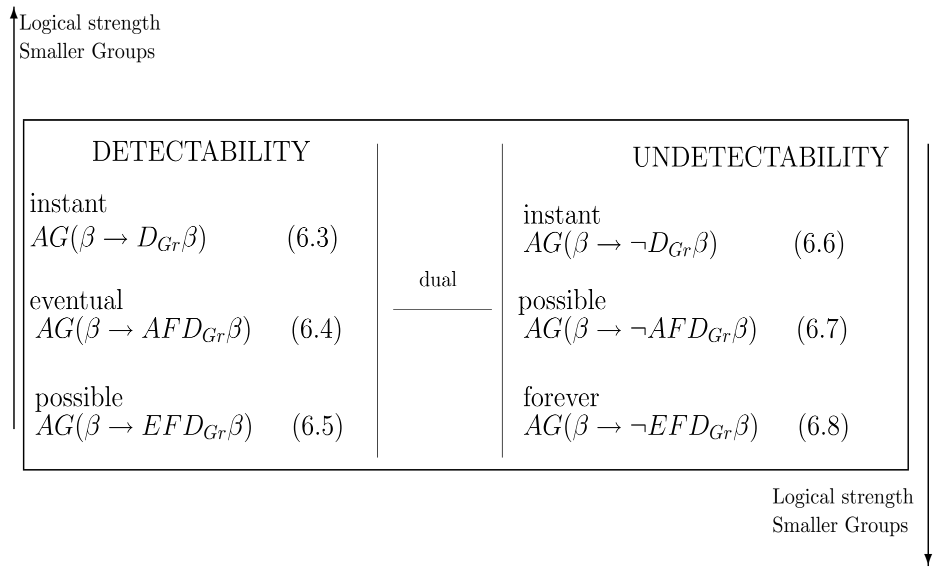

Logical strength
Smaller Groups
DETECTABILITY
instant
$AG(\beta \rightarrow D_{Gr}\beta)$ (6.3)
eventual
$AG(\beta \rightarrow AFD_{Gr}\beta)$ (6.4)
possible
$AG(\beta \rightarrow EFD_{Gr}\beta)$ (6.5)
dual
UNDETECTABILITY
instant
$AG(\beta \rightarrow \neg D_{Gr}\beta)$ (6.6)
possible
$AG(\beta \rightarrow \neg AFD_{Gr}\beta)$ (6.7)
forever
$AG(\beta \rightarrow \neg EFD_{Gr}\beta)$ (6.8)
Logical strength
Smaller Groups

Figure 6.1 Attack Detectability Schemata

The selected set Γ of requirements might include only the intended security goals of the protocol under analysis. However, it may be of interest to include also pre-goals in Γ. If we indeed include such a pre-goal in Γ, the failure of an instance of the schema (6.9) signals that the intruder is mounting an attack and the protocol will not complete successfully.

Culprit Detectability. Some attacks on security protocols (e.g., see Chapter 2, page 35 for impersonation attacks) require that the intruder is an insider. This means that the intruder knows a valid outset the private keys and/or signatures. If some honest agents can detect the identity of this corrupt participant then possible retaliation actions can be more precisely targeted, strengthening the deterrence.

To give this kind of detectability, we consider a state "undesirable" to the group Gr to be a state where condition $\beta ::= \neg\,\Gamma \land \mathit{corrupt}(v)$ holds. The atomic proposition $\mathit{corrupt}(v)$ holds if the intruder started the current run knowing the private key associated to some identity v. More precisely, when generating the M_{IS} models, the user-input always has the *intruderHonest* enabled (refer to Chapter 5, page 198). For ease of presentation, we assume that there is at most one corrupt participant in any run of an M_{IS} system on which $\mathit{corrupt}(v)$ is evaluated. Furthermore, as previously described in M_{IS}, assume that the name of the corrupt participant is randomised in each run. In this way the corrupt identity is initially unknown to honest agents.

By uniformly replacing the generic condition β with the specific $(\neg\,\Gamma \wedge \mathit{corrupt}(v))$, schema (6.2) gives the sub-schema for culprit detectability, i.e., schema (6.2) gives instances of (6.3)–(6.8) in which the condition $\beta ::= \neg\,\Gamma \,\wedge\, \mathit{corrupt}(v)$ states that a certain participant identity v is corrupt. In particular, the condition $(\neg\,\Gamma \wedge \mathit{corrupt}(v))$ is possibly detectable if:

$$AG\,(\neg\,\Gamma \wedge \mathit{corrupt}(v) \rightarrow EF\, D_{Gr}\,(\neg\,\Gamma \wedge \mathit{corrupt}(v))) \tag{6.10}$$

i.e., whenever requirements Γ are attacked and v is corrupt it is possible for group Gr to know this eventually.

Nested Attack-Detectability. It seems reasonable to assume that, for $\beta ::= \neg\,\Gamma$, possible culprit detectability (6.10) or even the weaker possible attack-detectability (6.5) achieve an adequate deterrence in many possible scenarios. Nevertheless, let us make the additional assumption that the intruder is theoretically able to know if and when he has been detected. This could influence future actions of the intruder. For instance, the attacker could prepare for possible retaliations. Or, if the attacker is assured that he will always know when he is detected, then he may be more willing to mount attacks.

Thus, we consider instances of schema (6.2) where the states of interest are those in which an attack is detectable, i.e., those states defined by the following β:

$$\beta \;\; ::= \;\; \Box\,\neg\,\Gamma \mid \Box(\neg\,\Gamma \wedge \mathit{corrupt}(v)),$$

where the attack-modality $\Box$ includes no negations. Intuitively, β defines states that are "undesirable" from the point of view of an attacker. To illustrate, consider the states given by $\beta ::= D_{Gr}\neg\,\Gamma$. These are states where an attack on requirements Γ is detectable by a group Gr of "honest" agents, i.e., states "undesirable" to the intruder. At its turn, condition β is instantly detectable by another group Gr' of "hostile" agents if:

$$AG\,(D_{Gr}\neg\,\Gamma \rightarrow D_{Gr'}\, D_{Gr}\,\neg\,\Gamma). \tag{6.11}$$

In (6.11), whenever the "honest" group Gr knows that there is an attack on requirements Γ, the "hostile" group Gr' knows that the "honest" group Gr knows this. Another more complex instance of schema (6.2) refers to states given by $\beta ::= AF\, D_{Gr} \neg\, \Gamma$. At these states, an attack on requirements Γ is eventually detectable by the group *Gr* of honest agents, i.e., these states are "undesirable" to the intruder. Then, condition β is instantly detectable by another group Gr' of "hostile" agents if:

$$AG\,(AF\, D_{Gr} \neg\,\Gamma \rightarrow D_{Gr'}\, AF\, D_{Gr}\, \neg\,\Gamma). \tag{6.12}$$

In (6.12), whenever it is the case that the "honest" group *Gr* will eventually know that there is an attack on requirements Γ, the "hostile" group Gr' knows that the "honest" group *Gr* knows this.

In the above, we specified the "hostile" group Gr' being able to detect states that are "undesirable" to the members of the group. However, it is of interest to consider whether the "hostile" group Gr' can also detect states that are "desirable" from its perspective. For instance, consider the states in which attacks are undetectable by the "honest" group *Gr*. To this end, we assume conditions β comprising an attack-modality $\Box$ with an odd number of negations. For example, consider the states given by the condition $\beta ::= (\,\neg(EF\, D_{Gr})\,)\neg\,\Gamma$. These are states "desirable" to the intruder as in these states an attack on requirements Γ is forever undetectable by the group *Gr* of "honest" agents. Then, this condition β is eventually detectable by the group Gr' of "hostile" agents if:

$$AG\,(\,\neg(EF\, D_{Gr})\,\neg\,\Gamma \rightarrow AF\, D_{Gr'}\; \neg(EF\, D_{Gr})\; \neg\,\Gamma). \tag{6.13}$$

In (6.13), whenever it is the case that the "honest" group *Gr* will never know that there is an attack on requirements Γ, the "hostile" group Gr' eventually knows that the "honest" group *Gr* will never know this.

Attack-launch Detectability

It might be the case that neither attacks on intended goals Γ nor attacks on pre-goals Γ' are detectable and not even possibly detectable. Nevertheless, attacks on the intended goals Γ might be observable in a weaker sense: an attack on the intended goals Γ cannot be mounted without the group *Gr* at least learning in the meanwhile that some pre-goal in Γ' fails.

We say that *attacks on pre-goals* Γ' *signal attacks on intended goals* Γ *to the group* Gr if requirements Γ hold at least until the group Gr knows that requirements Γ' fail:

$$A\,(\Gamma\ W\ D_{Gr}\neg\,\Gamma') \tag{6.14}$$

Note that attack-launch detectability (6.14) does not reduce to an attack detectability specification (6.2). In particular, the formulae implied by (6.14) may hold even if attacks on the pre-goals Γ' are not *possibly detectable* by the group Gr as per (6.5).

Attacks on the pre-goals Γ' signaling attacks on the intended goals Γ can be beneficial for protocol design/synthesis. More precisely, the flawed protocol can be “patched” with the epistemic test $D_{Gr}\neg\,\Gamma'$. For instance, one such amended protocol is that where the agents in the group Gr abort as soon as $D_{Gr}\neg\,\Gamma'$ is realised. Several attack-prone protocols have been “patched” in similar ways [136] by pen-and-paper inspection of flaws. However, in those cases, the tests for comparing values across participants were non-epistemic and un-systematised. Manually finding the appropriate non-epistemic tests can be non-trivial. By contrast, the relevant set Γ' of pre-goals can be derived automatically with little cost from the protocol description, as illustrated in Section 6.1. As next sections will show, these can then be manipulated to generate the epistemic test automatically.

In this section we have used temporal-epistemic logic to formalise subtle differences in the ability of a group Gr to detect attacks and other protocol failures. Taxonomies of attack detectability and attack-launch detectability have thus been introduced. In the next section, the relevance of these will be shown on a variant of the Kerberos protocol.

6.3 Case Study: the KSL Protocol

In this section we illustrate a case study on detectability of assaults against KSL [111], a variant of the Kerberos protocol. The next sections report on employing an extension of `PD2IS` for automatic analysis of detectability specifications. Some of the results presented in this section were provided by the automatic methodology aforementioned.

The KSL protocol was designed with two distinct levels. The first level is a *key-establishment* *level*. We present this level in Example 6.3.1. In this level, two parties A and B use a server S to establish a session key K_{AB} and a ticket $\{T, A, K_{AB}\}_{K_{BB}}$.

Example 6.3.1 (The First Level of the KSL Protocol)

```
1'.  A ->  B  : Na, A
2'.  B ->  S :  Na, A, Nb, B
3'.  S ->  B :  {Nb, A, Kab}Kbs, {Na, B, Kab}Kas
4'.  B ->  A :  {Na, B, Kab}Kas, {Tb, A, Kab}Kbb, Nc, {Na}Kab
5'.  A ->  B :  {Nc}Kab
```

We refer to [111] for details. The key and ticket are employed in the second level in order to mutually authenticate A and B in three steps. This second level is a *repeated-authentication level*, i.e., it can be run several times, until the ticket expires. Example 6.3.2 shows the second level of the KSL protocol.

Example 6.3.2 (The Second Level of the KSL Protocol)

```
1. A -> B : Ma, {Tb,A,Kab}Kbb
2. B -> A : {Ma}Kab, Mb
3. A -> B : {Mb}Kab
```

As Example 6.3.2 shows, in step 1, participant A sends to B a challenge-nonce Ma and the ticket $\{Tb, A, Kab\}Kbb$ established during KSL's first level. This ticket contains a timestamp Tb, the identity A and the session key Kab which was also established during the first level of KSL. All these are encrypted with Kbb, a key known only to B. If the timestamp Tb inside the ticket received by B has not yet expired, then B responds in step 2 by encrypting the nonce Ma using the pre-established session key Kab. B also sends along a challenge-nonce Mb of his own. Then, in step 3, A returns Mb encrypted with the session key Kab.

The purpose of the second level in KSL is to ensure that when an agent playing the role of B completes all its steps, it shares the nonces Ma and Mb with its intended communication partner. Therefore, we consider the intended authentication goal $auth@3(ag, \{Ma, Mb\})$ for each agent ag playing the B-role in the second-level.

The Hwang Attack. The following attack [106] on the repeated authentication level is due to Hwang:

```
i.1.   I(A) -> B : Ma, {Tb,A,Kab}Kbb
i.2.   B  ->  I(A): {Ma}Kab, Mb
ii.1.  I(A) -> B :  Mb, {Tb,A,Kab}Kbb
ii.2.  B -> I(A) :  {Mb}Kab, Mb'
i.3.   I(A)  -> B :  {Mb}Kab
```

This attack assumes a first-level session completed between participants *alice* and *bob*, playing an A-role and a B-role respectively. On these grounds, the intruder impersonates *alice* and initiates a second-level session i with *bob*. The intruder has previously intercepted a B-role ticket in the preceding first-level communication and, in step $i.1$, sends this ticket together with a nonce Ma to *bob*. In step $i.2$, *bob* responds according to his honest B-role by sending the nonce Ma encrypted with the session key Kab and the challenge-nonce Mb to his purported A-role partner. Instead of completing this session, the intruder impersonates *alice* again and initiates yet another second-level session ii with *bob*. In this second session ii, the intruder will actually use *bob* as an oracle to encrypt his own challenge-nonce Mb from the initial i session. The intruder finally inserts the oracled encryption $\{Mb\}Kab$ back into session i as if it were coming from *alice*, thus completing *bob*'s first second-level session in step $i.3$.

Informally, *bob* is fooled into believing that in session i he is sharing the values of Ma and Mb with *alice*, while in fact he has been constantly interacting with the intruder impersonating *alice*. Formally, KSL's authentication goal $auth@3(ag, \{Ma, Mb\})$ fails for *bob*'s agent ag engaged in session i.

Following the formalism presented in Chapter 3, we consider an M_{IS} model $\mathcal{I}$ to formalise concurrent sessions of KSL (comprising at least the scenario needed for mounting the Hwang attack above). We discuss detectability of KSL's failures on the model $\mathcal{I}$.

Attack-Launch Detectability. On the M_{IS} model $\mathcal{I}$, we can show that participant *bob* can detect preparations for the attack prior to its actual completion. More precisely, attacks on the pre-goal of authentication upon Ma at step 1 signals to *bob* the existence of attacks on the intended goal of authentication upon Ma and Mb at step 3. Formally, the intended authentication goal holds at least until *bob*'s agents collectively have sufficient information to infer that in one session, one of these agents received a replayed nonce instead of an expectedly fresh value for Ma. In other words, the corresponding attack-launch detectability formula that holds on the IS model $\mathcal{I}$ is:

$$A\,(auth@3(\{Ma, Mb\})\;W\;D_{bob}\,\neg\,auth@1(\{Ma\})) \tag{6.15}$$

where D_{bob} is the distributed knowledge modality D_{Gr} for the group Gr of agents representing *bob* and playing the B-role in the second level, the notation $auth@3(\{Ma, Mb\})$ abbreviates the set $\{auth@3(ag, \{Ma, Mb\}) \,|\, ag \in Gr\}$ of intended goals, and $auth@1(\{Ma\})$ is like $auth@3(\{Ma, Mb\})$ except that it refers to pre-goals.

To explain, the group Gr of *bob*'s agents playing a *B-role* can theoretically compare the value of the nonces Mb in each group-member with the value of Ma in each other group-member; there has been an attack on $auth@1(\{Ma\})$ if two values collide. In particular, the epistemic test $D_{bob}\,\neg\,auth@1(\{Ma\})$ can be reduced to the non-epistemic test:

$$\exists\, ag, ag' \in G \,|\, ag.Mb = ag'.Ma$$

which in turn translates (6.15) into a purely temporal property. However, such a reduction (e.g, translating $D_{bob}\,\neg\,auth@1(\{Ma\})$ into a non-epistemic test) is specific to this particular protocol and may be non-trivial to determine. By contrast, attack-launch specifications like (6.15) can be generated automatically as explained in Section 6.2 and later shown in Section 6.5.

Eventual Attack-Detectability. Despite the attack-launch detectability in specification (6.15), attacks on the intended goals $auth@3(\{Ma, Mb\})$ are not instantly detectable by *bob* in the sense of specification (6.3). In the M_{IS} model $\mathcal{I}$, the attack-trace $i.1$–$i.3$ for the Hwang attack above is indistinguishable to *bob* from an execution trace in which the intruder does not impersonate *alice*. This counterexample trace of system $\mathcal{I}$ is shown below:

Example 6.3.3 (Counterexample E1)

Trace of the M_{IS} model $\mathcal{I}$, counterexample for $D_{Gr}\neg auth@3(\{Ma, Mb\}))$

```
i'.1   A -> B: Ma, {Tb,A,Kab}Kbb
```

```
i'.2   B -> A: Mb, {Ma}Kab
ii'.1  I(A) -> B:  Mb, {Tb,A,Kab}Kbb
ii'.2  B -> I(A): Mb', {Mb}Kab
i'.3   A -> B: {Mb}Kab
```

We briefly explain the above execution exhibited in the model $\mathcal{I}$. As in the Hwang attack, the nonce Mb of *bob*'s agent from the first session i' is replayed to the agent representing *bob* in another session ii'. The trace is therefore indistinguishable to *bob* from the Hwang attack. Note that, in session i' of the counterexample-execution, participant *bob* actually authenticates himself correctly upon values Ma and Mb with *alice*.

Nonetheless, we can show that attacks on the intended goals $auth@3(\{Ma, Mb\})$ are eventually detectable by *bob*. In other words, the following eventual attack detectability formula holds on the IS model $\mathcal{I}$:

$$AG\,(\neg\, auth@3(\{Ma, Mb\}) \rightarrow AF\, D_{bob} \neg\, auth@3(\{Ma, Mb\})),$$

where the AF-modality quantifies only over execution paths that are fair in the sense that each agent eventually completes all its steps.

We briefly explain the eventual detectability by *bob* in the above. Assume that *bob*'s agent which was used as an oracle in session ii of the Hwang attack has completed his execution. By comparing the nonce received in this session with the nonce generated in the first session i by another agent of his, participant *bob* can infer that the nonce received did not originate from his purported partner *alice*. In other words, *bob*'s agent collectively know that he has completed session ii without agreeing with *alice*, i.e., *bob* knows that the intended authentication goal has been attacked.

The Lowe Attack. The discussion above assumed that a KSL participant does not engage in the second level prior to completing all the steps of the first level. However, KSL has also been interpreted [136] as a protocol that allows a participant playing the B-role to engage in a second-level session as soon as a ticket has been established, hence before actually completing the final challenge-response step of the protocol's first-level. We refer to [136, 111] for details.

As shown by Lowe, this opens up for more attacks onto KSL. For instance, *alice* can be made to generate a B-role ticket $\{Ta, bob, Kab\}Kbb$ and *bob* can also be made to generate another B-role ticket $\{Tb, alice, Kab\}Kbb$. The irregularity in this is that both these tickets are based simultaneously on the same session key Kab. Thus, the tickets can be used by the intruder to mount a more intricate impersonation attack on the second level:

```
i.1.    I(A) -> B :  {Tb,A,Kab}Kbb, Ma
i.2.    B -> I(A) :  Mb, {Ma}Kab
ii.1.   I(B) -> A : {Ta,B,Kab}Kaa, Mb
ii.2.   A -> I(B) :  Ma', {Mb}Kab
i.3.    I(A) -> B :  {Mb}Kab
```

The attack begins like the Hwang attack above: the intruder impersonates *alice* to enter the second-level session i with *bob*. However, in order to encrypt the challenge-nonce Mb with the session-key Kab the intruder does not use another agent representing *bob* as oracle like in Hwang attack. Instead, based on the fact that the same key Kab is part of two distinct tickets, in step $ii.1$ the intruder engages an agent representing *alice* and playing the B-role. The intruder finally completes session i with *bob* in step $i.3$ by forwarding nonce Mb as previously encrypted by *alice* in $ii.2$. We refer to [106, 119] for details.

Like in the Hwang attack, the agent representing *bob* in session i is fooled into believing that he is sharing the values Ma and Mb with *alice*.

We consider now an M_{IS} model $\mathcal{I}$ formalising KSL multi-session executions upon the relaxed interpretation in [136] described above, i.e., it is possible to initiate second-level KSL runs prior to the full completion of all first level KSL sessions.

Instant Attack-Detectability. On the M_{IS} model $\mathcal{I}$, we can show that the Lowe attack is instantly detectable by a subgroup of agents representing participants *bob* and *alice* together. More precisely, the group Gr of all second-level agents can instantly detect attacks on the intended authentication goals. In other words, the following detectability formula holds on the M_{IS} model $\mathcal{I}$:

$$AG\,(\neg auth@3(\{Ma, Mb\}) \rightarrow D_{Gr}\,\neg auth@3(\{Ma, Mb\}))$$

We explain the specification above. By comparing values between group members, the group can deduce whether any of the group-members is fooled into completing the second-level without agreeing with another agent in the group.

Possible Attack-Undetectability. On the M_{IS} model $\mathcal{I}$, we can show that the Lowe attack is possibly undetectable by participant *bob* alone. More precisely, attacks on the intended goals $auth@3(\{Ma, Mb\})$ for *bob*'s agents are possibly undetectable by the common effort of all *bob*'s agents. In other words, the following (un)detectability formula holds on the IS model $\mathcal{I}$:

$$AG\,(\neg auth@3(\{Ma, Mb\}) \rightarrow \neg AF D_{bob}\, \neg auth@3(\{Ma, Mb\})).$$

Under the MAS model $\mathcal{I}$, if the intruder follows the strategy of the Lowe attack[2] the resulting trace is indistinguishable to the group of *bob*'s agents from a trace in which there is no attack on the intended authentication goal. Unlike during the Hwang attack, in the Lowe attack the challenge-nonce Mb of one of *bob*'s agents is replayed to an agent representing *alice* rather than back to one of *bob*'s agents. Thus, there is no way for *bob* to observe this replay even at later stages.

However it is not the case that attacks on the intended goals $auth@3(\{Ma, Mb\})$ are forever undetectable by *bob* in the sense of (6.8). Since the Hwang attack is possible also under this modelling of the relaxed interpretation of KSL, there exists a path where *bob*'s undetection is eventually refuted.

Nested Detectability. On the M_{IS} model $\mathcal{I}$, we can show that even though attacks on the intended goals are not always undetectable by *bob*, whenever they are undetectable the intruder knows that this is the case. In other words, the following (nested) detectability formula holds on the IS model $\mathcal{I}$:

$$AG\,(\neg D_{bob}\, \neg auth@3(\{Ma, Mb\}) \rightarrow D_I\, \neg D_{bob}\, \neg auth@3(\{Ma, Mb\}),$$

stating that the states given by $\neg D_{bob}\, \neg auth@3(\{Ma, Mb\})$, "desirable" from the intruder's point of view are instantly detectable in the sessions of the intruder (i.e., $D_I \ldots$ in the formula above) .

In this section we have illustrated a case study on detectability of assaults against KSL [111], a variant of the Kerberos protocol. The next section will show an extension of `PD2IS` for automatic analysis of detectability specifications.

[2]Under a temporal-epistemic framework, this can be specified using fairness constraints.

6.4 PD2IS Extended: A Toolkit for Detectability-Analysis

In Chapter 5 we presented PD2IS (Protocol Descriptions to Interpreted Systems), an open-source compiler from CAPSL (Common Authentication Protocol Specification Language) protocol descriptions into ISPL (Interpreted System Programming Language), the input language of the model checker MCMAS [127]. In this section we report on the extension of PD2IS to generate the models and the formulae to support the methodology for detectability. Given a CAPSL description of a protocol Pr together with an instantiation of the protocol Pr, the extended PD2IS generates the pre-goals, auxiliary predicates and all the possible detectability formulae upon schema (6.2) and schema (6.14), for all relevant groups Gr under the given instantiation.

Firstly, we extend the `data-setup` module to generate the set of all the detectability-relevant groups mentioned in Section 6.2. Groups are indexed by name of participants and by role. This is possible simply by manipulating the data structures generated by the modules called previously to `data-setup`. In this extension, `predicate`, `assertion` and `formula` objects are indexed by groups (i.e., not only by agents). For example, in a model for NSPK, the aliveness of the *B-role* participant *bob* is "of interest" not only to one *A-role* agent, but also to the group of 2 agents of an *A-role*, to the group of 3 agents of an *A-role*, to the group of all agents of an *A-role*, to the group of all agents representing *alice*, etc.

Secondly, predicates that are related to the detectability schemata (i.e., $culprit(v)$ for all $v \in \mathcal{R}_{\mathbf{I}}$) are now generated by the `data-setup` model. Importantly, predicates to denote all pre-goals of the stipulated goals are also added. In doing so, we follow the guidelines presented in Section 6.1. Also, each pre-goal predicate is indexed by its originating goal. Intuitive names are given to all these predicates to ease the generation and the reading of the final detectability formulae. Pre-goals are also categorised by the groups for which they are meaningful (e.g., in KSL, $auth@1(\{Ma\})$ should not be indexed by agents of the *S-role*).

The extended `producer` module makes use of the fact (presented in Chapter 5) that we designed a hierarchy of *Assertion* and *Formulae* classes. For instance, for each group we select all CTLK injective agreement formulae that refer to the index of that group (i.e., the index of the group is the *subject* of the role *Assertion* object; see Chapter 5, page 201). Then, for each group we select all

CTLK non-injective agreement formulae[3] that refer to the index of that group, etc. In this fashion, the extended PD2IS generates a systematically structured list of detectability formulae. To give but a sense of the number of formulae generated, for the fixed scenario with 3 honest agents on the NSPK protocol we generate 160 detectability formulae. In Example 6.4.1, we illustrate a small excerpt of this set of formulae.

Example 6.4.1 (Detectability Formulae Generated with PD2IS — Excerpt)

```
 --Formula * number 1 * instant detectb. by roleA *
 AG (
!(AG(!(terminationOfinstance1 and honestInterlocutorOfinstance1_into_instance2) #
or free_agree_instance2_instance1)) or
DK(roleA,!(AG(!(terminationOfinstance1 and honestInterlocutorOfinstance1_into_instance2)
or free_agree_instance2_instance1))));
...

--Formula * number 26 * always-eventually detectb. by alice *
 AG (!(AG(!(terminationOfinstance1 and honestInterlocutorOfinstance1_into_instance2)
or free_agree_instance2_instance1)) or
AF DK(alice,!(AG(!(terminationOfinstance1 and
honestInterlocutorOfinstance1_into_instance2) or free_agree_instance2_instance1))));
...

--Formula * number 36 * forever undetectb. by G2Members_2 *
 AG (!(AG(!(terminationOfinstance1 and honestInterlocutorOfinstance1_into_instance2)
or free_agree_instance2_instance1))
or !(EF DK(G2Members_2,!(AG(!(terminationOfinstance1 and
honestInterlocutorOfinstance1_into_instance2) or free_agree_instance2_instance1)))));
...

--*ATTACK-LAUNCH DETECTABILITY*

 --Formula * number 49 * injection as attack--launch  detectb. by roleA *
AG(!(terminationOfinstance1 and honestInterlocutorOfinstance1_into_instance2)
or free_agree_instance2_instance1) or
A((!(terminationOfinstance1 and honestInterlocutorOfinstance1_into_instance2)
or free_agree_instance2_instance1)
     U
(DK(roleA, (EF! intermAgreementOnSomeValuesOfRoleA and ! purpotedReceiverForRoleA))));
...

--*CULPRIT DETECTABILITY*

 --Formula * number 65 * culprit instant detectb. by roleA *
AG(AG(!(terminationOfinstance1 and honestInterlocutorOfinstance1_into_instance2)
or free_agree_instance2_instance1) or DK(roleA,(! (AG(!(terminationOfinstance1
```

[3]For our CTLK expressions of non-injective agreement, see page 50 and page 201.

```
and honestInterlocutorOfinstance1_into_instance2) or
free_agree_instance2_instance1)) and (culpritBgreg))));
...

 --Formula * number 160 * culprit forever undetectb. by G2Members_0 *
AG(AG(!(terminationOfinstance1 and honestInterlocutorOfinstance1_into_instance2)
or free_agree_instance2_instance1) or !(EF DK(G2Members_0,(!
(AG(!(terminationOfinstance1 and honestInterlocutorOfinstance1_into_instance2)
or free_agree_instance2_instance1)) and (culpritBgreg)))));
```

In the case of detectability analysis, we generate un-constrained M_{IS} models for incrementally larger fixed scenarios. To ease the interpretation of the large result-space, we extend the testing procedures with a BASH script that classifies the counterexamples by groups (e.g., *A-role*, *bob*, etc.) and by type of detectability (e.g., eventual, instant, etc.). This is possible by automatically parsing the output of MCMAS in comparison with the ISPL file given at input; in the latter, the extended PD2IS producer module inserts before each formula a comment that explicates it (as the excerpt above shows). Then, if MCMAS reports that a certain formula fails, our script can automatically provide details on the failure, using the information in the ISPL comment in front of the formula. For instance, corresponding to formula 160 in the ISPL excerpt above, the script will report that detectability of culprits is forever undetectable by the group *Gr* formed with 2 of *alice*'s agents. The script is available with PD2IS [28].

To sum up, this section has explained some of the details behind an extension of PD2IS for automatic analysis of detectability specifications.

6.5 Experiments on Model Checking Detectability

In this section we report on automatically analysing detectability in a variety of protocols, using the extension of PD2IS presented in the previous section. The results presented generalise the ones discussed for KSL in the preceding section.

We selected a set of well known authentication and key establishment protocols from the SPORE repository [119]. Given a CAPSL protocol description we used PD2IS extended detectability-wise to generate a corresponding ISPL file for each loose scenario with four or fewer sessions; each generated ISPL file was then passed to MCMAS for verification. For each protocol we selected the models that exhibited attacks, and analysed the output of `MCMAS` for what concerns satisfiability of the detectability specifications. Some of these results are summarised in Table 6.1.

		Det	Undet	Launch	Culprit
Scenario	Groups				
NSPK	a	F	possible	F	possible
$a^1 \rightarrow I$	b^1	F	forever	F	F
$a^1 \leftarrow b^1$	a^1, b	eventual	possible	T	eventual
$a^2 \rightarrow b^1$	a, b	eventual	possible	T	eventual
WMF	a	F	forever	N/A	N/A
$a^1 \rightarrow S^1$	b	F	forever	N/A	N/A
$S^1 \rightarrow b^1$	a, b	eventual	instant	N/A	N/A
$I \rightarrow b^1$	S, b	eventual	possible	T	eventual
A. S-RPC	a	instant	F	F	N/A
$a^1 \rightarrow b^1$	a^2	F	possible	N/A	N/A
$a^2 \rightarrow b^2$	b	F	forever	F	N/A
	a^2, b^2	instant	F	T	N/A
KSL	a	F	forever	F	N/A
$a^1 \xrightarrow{1} b^1$	a^1	F	forever	F	N/A
$a^1 \xrightarrow{1} S$	b	F	possible	T	N/A
$a^1 \xleftarrow{2} b^1$	a^1, b	eventual	instant	T	N/A
$a^1 \xleftarrow{2} b^2$	a^2, b	F	possible	T	N/A
$a^2 \xrightarrow{2} b^3$	a, b	eventual	F	T	N/A

Table 6.1 Attack and Attack-Launch Detectability Results.

In Table 6.1, the first column denotes the fixed scenario considered (see below); the second column denotes the detectability group Gr considered; the third column reports on whether attacks were found to be detectable instantly, eventually, possibly, or never detectable ("F") by the group Gr considered; the fourth column reports on whether attacks were found to be undetectable instantly, possibly, or forever detectable; the fifth column states the satisfaction of detectability of attacks on pre-goals where it applies; the last column reports the results for satisfaction of culprit detectability formulae. “N/A” signifies that the corresponding detectability specification does not apply for the corresponding protocol or the corresponding group.

In more detail, in the first column the notation $a^1 \rightarrow I$ indicates that the NSPK scenario considered includes an agent a^1 representing participant *alice* in session 1 communicating with a corrupt insider I. Analogously, $a^2 \rightarrow b^2$ indicates that the scenario also includes an agent a^2 representing participant *alice* in session 2 communicating with an agent representing *bob*. The notation $\xrightarrow{i}$ in the KSL scenario denotes a communication direction within level i of the protocol.[4] The letter S in the WMF scenario represents the server identity. The letters a and b in the second column denote the groups of all agents representing *alice* and *bob* respectively.

Table 6.1 does not report the verification time of the toolkit. These are aligned with the attack-finding verification times reported in Chapter 5. As an example, checking 218 detectability specifications against an un-constrained model (cf. Chapter 5) for NSPK with 4 agents in a fixed communication setting enabling the Lowe attack [135] took approximately 27 seconds on an Intel Core 2 Duo clocked at 2.26GHz with 2.9 GB of memory, running Linux kernel 2.6.27.7. The corresponding number of reachable global states was in the region of 10^3.

While the discussions above were grounded on authentication and key-establishment protocols (directed by the use of our model M_{IS} and toolkit `PD2IS`), the methodology of detecting protocol failures easily transfers to IS models for other classes of protocols (e.g., group protocols, contract-signing protocols, e-voting protocols, etc.).

In [91], a proof-of-concept framework was designed for the actual collaboration of protocol participants to mount retaliations against the intruder. The application thereby presented is the NSPK protocol, which is manually extended with a rule for the retaliation step. The data proposed for the retaliation-driven communication is defined ad-hoc by the specifier. The thread model is that of a *general attacker*, i.e., any participant can engage in the added-on retaliation rule against the standard, Lowe attack [135] on NSPK. In [91], LTL [165] is used to encode a notion of attack trace extended towards retaliations. No aspects of the participants’ knowledge is employed to achieve

[4]The table reports only on the classical interpretation of level interleaving in KSL (which does not allow for the Lowe attack).

the retaliation, communication being enforced onto specific participants upon manually designated data. We distinguish ourselves from this line in that we use a knowledge-centred, CTLK-based framework to investigate a systematic and automatic way to model and verify groups of honest agents theoretically detecting attacks and protocol failures in standard Dolev-Yao thread model. In our methodology, the aspects of detection on the intruder side are also explored.

In this chapter we have shown an extension to our multiagent systems and temporal-epistemic approach to protocol analysis. We introduce the concept of detectability, i.e., the theoretical ability of protocol participants to detect that a protocol attack has been or is being mounted. As a result, we advanced a novel, fully automatic methodology which analyses different flavours of detectability in a MAS setting, i.e., whether different groups of agents have the joint ability to detect protocol failures. This research line is a case in point that the AI-inspired verification techniques can provide new ways and insights into security protocol analysis and towards intruder detection systems [140]. Moreover, its results can be used further in protocol implementation, protocol patching, protocol synthesis and protocol design.

Chapter 7

MAS Models for Equationally Specified Security Protocols

Motto: "It is the theory that decides what can be observed."

(Albert Einstein)

The descriptions of equationally specified protocols rely on a set of primitives that obey certain axioms; such protocols were recalled at page 54. These are usually receiver-opaque protocols that are not reducible to receiver-transparent protocols. Thus, the Υ_{IS} formalism introduced in Chapter 3 cannot be readily used to model the execution of protocols specified under equational theories.

In this chapter we introduce a new multiagent system formalism designed to model communication protocols specified under convergent equational theories. We formulate a notion of quotient-knowledge *representing knowledge modulo the underlying equational theory. We extend the model to formalise agents theoretically "interrogating" the equational theory to rewrite terms, thereby supporting the interpretation of* interrogative knowledge, *a weaker form of knowledge. We show that interrogative knowledge coincides with quotient-knowledge in communication protocols modelled appropriately. We present an on-the-fly algorithm for model checking interrogative knowledge. An extension of* ***MCMAS*** *is used to verify equationally-specified electronic voting protocols against e-voting requirements expressed in logics of time and interrogative knowledge. Aspects of this material are presented in [30].*

Section 7.1 presents the necessary background on equational theories and fixes the notations.

Section 7.2 introduces Υ_{IS}^{E}, an interpreted system formalism for protocols described by an equational theory E. Unlike the formalisms in previous chapters, the Υ_{IS}^{E} formalism is designed for modelling receiver-opaque protocols. In the unwound model M_{IS}^{E}, the indistinguishability relation between states is denoted $\sim^{E}$ and is given by the equality of terms in the quotient term-algebra modulo E. In Section 7.2, we also introduce the *quotient-knowledge modality* QK to be interpreted in M_{IS}^{E} over $\sim^{E}$.

Section 7.3 introduces the generic formalism of *interrogative interpreted systems* to model executions of equationally specified protocols. In this formalisation, every agent is enriched with a set of *interrogations*, i.e., predicates related to the equational theory. In the unwound model, we define an *interrogative indistinguishability relation* denoted $\sim^{Intr}$ and given by the uniformity of the interrogations of an agent (i.e., two indistinguishable states are not differentiated by the values of the interrogations at those respective states). Section 7.3 also introduces the *interrogative knowledge modality* denoted $\mathbb{I}K$ and interpreted over $\sim^{Intr}$. We will compare and contrast the interrogative knowledge operator $\mathbb{I}K$ with other, related knowledge modalities in [164, 134].

Under a convergent equational theory, we prove that if an agent "interrogates" about all the normal terms then his interrogative knowledge coincides with his quotient-knowledge. Section 7.3.3 formalises this fact within the notion of *interrogative equational interpreted system* $\Upsilon_{IS}^{\mathbb{I}E}$.

Consider interpreted systems that generalise the $\Upsilon_{IS}^{\mathbb{I}E}$ formalism in that they do not necessarily encode security protocols, but their specifications contain interrogations for agents. In Section 7.4 we introduce a model checking algorithm of such generic interpreted systems against the interrogative knowledge modality.

Section 7.5 shows how to model the FOO'92 [87] e-voting protocol as *interrogative equational interpreted systems* $\Upsilon_{IS}^{\mathbb{I}E}$. The requirements of the protocol are specified in the newly introduced languages of time and quotient knowledge. Furthermore, we use the implementation of the methodology presented in Section 7.4 to verify different modellings of the the FOO'92 e-voting protocol. We report and discuss the results.

At the end of each section, we relate the material introduced to existing work in the field of protocol analysis and/or logic-driven verification.

7.1 Protocols Specified by Equational Theories

7.1.1 Preliminaries

Equational theories were reviewed in Chapter 2, page 45 and page 54. In this chapter we will use algebraic signatures (see Chapter 2 and/or Chapter 3) to formalise equational theories. The denotation of an equational theory is given by a universal algebra. In this preliminary section we fix the notations related to equational theories and used in this chapter.

Let S be a non-empty set of *sorts*. Let $\Sigma = \{\Sigma_{(\omega,s)} | \omega \in S^*, s \in S\}$[1] be an S-*sorted signature*, $\sigma \in \Sigma_{(\omega,s)}$ be a *function symbol* of *type* $[\omega, s]$, X be an S-sorted set of variables, $T_{\Sigma,X}$ be the S-sorted set of terms over X and Σ, and T_Σ be the set of *ground* terms. If $\sigma \in \Sigma_{(\omega,s)}$ is a symbol of type $[\omega, s]$, $n \geq 1$, $1 \leq i \leq n$, $t_i \in T_{\Sigma,X}$ are terms of sort $s_i \in S$, and $\omega = (s_1, \ldots, s_n)$, then $\sigma(t_1, \ldots, t_n) \in T_{\Sigma,X}$ is a term of sort s, i.e., $\sigma(t_1, \ldots, t_n) \in T_{\Sigma,X,s}$. The notation $t = t'$ denotes a Σ-*equation* of sort S, i.e., a pair (t, t') of terms of sort s. An *equational theory* is a tuple (Σ, E) where Σ is a signature and E is a set of Σ-equations. The S-*sorted* Σ-*algebra* $\mathbb{A} = (A, \Sigma^A)$ is the semantics of equational theories [124]. The set A is an indexed set of values, i.e., $A = (A_s \,|\, s \in S)$ For each sort in s, the set A_s is the called the *support-set* for s. The set Σ^A is a set of functions f^A from A_ω to A_s, corresponding to function symbols f in Σ, $f \in \Sigma_{(\omega,s)}$, $\omega \in S^*$, $s \in S$. The *term algebra* and the *ground-term algebra* implied by Σ and $\mathbb{A}$ are denoted $\mathbb{T}_{\Sigma,X}$ and $\mathbb{T}_\Sigma$, respectively. The indexed set $\delta = (\delta_s \mid s \in S)$ of maps is an *assignment* of X into the algebra $\mathbb{A}$; $\delta[x/_a]$ is the assignment obtained from δ when $\delta_s(x)$ is replaced by the value a, for $x \in X_s$, $a \in A_s$, $s \in S$. Let Δ be the set of assignments of variables in X into $\mathbb{A}$ and let each assignment $\delta \in \Delta$ be homomorphically extended to non-atomic terms. An equational theory E is *convergent* if the algebra of its semantics can be mechanised into a rewriting system $\rightarrow_E$ which is convergent (i.e., *confluent* and *terminating* [15]). Hence, for all terms $t \in T_{\Sigma,X}$ there exists uniquely a term $t' \in T_{\Sigma,Y}$ such that $Y \subseteq X$ and $t \rightarrow^*_E t'$. The term $t' \in T_{\Sigma,Y}$ as above is called the *normal form* of t and this is denoted as $t{\downarrow_E}$. For more information on equational theories and/or algebras, we refer the interested reader to [77, 76, 124].

[1] S^* stands for the free monoid generated by S.

7.1.2 Communication Protocols

In Chapter 2, page 54, the notion of equationally specified protocols was summarised. We recall that the protocols for which the descriptions rely on a set of axioms formed with the underlying primitives are denoted *equationally specified protocols*. In this section we detail how protocols can be specified by equational theories.

Example 7.1.1 illustrates an equational theory (Σ_1, E_1) which simulates some of Peano's axioms.

Example 7.1.1 (A Simple Equational Theory (Σ_1, E_1)**)**

Signature Σ_1*:*	*The set* E_1 *of* Σ_1*-Equations :*
Sorts: $S = \{nat, bool\}$*; Variables:* $X_{nat} = \{x, y\}$*;*	
Function Symbols:	$((\neg true) = false)$;
$\Sigma_{\lambda,bool} = \{true, false\}$;	$((\neg false) = true)$;
$\Sigma_{[bool,bool]} = \{\neg\}$;	$(\leq(0, x) = true)$;
$\Sigma_{[\lambda,nat]} = \{0\}$;	$(\leq(succ(x), 0) = false)$;
$\Sigma_{[nat,nat]} = \{succ\}$;	$(\leq(succ(x), succ(y)) = \leq(x, y))$;
$\Sigma_{[(nat,nat),bool]} = \{\leq\}$;	
$\Sigma_{[\omega,s]} = \emptyset$*, otherwise (i.e., for other* $\omega \in S^*, s \in S$*) ;*	

For illustration purposes, we designed a simple communication protocol described by the equational theory (Σ_1, E_1). Example 7.1.2 gives the Alice & Bob notation of this protocol.

Example 7.1.2 (A Simple Communication Protocol Pr_1**)**

$$1.\, A \rightarrow B : n$$

$$2.\, B \rightarrow A : m$$

$$3.\, A \rightarrow B : \leq(n, m)$$

The communication protocol Pr_1 in Example 7.1.2 describes two roles: the initiator role, i.e., the *A-role*, and the receiver role, i.e., the *B-role*. The initiator role starts by sending the term n to its communication partner. The receiver replies with some (possibly related) term m. Based on the data previously transmitted, the initiator proceeds by sending a final acknowledgement in the form of $\leq(n, m)$, where $\leq$ is the function symbol specified by the equational theory underlying the protocol description and given in Example 7.1.1. The goal of protocol Pr_1 is the authentication between participants of A-*role* and B-*role* upon the data n and m.

Let signature Σ_1 used in Example 7.1.2 be extended (at least) with a sort *role* (e.g., for Pr_1, variables of sort *role* are given by $X_{role} = \{A, B\}$). As in Chapter 3, assignments give a scenario for a multi-threaded execution of communication protocols. For instance, the assignment $\delta[n/3, m/2, A/alice, B/bob]$ for Pr_1 unwinds the execution of one session of protocol Pr_1. This session is illustrated in Example 7.1.3.

Example 7.1.3 (A Session of the Communication Protocol Pr_1)

$$\begin{aligned} &1.\, alice \rightarrow bob : &&3 \\ &2.\, bob \quad \rightarrow alice : &&2 \\ &3.\, alice \rightarrow bob : &&.F. \end{aligned}$$

Formally, let the denotation of an equational theory (Σ, E) be a Σ-algebra $\mathbb{A}$, together with a countable set Δ of assignments in $\mathbb{A}$. These underlie a multi-session execution of a protocol specified by the theory (Σ, E).

In the following sections we will use the equational theory (Σ_1, E_1) in Example 7.1.1, the protocol Pr_1 in Example 7.1.2 and the scenario/assignment δ in Example 7.1.3 for different illustration purposes.

The protocol in Example 7.1.2 is a communication protocol specified by an equational theory. However, no mechanisms are implied in Example 7.1.2 to enhance the reliability of the protocol Pr_1 in malevolent environments. In turn, security protocols are communication protocols where some enhanced security is provided by certain cryptographic mechanisms. At the abstract level, this simply means that the underlying equational theory of security protocols is formed with cryptographic primitives. Furthermore, the protocol specification language `CAPSL` supports the expression of security protocols through equational theories. This is achieved mainly by specifying the `TYPESPEC` section of a `CAPSL` description, i.e., in the `TYPESPEC` section of the `CAPSL` description, `AXIOMS` can be stipulated in order to encode equations. In fact, in Chapter 2, page 54 we recalled this precisely. Namely, in Example 2.2.4 we specified the equational theory underlying the Gong authentication protocol, i.e., we formulated the `TYPESPEC` section of the `CAPSL` description for the Gong protocol. Then, in Example 2.2.5 we showed how the equational theory is used in the actual description of the protocol. In Example 2.2.5, we gave the actual `PROTOCOL` section of the `CAPSL` description for the Gong protocol and, in doing so, we used details from its `TYPESPEC` section.

In the next sections we will mostly refer to security protocols specified by equational theories. These are security protocols where the underlying cryptographic primitives are more intricate than encryption/decryption (i.e., they are not included in the `CAPSL` `prelude`[2] package and the specifier needs to formalise these primitives as explicit `AXIOMS` in the `TYPESPEC` section of the corresponding `CAPSL` description). Security protocols specified by equational theories are usually receiver-opaque protocols which are not reducible to receiver-transparent protocols. In more detail, in such protocols it is of interest that the participants "take into consideration" even composites that they cannot fully decrypt or analyse down to atomic parts. Also, two messages or terms of the protocol might be syntactically different, yet equal modulo the underlying equational theory. Therefore, the equality indistinguishability relation between states used in Chapter 3 and Chapter 5 is too coarse to interpret a correct notion of knowledge modulo equational theories. Hence, the Υ_{IS} formalism introduced in Chapter 3 and used in Chapters 4, 5, 6 is generally not suited to modelling security protocols specified by equational theories. Thus, in the next sections we introduce a MAS model appropriate for modelling such security protocols.

7.2 Multiagent Systems Model for Equationally Specified Protocols

In this section we present an interpreted system formalisation denoted Υ_{IS}^{E}. It encodes executions of security protocols specified by a convergent equational theory (Σ, E). Like the M_{IS} model, introduced in Chapter 3, this formalisation also follows the MAS approach. Therefore, we will

[2]For details on the `CAPSL` `prelude`, refer to Chapter 2, page 54.

re-use some of notions introduced in Chapter 3.

Definition 7.2.1 (Denotational Algebra of an Equational Theory) *Let (Σ, E) be a convergent equational theory. The Σ-algebra $\mathbb{A}$ that, together with a finite set Δ of assignments of variables X in $\mathbb{A}$, forms the denotation of the theory (Σ, E) is called the* denotational algebra *of the equational theory (Σ, E).*

For each $t \in T_{\Sigma,X}$, assume that the corresponding finite sequence $t \rightarrow_{E^*} t{\downarrow}_E$ mechanising $\mathbb{T}_{\Sigma,X}$ is provided (e.g., by a rewriting engine [52] or a tool based on multiset rewriting [181]).

In the denotational algebra $\mathbb{A}$ for the theory (Σ_1, E_1), we take the support-set A_{nat} to be the set $\mathbb{N}$ of natural numbers, the support-set A_{role} to be given by strings and the support-set A_{bool} to equal the set $\{.T., .F.\}$.

Let Pr be a protocol specified by a convergent equational theory (Σ, E). Let $T_{\Sigma,X}|_{Pr}$ be the *set of terms restricted to the* **`PROTOCOL`** *section of Pr's* specification/description (i.e., $T_{\Sigma,X}|_{Pr}$ does not include terms or variables which were used to describe the equational theory (Σ, E) in the first place). For instance, for the protocol Pr_1 in Example 7.1.2, the set $T_{\Sigma_1,X}|_{Pr_1}$ does not include variables x or y which described (Σ_1, E_1) and $T_{\Sigma_1,X}|_{Pr_1} = \{A, B, n, m, \leq(n, m)\}$. When Pr is implicit, we simply write $T_{\Sigma,X}$ to denote the set $T_{\Sigma,X}|_{Pr}$.

As in Chapter 3, the set of variables and the set of terms are partitioned by roles. Variables are either *bound to a role R*, $\mathcal{B}_R$, or *free in a role R*, $\mathcal{F}_R$. In the case of protocol Pr_1, the atom n is bound to the role of A ($n \in \mathcal{B}_A$), while the variable m is free to role A ($m \in \mathcal{F}_A$). For a role R, this naturally extends to terms: a term t is *free in a role R* if at least one of its variables is free in that role and a term t is *bound to a role R* if all its variables are bound to that role. Namely,

$$\begin{cases} t \in \mathcal{F}_R, \text{ if } t \in T_{\Sigma,X'} \text{ and } X' \cap \mathcal{F}_R \neq \emptyset \\ t \in \mathcal{B}_R, \text{ otherwise.} \end{cases}$$

We consider an S-sorted set $(\perp_s \,|\, s \in S)$ of constant function symbols. When the sort s is implicit, we simply write $\perp$ instead of $\perp_s$.

This is extended to non-atomic terms in the following way. Let $n \geq 1$ and $\{s_1, \ldots, s_n\}$ be a set of sorts. Let i be arbitrarily fixed where $1 \leq i \leq n$ and let any j with $1 \leq j \leq n$ be such that $j \neq i$. Any tuple $(a_1, \ldots, \perp_{s_i}, \ldots, a_n)$ with $a_j \in A_{s_j}$ is denoted $\perp_\omega$, where $[\omega, s] \in S^* \times S$ is a type for some function symbol $\sigma \in \Sigma_{\omega,s}$ and $\omega = (s_1, \ldots, s_n)$. In other words, for each $\omega \in S^*$, $\omega = (s_1, \ldots, s_n)$, all constant function symbols over ω where at least one component is $\perp$ are denoted $\perp_\omega$.

With the above, we can now extend the denotational algebra of an equational theory so that it operates also over $\perp$. The following definition formalises this fact.

Definition 7.2.2 (Extension of a Denotational Algebra) *Let* (Σ, E) *be a convergent equational theory. The algebra* $\mathbb{A}_\perp$ *is the* extension of the denotational algebra $\mathbb{A}$ *where:* $A_{s_\perp}$ *is* $A_s \cup \{\perp_s\}$, *for all sorts* $s \in S$ *and any operation in* $\mathbb{A}$ *operates on* $\perp$, *returning* $\perp$.
Formally, for every function symbol $f \in \Sigma_{\omega,s}$ *of type* $[\omega, s] \in S^* \times S$ *and every corresponding function* $f^{\mathbb{A}}$ *operating, in* $\mathbb{A}$, *from* A_ω *to* A_s, *the extended function* $f^{\mathbb{A}_\perp}$ *operates in* $\mathbb{A}_\perp$ *from* $A_\omega \cup \perp_\omega$ *to* $A_{s_\perp}$ *as:*

$$\begin{cases} f^{\mathbb{A}_\perp}(v) = f^{\mathbb{A}}(v), \ \textit{if } v \in A_\omega, \ \textit{where } \omega \in S^*; \\ f^{\mathbb{A}_\perp}(v) = \perp_s, \ \textit{if } v = (v_1, \ldots, v_n) \textit{ and } v_i = \perp_{s_i}, \ \textit{for some } i \in \{1, \ldots, n\} \ \omega = (s_1, \ldots, s_n) \in S^*, s \in S. \end{cases}$$

Definition 7.2.2 expresses that an extension of a denotational algebra $\mathbb{A}$ acts exactly like the original algebra $\mathbb{A}$ when operating over concrete values and it returns $\perp$ whenever it operates over $\perp$.

Using the above, we now formalise *initial role instantiations.*

Definition 7.2.3 (Initial Instantiation of Roles) *Let* $\delta \in \Delta$ *be an assignment. The* initial R-role instantiation $\delta|_R$ *is the projection of the assignment* δ *on a role* R, *extended to* $\mathbb{A}_\perp$*:*

$$\delta|_R = (\{t \mapsto \delta(t),\, t' \mapsto \perp_s \,|\, t \in (\mathcal{B}_A)_s,\, t' \in (\mathcal{F}_A)_s\} \ \mid \ s \in S).$$

Definition 7.2.3 expresses that an initial role instantiation maps terms of type $s \in S$ which are bound to the *R-role* onto A_s and maps terms free in the *R-role* to $\perp$. Unlike in Chapter 3, assignments are implicitly extended to non-atomic terms, as Section 7.1.1 stated and as Definition 7.2.3 formalised above.

On the protocol scenario in Example 7.1.3, this leads to the following initial role instantiations:

$\delta|_A[n/_3, m/_\perp, A/_{alice}, B/_{bob}, \leq(n,m)/_\perp]$,

$\delta|_B[n/_\perp, m/_2, A/_{alice}, B/_{bob}, \leq(n,m)/_\perp]$.

As in Chapter 3, initial role instantiations denote a *multiagent system initialisation scenario.* For each role R, we map each initial instantiation $\delta|_R$ of the *R-role* into an agent ag^δ_R. This gives the *set* $Ag = \bigcup_{\delta \in \Delta} \bigcup_{R \in X_{role}} \{ag^\delta_R\}$ *of agents.*

Remark 7.2.4 *Note that the extended denotational algebra and the initial instantiations operate over non-atomic terms. This is contrary to the denotations used in the Υ_{IS} formalism in Chapter 3 where instantiations were applied to atomic terms only.*

Therefore, in the formalism to be described, the local states of the agents will contain non-atomic terms as well. In a nutshell, this is the first major step towards modelling receiver-opaque protocols, i.e., agents might not decompose a term down to its atoms.

In the following, let the agent ag_R^δ correspond to an arbitrary *R-role* under the assignment δ and, in particular, let ag_A^δ correspond to the *A-role* in the Pr_1 protocol under the assignment δ in Example 7.1.3. We are now going to present the formal description of the agent ag_R^δ.

Definition 7.2.5 (Range of a Term for an Agent) *Let Pr be a protocol specified by a con- vergent equational theory. The* range $Range_R(t)$ of a term $t \in T_{\Sigma,X}|_{Pr}$ for an ag_R^δ agent *is as follows:*

$$Range_R(t) = \begin{cases} A_s & \textit{if } t \in (\mathcal{B}_R)_s \cap X_s \qquad (1) \\ A_{s_\perp} & \textit{if } t \in (\mathcal{F}_R)_s \cap X_s \qquad (2) \\ (A_{s_1} \times \ldots \times A_{s_n}) \cup A_s & \textit{if } t \in (\mathcal{B}_R)_s, t = \sigma(t_1, \ldots, t_n), \\ & \sigma \in \Sigma_{(s_1,\ldots,s_n),s}, t_i \in T_{\Sigma,X,s_i}, i \in \{1, \ldots, n\} \qquad (3) \\ (A_{s_1} \times \ldots \times A_{s_n}) \cup A_{s_\perp} & \textit{if } t \in (\mathcal{F}_R)_s, t = \sigma(t_1, \ldots, t_n), \\ & \sigma \in \Sigma_{(s_1,\ldots,s_n),s}, t_i \in T_{\Sigma,X,s_i}, i \in \{1, \ldots, n\} \qquad (4) \end{cases}$$

Case (1) of Definition 7.2.5 expresses that the range of a variable t of sort s is the support-set A_s of algebra $\mathbb{A}$ if t is a variable bound to the role R. According to case (2), a variable t of sort s ranges over $A_s \cup \{\perp_s\}$ if t is a free variable in the role R. Let $\sigma \in \Sigma_{\omega,s}$ be a function symbol such that $\omega = (s_1, \ldots, s_n)$, where $n \geq 1$, $s_i \in S$, $1 \leq i \leq n$. Let t_i be a term of sort s_i where $1 \leq i \leq n$ and $t = \sigma(t_1, \ldots, t_n)$ be a term of sort s. Case (3) expresses that any term t like above which is bound to the role of R ranges over the union between A_s and the generalised Cartesian product of the ranges dictated by the structure of $\omega \in S^*$ like above. For a non-atomic term t like above which is instead free in the role R, the value $\perp_s$ is added to the range defined by case (3) as above. This is sufficient since any term t in $T_{\Sigma,X}|_{Pr}$ will eventually be rewritten under the convergent equational theory (Σ, E) to some normal term of some sort s.

To clarify Definition 7.2.5, we will show its meaning using the protocol Pr_1. Let us take the example of term $\leq(m, n)$ which is free in the role of A in protocol Pr_1. The term $\leq(m, n)$ is free in the role of A and it is a term of sort *Bool*. Then, the initial *A-role* instantiation will assign $\leq(m, n)$ to $\bot_{\{.T.,.F.\}}$. But, as the execution proceeds, a participant of the *A-role* will set the free variable m to a concrete value, and then the participant will "rewrite" the term $\leq(m, n)$ to consequent concrete values (e.g., in the scenario in Example 7.1.3, $\leq(m, n)$ will become $\leq(3, 2)$, $\leq(2, 1)$, $\leq(1, 0)$ and, finally, $.F.$). Hence, according to case (4) of Definition 7.2.5, the term $\leq(m, n)$ of the *A-role* in protocol Pr_1 ranges over $(A_{nat} \times A_{nat}) \cup A_{bool_\bot} = (\mathbb{N} \times \mathbb{N}) \cup \{.T., .F.\}_\bot$.

Now, Definition 7.2.6 will formalise all the possible ways that terms can be mapped into values with the formalisation of an agent.

Definition 7.2.6 (Store of an Agent) *Let Pr be a protocol specified by a convergent equational theory. A* store *for an agent ag^δ_R is a relation between terms and their respective ranges for ag^δ_R, denoted store=$(t :: Range_R(t) \mid t \in T_{\Sigma,X}|_{Pr})$.*

Hence, a store unwinds all the acceptable valuations of terms over some established domains.

The store of ag^δ_A in a model for Pr_1 is as follows:
$store_{ag^\delta_A} = (A :: String, B :: String, n :: \mathbb{N}, m :: \mathbb{N}_\bot, \leq(n, m) :: (\mathbb{N} \times \mathbb{N}) \cup \{.T., .F.\}_\bot)$.

Remark 7.2.7 *Note that unlike in Chapter 3, the ranges for terms are not restricted to a particular role. This is because the present formalisation aims at modelling receiver-opaque protocols. Hence, it is reasonable to model agents that accept any value for an opaque term.*

In the following, we capture an actual instantiation of a store. For that, we introduce the notion of *view*.

Definition 7.2.8 (Views of an Agent) *An* initial view *for an agent ag^δ_R is a concrete initial R-role instantiation δ.*
A non-initial view *for ag^δ_R is a concrete assignment $\delta[^y/_v]$, for some $y \in (\mathcal{F}_R)_s$, $v \in A_s$ (i.e., $v \neq \bot_s$).*

A possible non-initial view for ag^δ_A in a model for Pr_1 is:

$$view_{ag^\delta_A} = (A \mapsto alice, B \mapsto bob, n \mapsto 3, m \mapsto 2, \leq(n, m) \mapsto \bot).$$

The non-initial view $view_{ag_A^\delta}$ shows that ag_A^δ has received the value 2 for the free term m and updated the value for m in her view from $\perp$ to 2. However, ag_A^δ has not yet "calculated" the value of $\leq(n,m)$ under the new view, i.e., $\leq(n,m)$ is still $\perp$ in $view_{ag_A^\delta}$. In the following, we will formalise the hereby mentioned updates.

The set *Adj* of *adjacent terms* is given by terms which are unrelated to the equational theory E. These terms are however induced by the protocol specification (i.e., they represent clock variables, flag variables, etc.). For each role R, we extend $Range_R$ and initial *R-role* instantiations to operate also over *Adj*.

Definition 7.2.9 (Local States of Agents.) *An* (initial) local state *is an (initial) view together with some adjacent terms and their assigned values.*

Concrete ranges and initial values to be added to the local states of each *R-role* are given by a precise protocol specification. An example of this will be shown in Section 7.5, through the modelling of the FOO'92 e-voting protocol.

Let *step* be a variable added to the Pr_1 specification. In this fashion, in a model for the protocol Pr_1, let $step \in Adj$ be an adjacent, atomic term of sort *nat*, let $Range_R(step) = \mathbb{N}$ and $step \mapsto 1$ in an initial setup (where $R \in \{A, B\}$). An initial state i_l and local state l of agent ag_A^δ in a model for Pr_1 are then as shown in the following.

Example 7.2.10 (A Possible (Initial) State of ag_A^δ in a model for Pr_1)
initial state $i\_l = (step \mapsto 1, A \mapsto alice, B \mapsto bob, n \mapsto 3, m \mapsto \perp_{\mathbb{N}}, \leq(n,m) \mapsto \perp_{\{.T.,.F.\}})$;
non-initial state $l = (step \mapsto 5, A \mapsto alice, B \mapsto bob, n \mapsto 3, m \mapsto 2, \leq(n,m) \mapsto .F.)$.

The set $L_{ag_R^\delta}$ of *possible local states* of ag_R^δ follows from the above.

Let Pr be a protocol specified by a convergent equational theory (Σ, E). Given $l \in L_{ag_R^\delta}$, we use the following notations:

- $l.view$ to denote the inner view of l;
- $l.t$ to denote an inner term $t \in T_{\Sigma,X}|_{Pr}$ of l;
- $l|_{t=x}$ to denote the fact that $l.t{=}x$, where $t \in T_{\Sigma,X}|_{Pr}$, $x{\in}Range_R(t)$;

- $l|_\delta$ to denote when $l|_{t=x}$ and $\delta[t/_x]$, for all $t \in dom(\delta)$, for $x \in Range_R(t)$;
- $l[t/_y]$ to denote the fact that $l.t$ being set to y, for $t \in T_{\Sigma,X}|_{Pr}$ for some $y \in Range_R(t)$.

In the following, we further describe the agents in this formalisation. Let i denote ag_R^δ and i' denote $ag_{R'}^{\delta'}$, the maps of an initiator role R and receiver role R', respectively. Let $l \in L_{ag_R^\delta}$, $t \in T_{\Sigma,X}$ and $x \in Range_R(t)$.

Definition 7.2.11 (Local Actions of Agents)
The set $LAct_i = \{send(t,x,i'),\ receive(t,x),\ rewrite,\ empty\}$ *is the set of* possible local actions *of agent* i*, where* t*,* x*,* i' *are as above.*

Definition 7.2.12 (Local Protocol of Agents) *The* local protocol P_i *of agent* i *is defined as follows:*

$$P_i(l|_{step=nr_1,\, l.t=x,\, l.R'=i'.R'}) = \{send(t,x,i')\},$$
$$P_i(l|_{step=nr_2}) = \{receive(t,x)\},$$
$$P_i(l|_{step=nr_3}) = \{rewrite\},$$

where t*,* x*,* i'*,* l *are as above,* $step \in Adj$*,* $nr_j \in Range_R(step)$ *and* $j \in \{1,2,3\}$*.*

Similarly to the Υ_{IS} formalisation given in Chapter 3, the parameters of the actions are restricted to proper subsets of $T_{\Sigma,X}$ when a particular protocol is given. For instance, the action *receive* is defined only over terms to be received by a *R-role*, i.e., $ReceivedMsg^R$, and nr_j is made concrete by a precise protocol description/specification, i.e., $nr_j \in Steps^R$.

The Environment Agent. We consider an environment agent *Env* that "records" all the communication. Thus, the *Env* agent will "record" any term t of any value x coming from any agent i and aimed at any other agent i'. Therefore, the local states of the *Env* agent are given by sets of tuples of the form $(t :: \bigcup_{R \in role} Range_R(t) :: Ag :: Ag \,|\, t \in T_{\Sigma,X})$. This gives the set L_{Env} of *possible local states of the Environment agent.*

The environment has one possible action denoted *listen* and enabled at any local state. Hence, the set $LAct_{Env} = \{listen\}$ is the set of *possible local action of the Env agent.*

Unlike in the Υ_{IS} formalisation, the *Env* agent does not model a Dolev-Yao intruder. In Section 7.5, we manipulate this agent to render a passive attacker model (see Section 2.2.1.3, page 34 for the notion of *passive attacker*).

Global States. As usual, a *global state* g is a tuple $(l_1, \dots, l_n, l_{Env})$, where $i \in Ag$, $l_i \in LAct_i$ and $l_{Env} \in L_{Env}$. The set G of global states is the set of all possible states g as above.

Joint Actions. As usual, a *joint action* a is a tuple $(a_1, \dots, a_n, a_{Env})$, where $i \in Ag$, $a_i \in Act_i$ and $a_{Env} \in Act_{Env}$. The set Act of joint actions is the set of all possible joint actions a as above.

Agents' Local Evolution Function. Let i denote the ag^{δ}_{R} agent, i' denote the $ag^{\delta'}_{R'}$ agent as above, let $l \in L_i$ be a local state of agent i and $a \in Act$ be a joint action.

The *local evolution function* E_i of agent i is defined as:

$$\begin{cases} l[step/_{nr+1}] & \text{if } l|_{t=x, step=nr, R'=i'.R'},\ \text{for } a_i = send(t,x,i'), a_{Env} = listen, \\ & a'_i = receive(t,x) \\ l[step/_{nr+1}, t/x] & \text{if } l|_{t_j=x_j, step=nr, R'=i'.R'},\ \text{for } a_i = receive(t,x), a_{Env} = listen, \\ & a'_i = send(t,x,i),\ t_j \in Sub(t) \\ l[step/_{nr+1}, t/t'] & \text{if } l|_{step=nr+1},\ \text{for } a_i = rewrite, a_{Env} = listen, \\ & a'_i = empty,\ t \in T_{\Sigma,X}, t' = t{\downarrow}_E \end{cases}$$

We give the denotation of the local evolution function E_i presented above. The preconditions of enabling a state-update upon receiving are similar to those in the Υ_{IS} formalisation presented in Chapter 3. These are summarised as follows:

- the local actions *receive* of agent i are synchronised with the local actions *send* of agent i' and with the local action *listen* of the *Env* agent;
- agent i is in the step nr where he awaits the receiving of the message t;
- the purported sender is the agent of the R'-*role*[3] (i.e., $i.R' = i'.R'$);

[3]If certain anonymising procedures [92] are used in the design of the protocol, then this condition is dropped.

- the values x_j of certain subterms t_j in the received term t are consistent with agent i's view, i.e., $l|_{t_j=x_j}$; this is a simplified expression in E_i of the denotation of symbol *out_match* in Chapter 3.

The meaning of the state-updates is mainly as in Chapter 3. For instance, the denotation of the second line is as follows. If the pre-conditions are satisfied (i.e., $l \in L_i$ is such that indeed $l|_{t_j=x_j, step=nr, R'=i'.R'}$), then the local state l is updated, i.e., $l[step/_{nr+1}, t/x]$. This update implies that the value nr of the step variable is incremented and the newly received term t is set to x. In the state-updates of this formalism, the denotation of *set* as per Chapter 3 (i.e., decomposing $[t/x]$ down to atomic terms) is not encompassed. Its meaning is embedded in the entry of the local evolution for the local action *rewrite*. Therefore, the local evolution function forces a *rewrite* action at a state newly updated upon receiving.

We illustrate on the local evolution function for action *rewrite* in the model for Pr_1. Assume that ag^{δ}_{A} has performed the action $receive(m, 2)$ at a local state $l|_{step=1}$. Recall that the protocol Pr_1 is defined by the convergent equational theory (Σ_1, E_1). The rewriting underlined by the equational theory (Σ_1, E_1) for $n = 3$ (as per the assignment $\delta|_A$ extracted from Example 7.1.3) and $m = 2$ (as per the action $receive(m, 2)$) is:

$$\leq\big(3,2\big) \longrightarrow \leq\big(succ(2), succ(1)\big) \longrightarrow \leq\big(2,1\big) \longrightarrow \leq\big(succ(1), succ(0)\big) \longrightarrow \leq\big(1,0\big) \longrightarrow .F. \tag{7.1}$$

Recall that, by the evolution function E_i, for $l|_{step=1} \in L_{ag^{\delta}_{A}}$ and action $a_{ag^{\delta}_{A}} = receive(t, x)$, it is the case that $E(l, a){=}l[step/_2, \leq(n, m)/_{(3,2)}]$. Then, the the evolution function E_i emulates the rewriting presented in (7.1). The implied local state update is shown in Example 7.2.13.

Example 7.2.13 (A State-Update Driven by the Rewriting in (Σ_1, E_1)**)**
At $l|_{step=2, \leq(n,m)=(3,2)} \in L_{ag^{\delta}_{A}}$ *and* $a_{ag^{\delta}_{A}} = rewrite$*, the equational theory* (Σ_1, E_1) *implies the following sequence of updates:* $l[step/_3, \leq(n, m)/_{(2,1)}]$, $l[step/_4, \leq(n, m)/_{(1,0)}]$ *and* $l[step/_5, \leq(n, m)/_{.F.}]$.

For particular classes of protocols (where the intermediate rewriting updates are not of interest), we optimise the local evolution function by compressing it into one update step, i.e., the sequence of updates triggered at the local state $l|_{step=2, \leq(n,m)=(3,2)}$ and shown in Example 7.2.13 is collapsed into

$l[step/_3, \leq(n,m)/_{.F.}]$. The summary of the local evolution function E_i aforementioned formalises this optimisation.

Definition 7.2.14 (The Global Evolution Function)
The global evolution function $t : G \times Act \to G$ *is such that* $t(g,a) = g'$ *if* $act_i \in P_i(g_i)$, $E_i(g_i, a) = g'_i$, *for all* $i \in Ag \cup \{Env\}$, *for* $g, g' \in G$ *and* $a \in Act$.

A *path* is an infinite sequence of global states described by the global evolution function. A finite sub-sequence of a path is also called a *path.* Paths naturally give the set of *reachable* states.

Definition 7.2.15 (Equational Interpreted System for Pr**)** *Let* Pr *be a protocol specified by a convergent equational theory* (Σ, E). *An* equational interpreted system for Pr *(*Υ^E_{IS}*) is a tuple* $\mathcal{I} = (G, Act, P, t, I_0, V)$, *where* $I_0 \subset G$ *is a set of initial global states,* $P = (P_i \,|\, i \in Ag \cup \{Env\})$, *and* $V : G \times PV \to \{true, false\}$ *is a valuation function for the propositions* PV *of a logic language of the system.*

For Definitions 7.2.16–7.2.20, let Pr be a protocol specified by a convergent equational theory (Σ, E), $\mathcal{I}$ be an equational interpreted system model for Pr and i be an arbitrary agent in $\mathcal{I}$.

Definition 7.2.16 (Local Satisfaction of Equational Equalities of Terms) *The local state* $l \in L_i$ satisfies $t =_E t'$, *written* $l \models (t =_E t')$, *if* $l|_{t=t'}$, *for* $t \to_{E*} t'$ *with* $t' = t{\downarrow_E}$, $t \in T_{\Sigma,X}|_{Pr}$ *(i.e.,* $t \notin Adj$*).*

Let $t =_E t'$ be an arbitrary equality of terms modulo E. Definition 7.2.16 expresses that a local state l of agent i satisfies this equality of terms modulo E if, under the interpreted system Υ^E_{IS}, the term t has been rewritten to the normal term t' and this is recorded in local state l of agent i.

Definition 7.2.17 (Local Equational Indistinguishability) *Two local states* $l \in L_i$ *and* $l' \in L_i$ *are* i-indistinguishable modulo E, *written* $l \approx^E_i l'$, *when* $l \models (t =_E t')$ *if and only if* $l' \models (t =_E t')$, *for all* $t \in T_{\Sigma,X}|_{Pr}$, *i.e.,* $t \notin Adj$.

Definition 7.2.17 expresses that two local states of agent i are indistinguishable modulo E if they satisfy the same equalities of terms modulo E.

Note that local satisfaction of equalities of terms modulo E is defined only with respect to normal terms. For instance, in our running example on the $\Upsilon_{IS}^{E_1}$ for the protocol Pr_1, writing $l \models (\leq(m,n) =_{E_1} \leq(2,1))$ is not well-founded since $\leq(2,1) \neq \leq(m,n)\downarrow_{E_1^*}$. This is in line with [4], i.e., different ciphertexts assigned to the same symbolic term t are not distinguishable if the value for the decryption key is not held. Let i be ag_A^{δ} in the $\Upsilon_{IS}^{E_1}$ for the protocol Pr_1 and its local states $l_1|_{step=2,\leq(n,m)=(3,2)}$, $l_2|_{step=3,\leq(n,m)=(2,1)}$ and $l_3|_{step=4,\leq(n,m)=(1,0)}$ be as in Example 7.2.13. The only equality of terms to be satisfied in $\Upsilon_{IS}^{E_1}$ at the local states L_i is $\leq(m,n) =_{E_1} .F.$ No state in the set $\{l_1, l_2, l_3\}$ satisfies it. Therefore, $l_1|_{step=2,\leq(n,m)=(3,2)} \approx_i^{E_1} l_2|_{step=3,\leq(n,m)=(2,1)} \approx_i^{E_1} l_3|_{step=4,\leq(n,m)=(1,0)}$.

Nevertheless, because the rewriting converges in the model for Pr_1, there exists a local state $l \in L_i$ such that $l \models (\leq (m,n) =_{E_1} .F.)$. Namely, as per Example 7.2.13, this state is $l|_{step/5,\leq(n,m)/.F.}$.

We lift local indistinguishability to global indistinguishability as usual.

Definition 7.2.18 (Global Equational Indistinguishability) *Two reachable global states $g, g' \in G$ are i-*indistinguishable modulo E*, written $g \sim_i^E g'$, if $g_i \approx_i^E g'_i$. The relation $\sim_i^E \subseteq G \times G$ is the* quotient-indistinguishability relation.

To refer to an indistinguishability relation aligned with standard approaches in interpreted systems models, we give Definitions 7.2.19 and 7.2.20.

Definition 7.2.19 (Local-State Equality) *Two local states $l \in L_i$ and $l' \in L_i$ are i-*equal*, written $l =_i l'$ if $l = l'$.*

Definition 7.2.20 (Global-State Equality) *Two reachable global states $g, g' \in G$ are i-*equal*, written $g =_i g'$, if $g_i =_i g'_i$. The relation $=_i \subseteq G \times G$ is the* equality relation.

Definition 7.2.21 (Equational Multiagent System Model for Pr) *Let Pr be a protocol specified by a convergent equational theory (Σ, E), $\mathcal{I}$ be an equational interpreted system for Pr. The unwound multiagent system model implied by $\mathcal{I}$ is called the* equational multiagent system model for Pr *(M_{IS}^E) and it is given by the tuple $(G, \sim_i^E, =_i, V)$, where G is the set of reachable states* *implied by $\mathcal{I}$, V is the set of atomic proposition in $\mathcal{I}$ and, for all $i \in Ag \cup \{Env\}$, $\sim_i^E$, $=_i$ are the aforementioned indistinguishability relations.*

In the following, we sometimes overload the notation $\mathcal{I}$ to denote both the equational interpreted system for Pr and the equational multiagent system model for Pr (i.e., the IS-based specification and the unwound model). However, the context will always disambiguate the notation $\mathcal{I}$.

7.2.1 Logical Language

In this section we give a logical language interpreted on M_{IS}^E models.

Let Pr be a protocol specified by a convergent equational theory (Σ, E), $\mathcal{I}$ be its equational multiagent system model M_{IS}^E, $p \in PV$ and $i \in Ag \cup \{Env\}$. The language $\mathcal{L}$ is defined by the following grammar:

$$\varphi ::= p \mid \neg\varphi \mid \varphi \wedge \varphi \mid K_i\varphi \mid QK_i\varphi \mid AX\varphi \mid AG\varphi \mid A(\varphi U \varphi).$$

We denote the operator K by the *knowledge modality* and the operator QK by the *quotient-knowledge modality.* Furthermore, $K_i\varphi$ reads "agent i knows the fact φ" and $QK_i\varphi$ reads "agent i knows the fact φ modulo E".

The semantics for CTL on equational multiagent system models is as on unwound interpreted system (see [80] or Chapter 2, page 25). In the following we give the interpretation of knowledge modalities on equational multiagent system model. Let Pr be a protocol specified by an equational theory (Σ, E), $\mathcal{I}$ be its equational multiagent system model, $\varphi \in \mathcal{L}, i \in Ag \cup \{Env\}$ and $g \in G$. Then,

$$\begin{aligned}
(\mathcal{I}, g) \models K_i\varphi \quad & \text{if} \quad (\text{for all } g' \in G)(g =_i g' \text{ implies } (\mathcal{I}, g') \models \varphi) \\
(\mathcal{I}, g) \models QK_i\varphi \quad & \text{if} \quad (\text{for all } g' \in G)(g \sim_i^E g' \text{ implies } (\mathcal{I}, g') \models \varphi).
\end{aligned}$$

The formula $\varphi \in \mathcal{L}$ is *valid on* $\mathcal{I}$ if $\varphi \in \mathcal{L}$ is satisfied at all states of the (unwound) equational multiagent system model $\mathcal{I}$.

Evaluation of the M_{IS}^{E} Model

In this section we have introduced $\mathit{\Upsilon}_{IS}^{E}$ as an interpreted system-based formalisation of the multi-session execution of a protocol specified under convergent equational theory (Σ, E). The multiagent system model unwinding $\mathit{\Upsilon}_{IS}^{E}$ was denoted M_{IS}^{E}.

Recall that the formalisations in Chapter 3 encoded receiver-transparent protocols, where the equality relation was an adequate state-indistinguishability relation. This is not the case for receiver-opaque protocols (e.g., different opaque terms might have the same meaning). Therefore, the formalisations in the previous section generalise those in Chapter 3. Firstly, the quotient indistinguishability relation is introduced to accommodate the distinction of states based on the underlying equational theory of the protocol. Other extensions to the formalism in Chapter 3 are present in the modelling of the local states, the local evolution function, etc. In this fashion, the $\mathit{\Upsilon}_{IS}^{E}$ and M_{IS}^{E} formalisms have been presented as a bespoke methodology for the modelling of a class of receiver-opaque protocols.

Nevertheless, many of the receiver-transparent protocols used in practice are described using convergent equational theories. In that sense, receiver-transparent protocols can also be modelled using $\mathit{\Upsilon}_{IS}^{E}$. Pursuing the latter specification to model an RTP would be less efficient than using the RT-tailored approach in Chapter 3 (i.e., the larger size of the states, the larger size of the ranges for terms, the increased number of transitions implied by the evolution function in $\mathit{\Upsilon}_{IS}^{E}$ than in $\mathit{\Upsilon}_{IS}$).

The following sections and Appendix C will show extensions of the RO-oriented formalisations in this section. The automation to generate the $\mathit{\Upsilon}_{IS}^{E}$ specifications and techniques to verify them will also be discussed.

7.3 Interrogative Multiagent Systems Model for Equationally Specified Protocols

Firstly, this section introduces *interrogative interpreted systems* ($\mathit{\Upsilon}_{IS}^{\mathbb{I}}$); this is a class of interpreted systems that augments the specification of agents with local predicates called *interrogations* (Section 7.3.1). A new knowledge modality is defined and interpreted on the unwinding of the $\mathit{\Upsilon}_{IS}^{\mathbb{I}}$

system (Section 7.3.2). Denoted $\mathbb{I}K$, this knowledge modality is shown to coincide with the quotient knowledge modality, if particular interrogations are imposed (Section 7.3.3). The usefulness of the latter lies in the fact that interrogative interpreted systems are better suited for automatic verification than equational interpreted systems. In fact, a model checking algorithm of interrogative interpreted systems against the $\mathbb{I}K$ modality is proposed.

7.3.1 Interrogation Sets

In Definition 7.3.4 of this subsection we formalise agents' interrogations. To achieve this we use the notions of logical signatures, logically extended signature, logical terms and predicates. We proceed to explain these notions.

An *S-sorted logical signature* contains *logic symbols* of type $[\omega]$, for $\omega \in S^*$. A logical signature is somewhat similar to a signature. Informally, a (standard) signature specifies symbols related to algebraic operators, e.g., *decrypt*, whereas a logical signature specifies symbols related to logic facts, e.g., *isDecrypted*. The terms constructed with such logic symbols, e.g., $\mathit{isDecrypted}(t_1, t_2)$, will be interpreted as logic predicates, e.g. $\mathit{isDecrypted}(t_1/_{\{3\}_{100001}}, t_2/_{100001})$, over $\{true, false\}$, as the next will show.

Definition 7.3.1 (Logically Extended Signatures) *A* logically extended signature *is given by a tuple (Σ, Σ_L), where Σ is an S-sorted signature and Σ_L is an S-sorted logical signature.*

Let $\mathcal{I}$ be an equational interpreted system for a (Σ, E)-specified protocol Pr, Δ be the set of assignments in the algebra $\mathbb{A}_\perp$, δ be an initial role instantiation and i be the agent ag^δ_R. The tuple (Σ, Σ_L, E) is the *logically extended equational theory* corresponding to the equational theory (Σ, E). A logically extended equational theory (Σ, Σ_L, E) describes a communication protocol Pr in more detail, e.g., Σ_L can capture certain protocol properties, like pre-goals[4].

The set $T_{\Sigma,\Sigma_L,X}$ of *logical terms* is defined on logically extended signatures (Σ, Σ_L) in the same way the set $T_{\Sigma,X}$ of terms is defined on the signature Σ.

[4] We recall that notion of *pre-goals* is introduced in Chapter 6, page 230.

The denotation of logically extended signatures is given through a *logical extension of the algebra* $\mathbb{A}_\perp$. In the logical extension of the algebra $\mathbb{A}_\perp$, the *interpretation* $i\_p^{\mathbb{A}_\perp}(\delta)$ *of a logical term* $p \in T_{\Sigma,\Sigma_L,X}$ *under assignments* $\delta \in \Delta$ is a predicate $p^{\mathbb{A}_\perp}$ evaluated over $\{true, false\}$. When $\mathbb{A}_\perp$ is implicit, we simply write $i\_p(\delta)$ instead of $i\_p^{\mathbb{A}_\perp}(\delta)$.

We exemplify with a logical signature for the Pr_1 protocol.

Example 7.3.2 (Σ_{1_L} – a Logical Signature for Protocol Pr_1)
Let Σ_{1_L} be a logical signature containing two logical symbols:

- *"smaller" of type $[nat, nat]$;*
- *"diffOne" of type $[nat, nat]$.*

The meanings of the predicates corresponding to the symbols in Example 7.3.2 follow the intuition: $i\_\mathrm{diffOne}(n, m)(\delta)$ is *true* if "the absolute difference between $\delta(n)$ and $\delta(m)$ is 1", etc.

Definition 7.3.3 (Logical Terms of Agents) *Let j be an arbitrary agent in an IS formalisation. A fixed set $In_j \subseteq T_{\Sigma,\Sigma_L,X}$ denotes* the set of logical terms of agent j.

Definition 7.3.3 equips IS agents with sets of logical terms. This can be seen as an extension of agents' stores.

Definition 7.3.4 (Local Interrogations for Agents) *Let j be an arbitrary agent in an IS formalisation. The set $Intr_j = \{i\_p(\delta) \mid p \in In_j\}$ of predicates contains the* local interrogations for agent j. *The $Intr_j$ sets are denoted as* local interrogation-sets, *for $j \in Ag$.*

The Ag-indexed set $Intr = (Intr_j \mid j \in Ag)$ is the interrogation-set.

Definition 7.3.4 augments the IS specification of agents with sets of predicates, i.e., interrogations. In parallel with Υ_{IS}, these sets can be seen as an extension of agents' views.

In Example 7.3.2 we illustrate the notions of logical terms and local interrogations of agents.

Example 7.3.5 (Possible Interrogations For Agent ag_A^δ in Pr_1)
Two possible sets of logical terms of agent $i = ag_A^\delta$ in the model Pr_1 are:

- $In1_i = \{smaller(n, m)\}$;
- $In2_i = \{diffOne(n, m)\}$.

The sets $In1_i$ and $In2_i$ respectively trigger two possible local interrogation-sets of agent $i = ag_A^\delta$ in the model Pr_1:

- $Intr1_i$=*i_smaller*$(n, m)(\delta)$;
- $Intr2_i$=*i_diffOne*$(n, m)(\delta)$.

We will now show how the interrogation-sets can be used to describe an indistinguishability relation. Such a relation is similar to the one used for explicit knowledge and implemented in `MCMAS-X` [134, 170, 130].

Definition 7.3.6 (Local Interrogative Indistinguishability) *Let j be an arbitrary agent in an IS formalisation. Two local states $l, l' \in L_j$ are* indistinguishable modulo $Intr_j$*, written $l \approx^{Intr_j} l'$, if i_p(δ) = i_p(δ') for all $p \in In_j$, where $l|_\delta$, $l'|_{\delta'}$ and $\delta, \delta' \in \Delta$.*

Definition 7.3.6 expresses that two local states l and l' of agent j are indistinguishable through interrogations if local state l evaluates all the interrogations of agent j in the same way as local state l', i.e., if predicate p is evaluated to $true$ at l if and only if it is evaluated to $true$ at l', and predicate p is evaluated to $false$ at l if and only if it is evaluated to $false$ at l', for all $p \in In_j$.

We further clarify the meaning implied by Definition 7.3.6 by an example.

Example 7.3.7 (Interrogative Indistinguishability in a Model for the protocol Pr_1) *Let i be the agent ag_A^δ in a model for the protocol Pr_1, $l_{ag_A}|_{\delta[n/_9, m/_8]}$ and $l'_{ag_A}|_{\delta[n/_9, m/_5]}$. Hence, the local states l and l' are the same except for the value of m. Let $Intr1_i$ and $Intr2_i$ be the interrogation-sets in Example 7.3.5. Note that*

- *i_smaller*$(n, m)(\delta[n/_9, m/_8])$=$false$ *and i_smaller*$(n, m)(\delta[n/_9, m/_5])$=$false$;
- *i_diffOne*$(n, m)(\delta[n/_9, m/_8])$=$true$ *and i_diffOne*$(n, m)(\delta[n/_9, m/_5])$=$false$.

Therefore, $l \approx^{Intr1_i} l'$ holds, but $l \approx^{Intr2_i} l'$ does not hold, i.e., $l \not\approx^{Intr2_i} l'$.

We lift local interrogative indistinguishability to global states in a standard way.

Definition 7.3.8 (Global Interrogative Indistinguishability) *Let j be an arbitrary agent in an IS formalisation. Two global states $g, g' \in G$ are* indistinguishable modulo $Intr_j$, *written $g \sim^{Intr_j} g'$, if $g_j \approx^{Intr_j} g'_j$. The relation $\sim^{Intr_j} \subseteq G \times G$ is the interrogative indistinguishability.*

Recall that Υ^E_{IS} is the IS formalisation of the execution for protocol Pr specified by the equational theory (Σ, E). The following definition augments this formalisation to support local interrogations.

Definition 7.3.9 (Interrogative Equational Interpreted System) *Let Pr be a protocol specified by (Σ, Σ_L, E) and $\mathcal{I}$ be the equational interpreted system Υ^E_{IS} for Pr. An* interrogative equational interpreted system *is the tuple $\mathcal{I}'{=}(\mathcal{I}, Intr)$, where $Intr$ is the indexed set $(Intr_j \,|\, j \in Ag \cup \{Env\})$ of interrogations for each agent j.*

Definition 7.3.9 expresses that an interrogative equational interpreted system $\mathcal{I}'{=}(\mathcal{I}, Intr)$ is an extension of the equational interpreted system $\mathcal{I}$ with an *interrogation-set $Intr$*.

The following definition describes the unwinding of an interrogative equational interpreted system.

Definition 7.3.10 (Interrogative Equational Multiagent System Model) *Let Pr be a protocol specified by (Σ, Σ_L, E) and $\mathcal{I}$ be the equational interpreted system Υ^E_{IS} for Pr. An* interrogative equational multiagent system model *is the tuple $(G, \sim^E_j, =_j, \sim^{Intr_j}, V)$, where G is the set of reachable states implied by $\mathcal{I}$, V is the set of atomic propositions in $\mathcal{I}$ and $\sim^E_j$, $=_j$, $\sim^{Intr_j}$ are the indistinguishability relations previously described, for all agents j.*

7.3.2 Logical Language

We introduce the language $\mathcal{L}'$ as an extension of $\mathcal{L}$ in Section 7.2.1. To $\mathcal{L}$, we add the modality $\mathbb{I}K_i$, for $i \in Ag \cup Env$. The reading of $\mathbb{I}K_i\varphi$ is "agent i interrogatively knows the fact φ". The denomination we use for the operator $\mathbb{I}K$ is the *interrogative knowledge modality.*

Let $\mathcal{I}' = (\mathcal{I}, Intr)$ be an interrogative equational multiagent system model. The language $\mathcal{L}$ is interpreted as on the underlying equational multiagent system $\mathcal{I}$, i.e, the model without interrogations. Let $g \in G$ be an arbitrary reachable state in the model $\mathcal{I}'$. The interpretation of interrogative knowledge modality on the model $\mathcal{I}'$ is as follows:

$$(\mathcal{I}', g)\models\mathbb{I}K_i\varphi \text{ if (for all } g' \in G)(g \sim^{Intr_i} g' \text{ implies } (\mathcal{I}', g')\models\varphi)$$

7.3.3 Interrogation Sets of Convergent Equational Theories

In this section we will explicitly link the interrogations of agents and the $\mathbb{I}K$ modality to the convergence of the underlying equational theory. In particular, we consider a special kind of interrogations related to the normal terms of the equational rewriting. These will describe a specific class of interrogative equational MAS models where the interrogative knowledge modality $\mathbb{I}K$ coincides with the quotient-knowledge modality QK. The latter is useful in terms of model checking procedures.

Let Pr be a protocol specified by a *convergent* equational theory (Σ, E), $\mathcal{I}$ be the equational interpreted system for Pr and j denote the ag_R^δ agent as above. Let Σ_L be a logical signature containing the (special) logical symbols $pred \in \Sigma_L$ of type ω, for all $\omega \in S^*$. Let t be an arbitrary term of type ω, i.e., $t\in T_{\Sigma,X}$.

Definition 7.3.11 (Predicates for Terms) *A logical term $pred(t) \in T_{\Sigma,\Sigma_L,X}$ is a* logical term for $t \in T_{\Sigma,X}$*. The* interpretation $i\_pred^E(t)(\delta)$ of a predicate for t in E *is always true, i.e., $i\_pred^E(t)(\delta) = true$, for all $\delta\in\Delta$.*

Definition 7.3.11 suggests that a predicate for a term $t \in T_{\Sigma,X}$ expresses a general truth about this term t, i.e., $i\_pred^E(t)$ is $true$ under all assignments δ for t, $\delta(t) \neq \bot$. In some models it is reasonable to assume that certain properties are constantly $true$ or constantly $false$ [101]. We will see that in our model such properties refer to normal terms. Indeed, in Definition 7.3.12, we consider predicates only for the normal terms on the underlying signature. These purposely selected predicates will constitute the interrogation-sets of agents.

Let j denote the agent ag_R^δ, for $\delta \in \Delta$.

Definition 7.3.12 (Local Interrogations of Convergent Theories)

$In_j^E = \bigcup_{t\in T_{\Sigma,X}} \{pred(t') \mid t' = t{\downarrow_E}\}$ *is the set of logical terms* for the convergent theory E *of agent* j.

$Intr_j^E = \{i\_pred^E(t)(\delta) \mid pr(t)\in In_j^E\}$ *is the set of local interrogations of the convergent theory E for agent j.*

Definition 7.3.12 expresses that local interrogations for convergent theories are particular interrogations; in more detail, they simulate the agents recording the normal terms of the equational rewriting.

Definition 7.3.13 (Interrogative Equational Interpreted System for Convergent Theories) *Let Pr be a protocol specified by (Σ, Σ_L, E) with E convergent, $\mathcal{I}$ be the equational interpreted system Υ^{E}_{IS} for Pr and let j denote the agent ag^{δ}_{R}, for $\delta \in \Delta$. Let $Intr^{E}_{j}$ be the local interrogations for the convergent theory E of agent j and $Intr^{E}$ be the indexed set $(Intr^{E}_{j} \,|\, j \in Ag)$. An interrogative equational interpreted system $\Upsilon^{\mathbb{I}E}_{IS}$* for the convergent theory E *is given by the tuple $\mathcal{I}'{=}(\mathcal{I}, Intr^{E})$.*

Definition 7.3.13 expresses that the system $\Upsilon^{\mathbb{I}E}_{IS}$ extends an equational interpreted system $\mathcal{I}$ with an interrogation-set $Intr^{E}$ for the theory E. In other words, $\Upsilon^{\mathbb{I}E}_{IS}$ is a special kind of interrogative equational interpreted system where the interrogations are as in Definition 7.3.12, i.e., where agents “track” normal terms.

Definition 7.3.14 (Interrogative MAS Model for Convergent Theories) *Let Pr be a protocol specified by (Σ, Σ_L, E) with E convergent and $\mathcal{I}$ be the interrogative equational interpreted system for the convergent theory E. The* interrogative multiagent system model $M^{\mathbb{I}E}_{IS}$ for the convergent theory E *is given by the tuple $(G, \sim^{E}_{j}, =_{j}, \sim^{Intr_j}, V)$, where G is the set of reachable states implied by $\mathcal{I}$, V is the set of atomic propositions in $\mathcal{I}$ and, for all agents j denoting ag^{δ}_{R} in $\mathcal{I}$, $\sim^{E}_{j}$, $=_{j}$, $\sim^{Intr_j}$ are the indistinguishability relations aforementioned.*

Definition 7.3.14 introduces the unwound interrogative IS models which are bespoke for a *convergent* theory E. In that sense, their underlying interrogative indistinguishability relation concerns only the normal terms of the equational theory E. Moreover, as per Definition 7.3.12, the interrogations involved always evaluate to $true$.

Nevertheless, the MAS models in Definition 7.3.14 are a special case of the interrogative MAS models in Definition 7.3.10. Therefore, any $M^{\mathbb{I}E}_{IS}$ model can interpret all knowledge modalities presented in this section (i.e., QK, $\mathbb{I}K$, K). The way we designed the interrogation-sets in the $M^{\mathbb{I}E}_{IS}$ models entails that the $\mathbb{I}K$ modality coincides with QK in the $M^{\mathbb{I}E}_{IS}$ models, as the following shows.

Theorem 7.3.15 *Let Pr be a protocol specified by a convergent equational theory (Σ, E) and $\mathcal{I}$ be an $M_{IS}^{\mathbb{I}E}$ model for E. Then, $\mathcal{I} \models QK_j\varphi$ if and only if $\mathcal{I} \models \mathbb{I}K_j\varphi$, for any $j \in Ag$, any $\varphi \in \mathcal{L}'$.*

Proof

Let $j \in Ag$ be an arbitrary agent with α its arbitrary initial R-role instantiation in $\mathcal{I}$, i.e., $j = ag_R^\alpha$. Let $\varphi \in \mathcal{L}'$ be an arbitrary formula.

$$
\begin{aligned}
\mathcal{I} \models QK_j\varphi \quad & \overset{df \models \mathcal{L}}{\Leftrightarrow} \quad (\text{for all } g' \in G)(g \sim_j^E g' \text{ implies } (I, g') \models \varphi) \\
& \overset{df.\sim^E}{\Leftrightarrow} \quad (\text{for all } g' \in G)(g_j \approx^E g'_j \text{ implies } (\mathcal{I}, g') \models \varphi) \\
& \overset{df.\approx^E}{\Leftrightarrow} \quad (\text{for all } g' \in G)(\text{for all } t \in T_{\Sigma,X})((g_j \models (t =_E t') \text{ iff } g'_j \models (t =_E t')) \text{ implies } (\mathcal{I}, g') \models \varphi) \\
& \overset{df \models \text{eq}}{\Leftrightarrow} \quad (\text{for all } g' \in G)(\text{for all } t \in T_{\Sigma,X},\, t' = t{\downarrow_E})((g_j|_{t=t'} \text{ iff } g'_j|_{t=t'}) \text{ implies } (\mathcal{I}, g') \models \varphi) \ (1)
\end{aligned}
$$

Without loss of generality, let

$$g_j|_\delta, \ \ g'_j|_{\delta'} \quad (2)$$

denote the local states in (1), where δ, δ' are arbitrary assignments extending α, i.e., the initial R-role instantiation of agent j.

Note that in $\mathcal{I}$ the following holds:

$$In_j^E = \bigcup_{t \in T_{\Sigma,X}} \{pred(t{\downarrow_E})\}, Intr_j^E = \bigcup_{t \in T_{\Sigma,X}} \{i\_pred(t{\downarrow_E})(\delta) = true\} \ \ (3)$$

From (2), (3) and the definition of $\approx^{Intr_j}$, it follows that in (1) the following holds:

$$g_j \approx^{Intr_j^E} g'_j \quad (4).$$

From (1), (4), it follows that:

$$
\begin{aligned}
(1) \quad & \Leftrightarrow \quad (\text{for all } g' \in G)(g_j \approx^{Intr_j^E} g'_j \text{ implies } (\mathcal{I}, g') \models \varphi) \\
& \overset{df.\sim^{Intr_j}}{\Leftrightarrow} \quad (\text{for all } g' \in G)(g \sim^{Intr_j} g' \text{ implies } (\mathcal{I}, g') \models \varphi) \\
& \overset{df \models \mathcal{L}'}{\Leftrightarrow} \quad (\mathcal{I}, g) \models \mathbb{I}K_j\varphi \quad \blacksquare
\end{aligned}
$$

Evaluation of the Models in Section 7.3

The interrogative equational interpreted system extends the IS formalisms with interrogations (i.e., predicates) for agents. Similar approaches are present in [134] where the IS is enriched with awareness predicates. For instance, in [134], agent i would be aware of ϕ if the formula ϕ is within his facts (i.e., similar to ϕ being an interrogation in $\Upsilon_{IS}^{\mathbb{I}E}$). In [134], agent i explicitly knows ϕ if i knows it in the standard sense and i is aware of ϕ. Therefore, it is easy to see that the knowledge modalities presented in [134] (i.e., the explicit knowledge and the awareness modality) are interpreted differently than the $\mathbb{I}K$ modality in our interrogative equational interpreted system models. The closest resemblance to our formalisation is that between the interrogative indistinguishability relation and the indistinguishability relation used in `MCMAS-X` [170] to interpret explicit knowledge.

Our interrogative equational interpreted systems are in the style of the observational systems in [164]. The observational systems in [164] encode notions of security, i.e., secrecy, building on the work of [97]. Nevertheless, the observational systems in [164] are less concrete then the interrogative equational interpreted systems hereby presented (e.g., the observations of agents do not systematically refer to the underlying equational theory of the protocol, but they are arbitrarily chosen sets of facts). To model whether an agent can make a particular observation in a local state, in [164] every agent is supplied with an algorithm that takes an observation and a local state as input and computes whether the agent is capable of making that particular observation in that particular state. The algorithms are based on a set of pre-established deduction rules. Our model loosens that, maintaining agents' interrogations at a higher-level, i.e., no actual deductions or actions are required on the agents' side. We also tailor the model for convergent equational theories. In that sense, an agent is able to interrogate all normal terms and only those (i.e., these interrogations relate to Pucella *et al*'s observations). In our interrogative equational IS, the values of terms in the agent's local state evolve with the equational rewriting embedded in the local evolution functions (i.e., this can be considered as partially encoding Pucella *et al*'s local algorithms).

Other epistemic contexts have also employed predicates that can be likened to our notion of interrogations to interpret a notion of knowledge. Recently, Halpern has used predicates (i.e., "observations" or "experiments") in formalising amongst others notions of probabilistic knowledge in MAS [100] and revision of belief [86]. The modalities thereby explored do not significantly relate to the $\mathbb{I}K$ modality other than by the inherent use of agents' predicates.

The well-established notion of observation or observation function on a point (r, m) of an interpreted environment [146] captures a notion similar to a standard local state in the interpreted systems formalism that we use [169]. In that sense, our interrogations differentiate themselves from Meyden *et al*'s notion of observation on the system's points.

In this chapter we focus on verification aspects, i.e., model checking. We do not report either on the logical properties of the modalities hereby introduced (i.e., QK and $\mathbb{I}K$) or the relations formally established amongst them or with the standard knowledge modality K. These investigations form part of our on-going work.

7.4 Model Checking Knowledge in Interrogative Multiagent Systems Models

In this section we present a model checking algorithm for verifying interrogative knowledge. The methodology allows for the specification and verification of standard interpreted systems equipped with local interrogation-sets, i.e., not only for $\Upsilon_{IS}^{\mathbb{I}E}$.

Algorithm 1 is presented on page 268. It outlines our approach to calculating the set $[\![\Box_j^{\mathbb{I}K}\varphi]\!]$ of states that satisfy the formula $\mathbb{I}K_j\varphi$.

Lines 11 and 13 construct in ϕ_g the conjunction of those interrogations which are evaluated at the current state g (i.e., state g in Algorithm 1). Therefore, for a state g not satisfying φ, a cycle between Line 9 and Line 18 updates the set Y to contain the set of states indistinguishable from g from the point of view of j's interrogations.

The set Y is updated at each step to contain $[\![\phi_g]\!]=\{g' \in G \,|\, g' \sim^{Intr_j} g\}$ for the current g, drawn from X, i.e., $g \models \neg\varphi$. In order to obtain $\{g \in G \,|\, (\exists g' \in G)(g' \sim^{Intr_j} g) \wedge (g \models \neg\varphi)\}$ in Y, it is safe to remove $[\![\phi_g]\!]$ from the iterated set X; i.e., Line 20 does not alter the result computed in Y. Line 20 is an optimisation step to avoid the re-computation of conjunctions identical to ϕ_g. Thus, we reduce the number of iterations in Lines 5–21.

Algorithm 1 *Set* $\textsc{Sat}_{\textsc{ik}}(\varphi : \textsc{Formula}, j : \textsc{Agent})$ *of* $\textsc{States}$

```
1: // $X$ starts as the set of states that refute $\varphi$
2: $X \leftarrow [\![\neg\varphi]\!]$
3: // $Y$ starts as the set of states that refute $\varphi$
4: $Y \leftarrow X$
5: while $X \neq \emptyset$ do
6:    // while there are still states that refute $\varphi$ and have not yet been accounted for (see
          Lines 17–20), process one such state
7:    $g \leftarrow X$.pop()
      // for current $g$, which refutes $\varphi$ by Lines 2, we will use $\phi_g$ to encode the states that
          are indistinguishable from $g$ through the interrogations of agent $j$; $\phi_g$ is a formula
          given by the interrogations of agent $j$ and it starts by being true
8:    $\phi_g \leftarrow .T.$
9:    for $intr \in Intr_j$ do
10:      if $g \in [\![intr]\!]$ then
11:         $\phi_g \leftarrow \phi_g \wedge intr$
12:      else
13:         $\phi_g \leftarrow \phi_g \wedge \neg intr$
14:      end if
15:      // $\phi_g$ is the conjunction of interrogations of agent $j$ that apply to the state $g$
16:   end for
17:   // $Y$ started as $[\![\neg\varphi]\!]$ and it is constructed as the closure of $[\![\neg\varphi]\!]$ under $\sim^{Intr_j}$, i.e., its
          step-by-step union with the set $[\![\phi_g]\!]$ of states that satisfy $\phi_g$
18:   $Y \leftarrow Y \cup [\![\phi_g]\!]$
19:   // $X$ is refined; optimisation step that does not alter $Y$
20:   $X$.remove($[\![\phi_g]\!]$)
21: end while
22: // $\neg Y$ means $G \setminus Y = G \setminus \{g \in G \mid (\exists g' \in G)(g \sim^{Intr_j} g') \wedge (g' \models \neg\varphi)\}$
23: return $\neg Y$
```

In this fashion, between Line 5 and Line 21 the algorithm calculates Y as the set $[\![\lozenge_j^{\mathtt{IK}}]\!]$ of states satisfying the dual of interrogative knowledge for the given agent j with respect to the fact φ (i.e., satisfying $\neg\mathbb{I}K_j\neg\varphi$). As such, at Line 21 the set Y contains all of the states which refute φ and are related under interrogative indistinguishability with at least one other state ($Y \equiv [\![\lozenge_j^{\mathtt{IK}}(\neg\varphi)]\!]$). Finally, at Line 23, the algorithm calculates $\neg Y$ as the set difference between the set of global states G and Y this set. A formal proof of these facts follows.

The algorithm takes an on-the-fly approach to determining the states related under interrogative indistinguishability. The naïve approach to verifying interrogative knowledge would be to pre-process the interrogative indistinguishability relation for each agent. The disadvantage of this technique is that it would require the construction of a BDD relation based on every permutation of clauses for the interrogation-set of each individual agent. This would clearly not be an appropriate solution, as it would exacerbate the state-space explosion problem.

Proposition 7.4.1 *Algorithm 1 calculates the set of states* $[\![\Box_j^{\mathtt{IK}}\varphi]\!]$.

Proof In the computation of the `while` loop (Lines 5–21), consider an arbitrary global state $g \in X$; by the initial construction of X and the modifications applied to X in Algorithm 1, it follows that $g \in G$ and $g \not\models \varphi$.

The `for` (Lines 9–16) iterates over all the interrogations $intr \in Intr_j$ of the agent j at state g. The inner `if-else` statement (Lines 10–14) is exhaustive with respect to the test $g \in [\![intr]\!]$. It follows that the inner loop computes the formula ϕ_g, as follows:

$$\phi_g = (\bigwedge_{\substack{g \models intr_k,\\ intr_k \in Intr_j}} intr_k \wedge \bigwedge_{\substack{g \not\models intr_l,\\ intr_l \in Intr_j}} \neg intr_l)$$

The relation $\sim^{Intr_j} \subseteq G \times G$ (Def. 7.3.8) is overridden here to $[\![\phi_g]\!]$, while preserving the original semantics. As such, $g \sim_{Intr_j} g'$ iff $g' \in [\![\phi_g]\!]$ (1).

At Line 23, when there are no more states to be processed (i.e., $X = \emptyset$), the set Y (modified through Line 18) will have converged as follows:

$$Y = \bigcup_{g \in X} [\![\phi_g]\!]$$

Lines 2 to 21 keep Y invariant with respect to $[\![\neg\varphi]\!]$, i.e., $[\![\neg\varphi]\!] \subseteq Y$ between Lines 4 to 21. As aforementioned, Line 20 is an optimisation step to avoid the re-computation of identical ϕ_g and, thus, it reduces the number of iterations in Lines 5–21. Line 20 does not alter Y (i.e., it eliminates from X states that have already been accounted for in the construction of Y and it does not change the nature of X including $[\![\neg\varphi]\!]$).

Therefore, the value of Y at Line 23 is as follows:

$$\begin{aligned}
Y &= \bigcup_{g\in X}\{g' \in G \mid g' \in [\![\phi_g]\!] \wedge g' \models \neg\varphi\} & \\
&= \bigcup_{g\in X}\{g' \in G \mid g' \sim^{Intr_j} g \wedge g' \models \neg\varphi\} & (\textit{from } 1)\\
&= \{g \in G \mid \exists(g' \in G)(g \sim^{Intr_j} g' \wedge g' \models \neg\varphi)\} & (2)\\
&= [\![\Diamond_j^{\mathtt{IK}}\neg\varphi]\!] & (\textit{from def. of } \mathtt{IK}_j\textit{, Sec. 7.3.2})
\end{aligned}$$

Finally, at Line 23, the algorithm returns $\neg Y$, i.e., $G \setminus Y$. By (2), and dual-modality interpretation ($\neg\Diamond\neg\varphi \equiv \Box\varphi$), the set of states returned is indeed $[\![\Box_j^{\mathtt{IK}}\varphi]\!]$. ▮

Remark 7.4.2 *In actual modelling and implementations, we consider a sub-language of the language $\mathcal{L}'$ where the nesting of modalities $\mathbb{I}K$ and QK is not used.*

An implementation of Algorithm 1 was made in an experimental branch of the model checker `MCMAS`. We refer to this branch as `MCMAS-I`[5] and it is made available at [31].

Succinct Evaluation of Sections 7.2, 7.3 and 7.4

In the last three sections we have progressively introduced the M^E_{IS} and $M^{\mathbb{I}E}_{IS}$ models. These models are similar in the MAS nature to the M_{IS} model in Chapter 3. However, several significant differences arise from the fact that the M^E_{IS} and $M^{\mathbb{I}E}_{IS}$ models are purposely designed to model protocols specified by convergent equational theories. We recall a few of these differences. Firstly, the local states of M^E_{IS} and $M^{\mathbb{I}E}_{IS}$ agents contain terms and not only atomic terms as in M_{IS}. Secondly, in M^E_{IS} and $M^{\mathbb{I}E}_{IS}$, the Environment is a simple observer and, therefore, it encodes a

[5]Andrew V. Jones implemented Algorithm 1 in the `MCMAS-I` branch of `MCMAS`.

weaker thread-model than in M_{IS}. Thirdly, the evolution functions of “honest” agents in M_{IS}^{E} and $M_{IS}^{\mathbb{I}E}$ are more complicated than those in the M_{IS} formalism in Chapter 3. Fourthly, the M_{IS}^{E} and $M_{IS}^{\mathbb{I}E}$ models also simulate the convergent rewriting underlying the equational theories. Lastly, unlike the M_{IS} formalism, the M_{IS}^{E} and $M_{IS}^{\mathbb{I}E}$ systems can model receiver-opaque protocols which are not reducible to receiver-transparent protocols. In fact, all the models in Sections 7.2 and 7.3 are tailored to model ROP rather than RTP (i.e., see the evolution function, the local states of the agents, the abilities of the intruder, etc.). We recall that the need for such models arose from the fact that the simple equality relation used as state-indistinguishability in M_{IS} is too coarse for the modelling of ROP protocols. For instance, in the Pr_1 protocol, the terms $\leq (2, 5)$ and $\leq (3.5)$ are unequal but are equal modulo the equation theory E_1. Therefore, we introduced the quotient indistinguishability relation and the M_{IS}^{E} and $M_{IS}^{\mathbb{I}E}$ models to cater for the formalisation of a class of ROP.

In M_{IS}^{E} we introduced the quotient-knowledge modality to express knowledge modulo the equational theory. We introduced the notion of interrogative knowledge (i.e., a knowledge modality based on local “interrogations” of agents). In particular, interrogative knowledge is attained in $M_{IS}^{\mathbb{I}E}$ by inspection of the normal terms of the underlying rewriting system. Under these circumstances, we have shown that the interrogative knowledge and the quotient knowledge modalities coincide in $M_{IS}^{\mathbb{I}E}$. We have also advanced a model checking algorithm for testing these models against interrogative knowledge.

At page 266 we have evaluated these models by comparison with existing approaches in the related literature.

In the following sections, we are going to exemplify the usage of this methodology on the automatic verification of electronic voting.

7.5 Verifying Multiagent System Models for E-Voting Protocols

In this section we apply the methodologies of Sections 7.2, 7.3 and 7.4 to verify e-voting protocol models against temporal-epistemic specifications of their requirements.

7.5.1 E-Voting Protocols

E-voting protocols are cryptographic protocols used in electoral systems. Some of the requirements of e-voting (e.g., receipt-freeness, coercion-resistance) are rather involved and often more complicated than the usual trace-based properties (e.g., secrecy) traditionally demanded by the simpler authentication and key-establishment protocols. In order to enhance the security of electronic elections, the design of e-voting protocols relies heavily on equational cryptographic theories.

The e-voting requirements aforementioned can be formulated as properties of time and knowledge modulo these equational theories. Recall from Chapter 2, page 33 that vote-privacy requires that, at no point during the voting process or at its completion, is an observer able to know the votes or link them to their voters. Similarly, receipt-freeness requires that, at no point during the voting process or at its completion, is an intruder able to link a voter to his vote even if aided by certain voting material facilitated by this voter. The expressions above are specialisations of anonymity and role-interchangeability properties; the latter found effective epistemic-based formulations in [189, 97]. We will build upon these formulations in [189, 97] and specify e-voting requirements in logics of time and quotient-knowledge.

For the reasons summarised above, we use the methodologies presented in Sections 7.2, 7.3, 7.4 to formalise and automatically analyse the FOO'92 [87] e-voting protocol.

We begin by presenting the FOO'92 protocol. In our description, we use a language close to natural language in order to describe the underlying cryptographic mechanisms.

Example 7.5.1 (A Description of the FOO'92 Protocol)

• ***Phase1***:

Voter*:*

1.1. $v = commit(x, r)$

1.2. $blinded\_v = blind(x, b)$

1.3. $signedByVoter\_blinded\_v = sign(blinded\_v, V)$

1.4. $V \rightarrow A : signedByVoter\_blinded\_v$

Admin*:*

1.5. $checksign(signedByVoter\_blinded\_v) = checksign(sign(blinded\_v, V))$

1.6. $signedByAdmin\_blinded\_v = sign(blinded\_v, admin)$

1.7. $A \rightarrow V : signedByAdmin\_blinded\_v$

Voter*:*

1.8. $signedByAdmin\_v = unblind(signedByAdmin\_blinded\_v, b) = sign(v, admin)$

- ***Phase2****:*

Voter*:*

2.1. $V \rightarrow C : signedByAdmin\_v$ *//on an anonymous channel*

Collector*:*

2.2. $checksign(signedByAdmin\_v) = checksigned(sign(v, admin))$

2.3. $(l, v, signedByAdmin\_v)$

- ***Phase3****:*

Collector*:*

3.1. $C \rightarrow: (l_i, v_i, signedByAdmin\_v_i)$

Voter*:*

3.2. $V \rightarrow C : l, r$

Collector*:*

3.3. $checkcommit(v, r) = checkcommit(commit(x, r), r)$

3.4. $C \rightarrow: x$

FOO'92 starts with a voter deciding upon a vote x and creating its bit-commitment v, i.e., step 1.1 in Example 7.5.1. Then, the voter blinds the bit-commitment, digitally signs it and sends the cipher-text $signedByVoter\_blinded\_v$ to a vote administrator, i.e., steps 1.2–1.4. If the verification of the legitimacy of the voter and of his digital signature succeeds (i.e., step 1.5), then the administrator signs the data. In step 1.7, the administrator sends *signedByAdmin_blinded_v* back to the voter. Once the data is unblinded, e.g., *unblind*(*signedByAdmin_v*, b), the voter uses an anonymous channel [92] to send the result, e.g., *signedByAdmin_v*, to a vote-collector, i.e., step 2.1. If the checks of signatures succeed and once the voting is finished, the collector publishes all the data he has acquired, i.e., steps 2.2, 2.3 and 3.1. The voter is able to identify his commitment and will send back to the collector the entry l associated with his identity and his random number r originally used in the commitment, i.e., step 3.2. Thus, the collector is able to extract the vote x at entry l and publish it, i.e., steps 3.3 and 3.4. A fundamental requirement of FOO'92 is vote-privacy.

7.5.2 $\Upsilon^{\mathbb{I}E}_{IS}$ Formalisations for E-Voting Protocols.

E-Voting Protocol Specification. In Chapters 3, 4 and 5 we used `CAPSL` as the high-level description language for authentication and key-establishment protocols. In this chapter, we use `CAPSL` to specify e-voting protocols given by equational theories.

In Example 7.5.2, we give our specification of the FOO'92 protocol in `CAPSL`, which maintains the intuitive denominations that we employed in Example 7.5.1.

Example 7.5.2 (A CAPSL Description of the FOO'92 Protocol)

```
TYPESPEC VoteFoo;
FUNCTIONS
blind(Field,Nonce): Field;
comm(Nonce,Nonce): Field;
open(Field,Nonce): Field;
sign(Field,PKUser): Field;
unblind(Field,Nonce):Field;
checksign(Field,Pkey):Field;
VARIABLES
m: Field;
p,n:Nonce;
X: PKUser;
AXIOMS
open (comm(p,n), n) = p;
checksign(sign (m, X), pk(X)) = m;
unblind(blind(m,n),n) = m;
unblind(sign(blind(m,n),X),n) = sign(m,X);
END;
```

```
PROTOCOL FOO;

VARIABLES
V,A:PKUser;
C: Node;
l:Atom,FRESH;
v,r,b:Nonce,FRESH;
vote:Field;
comm_v_r,blind_comm_v_r_b:Field;
signedBy_blind_comm_v_r_b_V:Field;
signedBy_comm_v_r_A:Field;
ska:Skey;
skv:Skey;
pkv:Pkey;pka:Pkey;
UNCHECKEDsignedBy_blind_comm_v_r_b_V:Field;
signedBy_blind_comm_v_r_b_A:Field;
UNCHECKEDsignedBy_comm_v_r_A: Field;
CHECKEDcomm_v_r:Field;
DENOTES
comm_v_r = comm(v,r): V;
blind_comm_v_r_b = blind(comm_v_r,b): V;
signedBy_blind_comm_v_r_b_V=sign(blind_comm_v_r_b,V):V;
pkv=pk(V):V;
ASSUMPTIONS
HOLDS V: A,C,v,r,b;
HOLDS A: V,pkv;
HOLDS C: A,V;
MESSAGES
1.V -> A:   V,signedBy_blind_comm_v_r_b_V % UNCHECKEDsignedBy_blind_comm_v_r_b_V;
            blind_comm_v_r_b=checksign(UNCHECKEDsignedBy_blind_comm_v_r_b_V,pkv);

2. A -> V:  sign(blind_comm_v_r_b,A) % signedBy_blind_comm_v_r_b_A;
            signedBy_comm_v_r_A=unblind(signedBy_blind_comm_v_r_b_A,b);

3. V -> C:  signedBy_comm_v_r_A % UNCHECKEDsignedBy_comm_v_r_A;
            CHECKEDcomm_v_r=checksign(UNCHECKEDsignedBy_comm_v_r_A,pk(A));

4. C -> V : l, CHECKEDcomm_v_r;

5. V -> C:  l,r;
          vote=open(CHECKEDcomm_v_r,r);
```

```
6. C -> V:  vote;

GOALS
SECRET v;
END;
```

The equational theory is given in the `TYPESPEC` section of the `CAPSL` description. The `FUNCTIONS` subsection introduces the function symbols to be used, whereas the subsection `AXIOMS` specifies the equations. For instance, `open(comm(p,n),n)=p` symbolises that a bit-commitment can be opened if the receiving agent possesses the random number n used to compose the commitment in the first place. In the `DENOTES` section we defined terms over the signature and the equational theory introduced in section `TYPESPEC`. For instance, `comm_v_r` stands for the bit-commitment of the vote v under the random number r, i.e., `comm_v_r=comm(v,r)`. The qualifier "`:V`" in `comm_v_r:V`, shows that the bit-commitment pertains to a voter-role.

In the `PROTOCOL` section of the `CAPSL` description, we used the `CAPSL` operator "%" [65] to denote that a certain term is opaque and it would eventually be rewritten under the equational theory. Then, we specified the operation to be applied in order to perform this rewriting. To exemplify, in step 2, V receives `sign(blind_comm_v_r_b,A)`, the blinded vote-commitment signed by the administrator. He stores this as an un-deciphered message (hence, `% signedBy_blind_comm_v_r_b_A`). To retrieve more meaning for the newly received data, V needs to unblind the message, i.e., rewrite `signedBy_blind_comm_v_r_b__A` to `signedBy_comm_v_r_A` by applying an `AXIOM` stipulated in the `TYPESEC` section, e.g., `signedBy_comm_v_r_A =unblind(signedBy_blind_comm_v_r_b_A,b)`.

We extended an existing translator [65] from `CAPSL` to `CIL` (`CAPSL` Intermediate Language) [65] to output the rewriting rules for each protocol-role not in pure `CIL`, but in a format which suits better an IS-based semantics. We input the resulting file into a bespoke translator that we designed. This translator is similar in its nature to the `PD2IS` toolkit, presented in Chapter 5. However, the translator used in this chapter is only aimed at dealing with the FOO'92 protocol. Initially, the input is processed into an agent-oriented description of each symbolic protocol role. Next, with a set of role-instantiations, the program produces the agents and the ground $\Upsilon_{IS}^{\mathbb{I}E}$ operational semantics. Finally, it outputs the corresponding `ISPL` files extended with local interrogations as per Section 7.3.3. The programs encoded in these `ISPL` files are specialisations of the $\Upsilon_{IS}^{\mathbb{I}E}$ formalism and are explained in detail in the following. The e-voting specifications in the language $\mathcal{L}'$ are also added in this last step.

The `CAPSL` equational specification for the protocols verified, their respective `CIL`-like files and `ISPL` files are available at [31].

$\Upsilon_{IS}^{\mathbb{I}E}$ Formalisations for E-Voting in `ISPL`. Distinct electoral systems stipulate different e-voting requirements. For instance, some electoral systems may demand a minimum number of voters which can collectively verify the output of the voting procedures. More strongly, other systems may require that every voter is individually able to carry out this verification. Some voting systems demand for receipt-free mechanisms and for guarantees that some particular voting material is unguessable. Other e-voting systems may even demand resistance to coercing agents, etc.

To capture some of these differences in electoral systems, we design three classes of IS-based formalisations, $\mathcal{M}_1$, $\mathcal{M}_2$ and $\mathcal{M}_3$ as specialisations of $\Upsilon_{IS}^{\mathbb{I}E}$. While the design of the classes is ad-hoc and property-driven, it is natural and aligned with e-voting modellings made in the applied-pi formalism [63]. Furthermore, the translator aforementioned processes a `CAPSL` protocol description given at input and automatically generates `ISPL` programs corresponding to formalisations $\mathcal{M}_1$–$\mathcal{M}_3$.

I. The first formalisation ($\mathcal{M}_1$) specialises $\Upsilon_{IS}^{\mathbb{I}E}$ as summarised below.

1. A minimum of two agents representing the voter-role (i.e., *V–role*) are considered.
2. The set *Adj* of terms is enlarged with several flag-variables, e.g., to represent amongst others the end of the different voting phases and the publication of results (see Example 7.5.1).
3. The ranges of terms used are large enough to ensure initial unlinkability between any agent and its data.
4. In any initialisation scenario the communication partners are set to $\perp$ (this is to simulate the anonymous channels assumed in the FOO'92 protocol), cryptographically unguessable data (e.g., the blinding factor used in the blind-commitment) are set by randomisation, the fresh data (e.g., the actual vote x) are initially assigned to $\perp$. Then, the voter-agent will arbitrarily "choose" one concrete value for x.

5. A passive intruder agent called *Attacker* and abbreviated *at* is introduced. This agent has one action denoted *Eavesdrop* and enabled at all states. We use the Environment agent in $M_{IS}^{\mathbb{I}E}$ and the mechanism of local observable variables [127] to model the *Attacker* agent efficiently in `ISPL`.

On the $\mathcal{M}_1$ formalisation above we can specify vote-privacy.

Vote Privacy.

Assume an $\mathcal{M}_1$ formalism as above where:

- $i \in Ag \setminus \{at\}$ is arbitrarily fixed,
- $i' \in Ag \setminus \{i, at\}$,
- $l \in L_j$, $j \in \{i, i'\}$,
- $l.store \supseteq (vote : Range_V(vote))$,
- $v \in Range_V(vote)$ is arbitrarily fixed,
- $v', x \in Range_V(vote)$.

We introduce the propositions $votes(j, x)$ and $votes(j)$ which evaluate to true when $l_j|_{vote=x, endOfPhase1=true}$ and $l_j|_{endOfPhase1=true}$, respectively. They respectively mean that at the end of phase 1 the agent j has chosen to vote x and that the end of phase 1 has simply been reached by agent j, i.e., he has voted some unspecified value.

Let ufair$_1$= $\bigwedge\limits_{i' \neq i} AF(votes(i) \wedge votes(i'))$ and ufair$_2$= $\bigvee\limits_{i' \neq i} AF(votes(i, v) \wedge \neg votes(i', v))$ be unconditional fairness constraints [16] on the corresponding $M_{IS}^{\mathbb{I}E}$ model. We consider only those paths where these fairness constraints hold infinitely often. The constraints respectively denote that on any voting session eventually both agent i and agent i' vote and that there is no protocol session where the voting is unanimous.

We give two specifications for the notion of vote-privacy: *vote privacy* (VP) and *voter-vote unlinkability* (VVU). We use the notation $P_j\varphi$ to mean $\neg QK_j\neg\varphi$, for any agent j.

$$AG(votes(i,v) \rightarrow AG \bigwedge_{v' \neq v} P_{at}(votes(i,v'))) \qquad (VP)$$

$$AG(votes(i,v) \rightarrow \bigwedge_{i' \neq i} \bigwedge_{v' \neq v} [votes(i',v') \rightarrow AGP_{at}(votes(i,v') \wedge votes(i',v))]) \qquad (VVU)$$

VP stipulates that the *Attacker* will never know what a voter has voted, i.e., on all paths, at any point if agent i has voted v it is implied that on no path at no point will the intruder be sure that it was i who voted v. This formulation follows the specifications of anonymity in [97] and their extensions in [189]. According to the latter, it stipulates *privacy up to* $Range_V(vote) \setminus \{v\}$.

Along similar lines, *VVU* stipulates that the *Attacker* will always consider it possible that agents i and i' have cast mutually swapped votes. The formula in VVU follows the specification of *total role-interchangeability* [189], built of the work of Halpern and O'Neill to formalise secrecy [97]. From the point of view of the attacker, it is always possible that i and i' change their roles with respect to casting the votes v and v', respectively.

Note that the design of the $\mathcal{M}_1$ class of IS does not cater for intricate scenarios, e.g., corrupt voter, administrator or collector agents, active coercers, etc. In that sense, we designed the class $\mathcal{M}_1$ of $\Upsilon_{IS}^{\mathbb{I}E}$ formalisation such that we had sufficient resources to specify and verify correctly vote-privacy as in VP and VVU. Along these lines, we attempted to design $\mathcal{M}_1$ by systematically introducing as few specialisations to $\Upsilon_{IS}^{\mathbb{I}E}$ as possible.

II. The second class of IS-based formalisms ($\mathcal{M}_2$) specialises $\mathcal{M}_1$ as summarised below.

1. An additional agent called i_r is introduced to emulate a voter-role with additional capabilities of providing receipts (i.e., details about the voting process) to the attacker. In `ISPL` this is produced in an optimised way: the i_r agent and the *Attacker* agent both observe certain variables of the Environment agent. Additionally, in any initialisation scenario, the i_r agent and the *Attacker* agent share the atomic values that will be used to form the receipts.

2. Several steps are systematically added in the local evolution of the *Attacker* agent in order to model the analysis of receipts.

In the $\mathcal{M}_2$ formalisation above we can additionally specify receipt-freeness.

Receipt-freeness (RF).

Assume an $\mathcal{M}_2$ formalism specialising an $\mathcal{M}_1$ where:

- $i_r \in Ag \setminus \{at\}$ is arbitrarily fixed such that $i \neq i_r$,
- $v_r \in Range_V(vote)$ is arbitrarily fixed,
- $v \in Range_V(vote) \setminus \{v_r\}$,
- $v' \in Range_V(vote)$.

Let ufair$_2$ as above be a fairness constraint on this model. Consider another fairness constraint on this model to be $\underset{i' \notin \{i_r, i\}}{\wedge} (AF(votes(i) \wedge votes(i_r) \wedge votes(i')))$. The latter fairness constraint corresponds to ufair$_1$ in the unwound model of $\mathcal{M}_1$, but it additionally considers the newly adjoined agent i_r. The fairness constraints are self-explanatory. We consider only those paths where these fairness constraints hold infinitely often.

RF stipulates that, whenever agent i counterbalances the vote of the receipt-providing agent i_r, the *Attacker* is not able to link any of the voters to their respective votes at any point. Like VVU, RF follows a total role-interchangeability specification [189]. This specification has a larger domain of three pivot-agents and their respective votes (i.e., even if the receipt-providing agent is included, the attacker still considers it possible for the agents to have swapped roles).

$$\begin{aligned}
&\text{for } v \in Range_V(vote) \setminus \{v_r\},\\
&AG(votes(i_r, v_r) \wedge votes(i, v) \rightarrow\\
&\bigwedge_{i' \notin \{i, i_r\}} \bigwedge_{v'} [votes(i', v') \rightarrow AGP_{at}(votes(i, v_r) \wedge votes(i', v) \wedge votes(i_r, v'))]) \quad \text{(RF)}
\end{aligned}$$

Note that any $\mathcal{M}_2$ model subsumes an $\mathcal{M}_1$ model. Therefore, vote-privacy can be specified and verified on $\mathcal{M}_2$ models also. The design of $\mathcal{M}_2$ does not additionally capture intricate corruption e.g., active Dolev-Yao attackers [71] manipulating the receipts. We designed the class $\mathcal{M}_2$ of models in order to be able to specify and verify e-voting requirements beyond vote-privacy, e.g., receipt-freeness as in RF. Along these lines, we attempted to design $\mathcal{M}_2$ by systematically introducing as few specialisations to $\mathcal{M}_1$ as possible (e.g., the receipt-providing agent i_r and an adapted *Attacker*).

III. The final formalisation ($\mathcal{M}_3$) specialises $\mathcal{M}_2$ in that an agent called i_c is added. This agent is based on the receipt-providing i_r agent in $\mathcal{M}_2$. However, its corrupt behaviour is exacerbated: the i_c agent exchanges more data with the *Attacker* agent, possibly even i_c's vote, i.e., the *Attacker* agent coerces the i_c agent to vote in a certain way.

On the $\mathcal{M}_3$ formalisation above we can additionally specify coercion-resistance.

Coercion-resistance (CR).

CR is the correspondent of RF in $\mathcal{M}_2$ in a specialised $\mathcal{M}_3$ model. Like VVU and RF, CR follows a total role-interchangeability specification [189]. This specification is stronger than RF in that even if the corrupt agent is included, the attacker still considers it possible for the agents to have swapped roles.

$$\begin{aligned}
&\text{for } v \in \mathit{Range}_V(\mathit{vote}) \setminus \{v_c\},\\
&AG(votes(i_c, v_c) \wedge votes(i, v) \rightarrow\\
&\bigwedge_{i' \notin \{i, i_c\}} \bigwedge_{v'} [votes(i', v') \rightarrow AGP_{at}(votes(i, v_c) \wedge votes(i', v) \wedge votes(i_c, v'))]) \quad \text{(CR)}
\end{aligned}$$

Note that an $\mathcal{M}_3$ model is an $\mathcal{M}_2$ model. Therefore, vote-privacy and receipt-freeness can be specified and verified on $\mathcal{M}_3$ model also. The design of $\mathcal{M}_3$ does not additionally capture intricate corruption, e.g., active Dolev-Yao attackers pursing the coercion. By contrast, the $\mathcal{M}_3$ models only enhance the *Attacker* in $\mathcal{M}_2$ models such that it communicates with the coercible agent i_c. We designed the class $\mathcal{M}_3$ of models in order to be able to specify and verify e-voting requirements beyond vote-privacy and receipt-freeness, e.g., coercion-resistance as in CR. Along these lines, we attempted to design $\mathcal{M}_3$ by systematically introducing as few specialisations to $\mathcal{M}_2$ as possible (e.g., the additional coercible-agent and an adapted *Attacker*).

7.5.3 Experiments

In this section we present our verification results on the FOO'92 protocol models analysed and relate them to existing work.

The machine used for the following evaluation was based on an Intel Core 2 Duo processor clocked at 3.00 GHz, with a 6144 KiB cache. The machine ran 32-bit Fedora Core 12, kernel

2.6.32.10.

We automatically generated the $\mathcal{M}_1$, $\mathcal{M}_2$ and $\mathcal{M}_3$ formalisations of FOO'92 in `ISPL`. For that, we used the `JAVA` program aforementioned (which embeds our modified translator for `CAPSL` to `CIL`). The `ISPL` files are in the region of 8000 lines of code. The generation takes approximately 15 seconds. Therefore, in the following, we only report and discuss the results of the actual verification.

We recall that `MCMAS-I` is the branch of `MCMAS` containing an implementation of our model checking algorithm for the $\mathbb{I}K$ modality. We use `MCMAS-I` to verify the `ISPL` programs for the FOO'92 protocol. `MCMAS-I` was linked against `CUDD` release 2.4.2.

In Table 7.1, the leftmost column shows the class of model considered. The results presented reflect the average of two `MCMAS-I` verification runs of each model considered. The *Memory* column shows the average memory usage in each run. The *Time* column gives the CPU time of the `MCMAS-I` processes. Both time and memory usage were measured using `tstime`. The *States* column reports the number of reachable states in each class of model verified with `MCMAS-I`.

Table 7.1 Averaged Multiple Experiments on FOO'92 Models for E-Voting

Model	Formula	Memory (KiB)	Time (s)	States
$\mathcal{M}_1$	VP/VVU	176032	66441	$6.69 \cdot 10^{11}$
$\mathcal{M}_2$	(Weaker) RF	175496	66168	$6.69 \cdot 10^{11}$
$\mathcal{M}_3$	VP/VVU	181926	70401	$6.69 \cdot 10^{11}$

7.5.3.1 Discussion of Results

The *Formula* column reports the strongest e-voting specification that was found to hold on the model. We found vote privacy (VP and VVU) to hold on all three classes of models considered. Receipt-freeness (RF) was found to be refuted on $\mathcal{M}_2$ and $\mathcal{M}_3$ models, i.e., a path was found where eventually the intruder is able to link the receipt-providing agent and its vote. However, deeper studies showed that a logically weaker form of RF holds on $\mathcal{M}_2$ and $\mathcal{M}_3$ models:

for $v \in \mathit{Range}_V(\mathit{vote}) \setminus \{v_r\}$

$$AG(votes(i_r, v_r) \wedge votes(i, v) \rightarrow$$
$$\bigwedge_{i' \notin \{i, i_r\}} \bigwedge_{v'} [votes(i', v') \rightarrow AFP_{at}(votes(i, v_r) \wedge votes(i', v) \wedge votes(i_r, v'))]) \text{ (weaker } RF) \;.$$

This finding expresses that on $\mathcal{M}_2$ and $\mathcal{M}_3$ models there exists no path where, at all points, is the intruder able to link i_r to his vote. We have also found that slight modifications of the *Attacker* capabilities lead to different verification results on RF and CR. This is aligned with the statements [159] of one the protocol designers. On $\mathcal{M}_3$ models for the FOO'92 protocol, we found that there exists a path where eventually the coercion, as modelled, brings voter-vote linkability.

A full Dolev-Yao model was not encoded mainly because e-voting protocols assume anonymising procedures and advanced cryptographic primitives, hence a full Dolev-Yao is considered unrealistic. Moreover, in our MAS setting with a bounded number of sessions, the intricacy of the e-voting models showed that even relatively small increases the capabilities passive intruder affect the performance considerably. In the general setting of an unbounded number of protocol sessions, it is yet unknown if e-voting properties (i.e., properties which are not trace-based) are decidable under a full Dolev-Yao model for protocols specified under equational theories.

We refer the reader to our `ISPL` files [31] to find the complete set of verified models and specifications which specialised RF and CR.

As aforementioned, our VP, RF, CR specifications are expressions of state-based properties inspired by anonymity formulations in [189, 97], recalled at page 2.2.3.2. This differs from the applied-pi approach in [63], where vote-privacy, receipt-freeness and coercion-resistance are given as process equivalences. Their models are said to be expressible in `ProVerif` [23], but verification with `ProVerif`[6] can be carried out only when the specifications reduce to trace-based properties.

While an infinite number of symbolic protocol sessions is approached in [63], we verify a bounded number of ground protocol sessions. This theoretically means that more attacks against e-voting properties could be found in [63]. Nevertheless, the satisfactions or refutations of e-voting specifications as reported in Table 7.1 for FOO'92 protocols are aligned with the pen-and-paper analyses [63] of the same protocol. The thread-model in [63] is also that of a passive attacker, which in points is weaker that our thread-model.

[6]For a debrief on ProVerif, see Chapter 2, page 47.

In contrast with our automatic approach, the analysis in [63] is mainly based on manual proofs and, while details are not given, it is said to be partially supported by the tool `ProVerif`. However, for certain models in [63], `ProVerif` might not terminate. Verification times are not mentioned in [63] for the case where they used `ProVerif`. Therefore, we cannot compare our methodology with the one in [63] with respect to the verification times or the size of the unwound models obtained respectively.

In [63], the techniques do not refer specifically to protocols that are described by convergent theories, but to e-voting protocols. In that sense, in [63] the equations and models are created in an ad-hoc manner, but pen-and-paper, satisfaction proofs are presented not only for the FOO'92 protocols, but also for the Okamoto [159] and Lee [121] protocols. We have also verified several models of the Okamoto e-voting protocol [159], applying our methodology. The `CAPSL`, `CIL` and `ISPL` files corresponding to these protocols are to be found in [31], along with the files for the FOO'92 protocol hereby discussed. Details about the verification of the models for the Okamoto protocol are given in Appendix C.

Building on the theoretical results that observational equivalence in applied pi calculus is decidable under convergent equational theories and passive intruders [1], a tool for handling more properties than `ProVerif` (i.e., not only reachability) has recently emerged [47]. The tool supports the specification and verification of a subset of the class of convergent equational theories. A deduction system is hard-coded and implemented for different e-voting primitives (e.g., bit-commitment). In that sense, the approach is less general than ours. Like ProVerif, it is semi-decidable (i.e., it might not terminate, but when it terminates it terminates with the correct answer). Unlike in our case, [47] can handle an unbounded number of protocol sessions, similarly to `ProVerif`. In [47], the emphasis is put on designing the e-voting primitives in a theory of applied pi calculus and practical results are not mentioned. Therefore, with respect to the verification times or the size of the unwound models, we cannot draw a compelling comparison between our approach and the methodologies based on applied pi calculus. In theory, the protocols that could be verified in [47] are the same as in [63]. Whilst in [47], the terminology "knowledge" is used, it does not refer to standard, Kripke semantics but to evaluation of predicates in the context of applied pi processes. Therefore, we compare to [47] in a similar way that we compare to [63], with the exception that –like us– [47] also falls into the category of *automatic* verification of e-voting protocols. Whilst both the aforementioned applied pi approaches are purposely designed for e-voting verification, the methodology we introduced in this chapter can be applied to generic systems modelled as interrogative interpreted systems or interrogative equational interpreted systems.

Conclusions on the MAS Approach to Equationally Specified Protocols

In this chapter we have presented a methodology of modelling multi-session executions of advanced security protocols as multiagent systems. Whilst the formalism is similar in its MAS nature to the M_{IS} model introduced in Chapter 3, its essentially novel features are the following:

- the systematic modelling of executions of protocols specified under convergent equational theory as specialised interpreted systems;
- the modelling of executions of a class of receiver-opaque protocols that are not reducible to receiver-transparent protocols as specialised interpreted systems;
- the introduction of a knowledge modality appropriate in the context of convergent equational theory describing security protocols and of related, more general knowledge modalities;
- the systematic modelling of executions of e-voting protocols as specialised interpreted systems and their verification against suitable temporal-epistemic formulations of their requirements.

By introducing a knowledge modality evaluated modulo an equational theory, the framework opens up a well-founded way of specifying and verifying all receiver-opaque protocols. Two opaque terms are told apart if they are distinguishable modulo the underlying cryptographic operators. The modelling (e.g., of the agents' states containing non-atomic terms, of the local evolution function, etc.) employed in this chapter caters for the general modelling of RO protocols as interpreted systems.

In this chapter, unlike in some of the previous ones, we focused rather on the verification methodology than on the generation of models. In that sense, we presented only the application of

$M_{IS}^{\mathbb{I}E}$ to verify an e-voting RO protocol. Its models have been automatically generated by a bespoke translator. Aspects related to the modalities introduced (i.e., axiomatisation, relations between them) are the subject of current and future work.

Evaluations of each model and formalisation in this chapter were presented at the end of the sections where they were introduced (i.e., page 258, page 266 and page 270). A comparison between our verification methodology and relevant formalisms based on applied pi calculus were discussed in the previous section. Our IS-based models and the algebraic ones in [63] follow similar intuitions. However, the underlying formalisations are completely distinct. Our interrogative knowledge modality can be likened to observational equivalence in applied-pi. In formalising our specifications, we follow the footsteps of [189]. Therefore, our expressions of e-voting requirements are based on logics of time and knowledge. By contrast, in [63] the e-voting requirements are formulated as process equivalences. Most importantly, our methodology is more general than those in applied pi; we hereby apply the techniques to e-voting, whilst interrogative IS and its modalities facilitate more generic encodings.

In [78] an e-voting protocol is formalised in an ad-hoc manner using dynamic epistemic logic. The protocol is analysed, but performance details are not presented. Apart from the use of non-classical logics, our methodology does not compare with [78].

We exploited our modified translator from `CAPSL` to `CIL`-like hereby presented to create a fully-fledged, automatic tool similar to `PD2IS` in Chapter 5. It deals with the systematic and automatic generation of $\Upsilon_{IS}^{\mathbb{I}E}$ models for any ROP (i.e., from `CAPSL` to `CIL`-like to `ISPL`). Therefore, we extended the approach hereby presented to a systematic methodology for analysing ROP. It is aimed at modelling authentication receiver-opaque protocols (under more powerful adversary models) and at verifying them against suitable quotient-knowledge modalities. See Appendix C for a discussion.

Chapter 8

Conclusions

Motto: "In my end is my beginning."
(T. S. Elliot)

In this chapter we assess the achievements of this thesis and draw a comparison with related work. We conclude with possible future directions.

Section 8.1 summarises the main contributions of each chapter. Then, Section 8.2 evaluates these in the context of the related work presented in Chapter 2. In light of this analysis, Section 8.3 underlines the main overall achievements of the thesis. Section 8.4 finalises with possible future directions and further developments of this line of work.

8.1 Summary of Chapters' Achievements

Chapter 1 stated the motivations of this thesis, anticipating its main contributions. An in-depth summary of related work and, in general, the relevant background of this thesis were presented in Chapter 2.

With Chapter 3 we introduced Υ_{IS}, a novel and systematic modelling based on temporal-epistemic logic for a large class of security protocols and for their respective security goals (i.e., for receiver-transparent[1] protocols). Thus, the current line of research reinstates the original ISO expressions [82, 85, 84, 83] of authentication requirements through well-founded formulations in logics of time and knowledge. Whilst Chapter 3 follows the non-classical logics ideas in the seminal work of BAN-like formalisms [42,95,5,191,184], it advances well-founded and automatable methodologies for protocol verification based on logics of time and knowledge.

In Chapter 4 we gave a guarantee of correctness of our M_{IS} MAS-based model with respect to classical, trace-based approaches [74, 188] for the verification of a bounded number of receiver-transparent protocol sessions. Naturally, this guarantee can only apply to security requirements which can be expressed in both approaches (i.e., initial secrecy and agreement on initials). If traditional `CAPSL` models validate/refute a standard secrecy `CAPSL` goal then our unwound M_{IS} model also validates/refutes the corresponding formulation(s) of the goal. Nevertheless, our approach supports the systematic specification of many more security requirements.

In Chapter 5 we presented `PD2IS`, the first fully automatic toolkit to generate and verify MAS-based models for security protocols against temporal-epistemic specifications of their requirements. We used `PD2IS` to verify several authentication and key-establishment protocols drawn from known repositories [48, 119]. We reported and discussed the results. Conventional searches for attacks against standard security goals were presented and analysed. Moreover, to increase the relevance of the study, new approaches to the analysis were investigated (e.g., the organised search of attacks, following a set of protocol scenarios). We also compared the performance of `PD2IS` with that of other state-of-the-art tools in the field of protocol analysis. We discovered that these performances are aligned in the case of finding well-known attacks, whereas in the case of general, protocol analysis some unbounded-protocol verification tools [61, 180] outperform our methods. Nevertheless, we report on exploring novel settings, inherent to our temporal-epistemic setting.

In Chapter 6 our AI-inspired setting permitted us to take protocol verification further, into automatic reasoning about protocol attacks. To do so, we introduced and formalised the notion of detectability, i.e., groups of agents being theoretically able to detect a protocol attack and/or early signs of the attack. We extended `PD2IS` to support the generation of models and formulae for the analysis of detectability of protocol failures. The latter allowed for the verification of several well

[1]See Definition 3.1.3.

known, attack-prone protocols with respect to the (un-)detectability of the failures to which they are exposed.

In the past, manual inspection of protocol executions for inter-session errors was attempted [136]. The tests applied were simplistic (e.g., based on comparisons of values) and trace-based. The CTLK-based detectability tests that we obtain automatically have a larger purpose (i.e., active attack detection by groups of agents) and are not trivially reducible to simpler, trace-based error-finding tests.

Chapter 7 presented Υ_{IS}^{OE}, a systematic model for receiver-opaque[2] protocols. We also designed the model checking methodology to verify these models. In particular, our formalisation referred to protocols expressed under convergent equational theories. For the modelling and verification of these protocols, new knowledge modalities were introduced. The latter allowed us to give theoretically compelling answers to questions like: "Can Alice confirm the identity of her communication partner, though she only sees his digital signature?", "Can the intruder tell apart the respective votes of Alice and Bob, though he cannot understand all encrypted parts?", etc. The main application of Chapter 7 resided in automatically verifying e-voting protocols specified by convergent equational theories. We discussed the results and positioned them within the community of e-voting protocol verification (e.g., [63]).

Appendix A gives additional details on Chapter 3 and Chapter 5 (i.e., on the formalism and tools for receiver-transparent protocols).

Appendix B gives additional details on Chapter 4 (i.e., on the proof that our MAS model is aligned with traditional, trace-based semantics of security protocols).

Appendix C generalises the ideas in Chapter 7, presents an automatic tool to generate and verify MAS models for receiver-opaque protocols and reports the results of the analysis of several such models against the CTLK-based specifications of their requirements.

[2]See Definition 3.1.5.

8.2 Comparison with Related Work

In this section we draw a comparison between our line of work and existing state-of-the-art in the related fields of research. At the end of each chapter or large section, relevant aspects of related work were compared and constrasted with the material presented therein. This section is a systematisation and an extension of those presentations. The motivations and contributions of this thesis give the main criteria to be considered in this comparison. These criteria are as follows: *1)* the protocol model employed; *2)* the specification of security requirements; *3)* the verification methodology used; *4)* the intruder model applied; *5)* the approach of encoding the underlying cryptographic aspects.

8.2.1 Protocol Model

A Comparison with Authentication Logics. In the nineties, authentication protocols were specified and analysed using several, purpose designed logic languages [41, 42, 95, 5, 191] (the BAN, GNY, AT and VO authentication logics, respectively). The main criticisms [156] brought to these formalisms resided on the lack of axiomatisations and well-founded semantics. Nevertheless, Syverson's late authentication logic (e.g., SVO [184]) was equipped with a multidimensional $S5$ semantics. Its downfall remained the manual, error-prone analysis of the protocols, i.e., the lack of verification tools to support it.

As in the SVO [184] authentication logic, the protocol models we propose follow an underlying multidimensional $S5$ semantics. To benefit from the recent advancement of model checking temporal-epistemic specifications [127, 110, 89], our language is based on temporal-epistemic logic. This differentiates us from authentication logics, which employed belief operators in their protocol models. Our choice of logic language and semantics relieves us from the manual analysis of the protocols, specific to BAN-like logics[3]. More precisely, our approach allows us to use the model checker `MCMAS` [127] for the automatic verification of our temporal-epistemic specifications of protocol models. Henceforth, the BAN annotation and idealisation is replaced in our case by the `PD2IS` tool that automatically generates the protocol models from high-level protocol descriptions.

[3]For a summary BAN protocol verification, see Chapter 2, page 39.

In BAN-like logics, the protocol theory was embedded in the logic language[4]. For instance, $A \overset{k}{\longleftrightarrow} B$ denoted that the key k will not be learned by any parties, other than A and B. These kind of specifications lead to protocol-dependency and expressivity problems [156] in BAN-like formalisations of protocols (e.g., quantification ambiguities, impossibility to formulate key-possesion, etc). To avoid this type of issues, our framework does not embed the protocol theory into the logic language. Instead, starting from a high-level protocol description, we design a denotational and an operational semantics applicable systematically to entire classes of protocols (e.g., RTP and ROP).

In trying to resolve several of BAN's expressivity problems, the later GNY authentication logic [95] differentiated between inherently owned, locally generated messages and/or received messages. We pursue GNY's approach in this direction and, in the model in Chapter 3, we separate terms into semantically distinct categories: $OwnedAtoms^A$, $LearnedAtoms^A$, $Composites$, Msg^A, etc. Moreover, our model further refines such distinctions into a denotational semantics for protocol executions (as Chapter 3, Section 3.3.1 has shown).

A Comparison with Epistemic-based Approaches to Security Protocol Modelling. There are few notable approaches to *cryptographic* protocol verification which are based on epistemic logic [78,125,90]. We will now compare and contrast each of these three frameworks with our security protocol specification formalism.

Our protocol model is based on a multiagent system (MAS) semantics for temporal-epistemic logic. In that sense, it differs from [78] where dynamic epistemic logic is employed. The methodology in [78] is applied to one e-voting protocol only. By contrast, we systematically approach several classes of security protocols in an unitary fashion. Moreover, the modelling of protocols in [78] is ad-hoc, whereas our approach is systematic and automatable (as Chapter 3 and Chapter 5 has shown). A further comparison cannot be drawn as in [78] the details about the e-voting modelling or about the performance of the analysis are scarce.

Our formalism follows the steps of the LDYIS [125] model. This framework proposed a temporal-epistemic language and an interpreted system [161] formalisation of the NSPK [155] protocol. LDYIS advanced a matched send-receive semantics for encoding the protocol communication. This implied that the modelling assumed trusted communication channels [145]. Therefore, certain

[4]For a summary on the BAN language, see Chapter 2, page 39

protocol attacks (e.g., binding attacks [145, 192]) cannot be captured under the LDYIS formalism. To verify the LDYIS specifications, a bounded model checking algorithm was presented in [125]. The impact of LDYIS was limited given the ad-hoc protocol model and the lack of a systematisation for their methodology. For an extended summary of LDYIS, please refer to Chapter 2, page 62.

Like LDYIS, we also use the interpreted system formalism in our MAS approach to protocol modelling. However, in our Υ_{IS} formalisation the modelling of agents' local states is different from the one in LDYIS. More precisely, unlike in LDYIS, Υ_{IS} agents do not "store" *letters* or *addresses*. Most importantly, Υ_{IS} agents "recall" atoms, whereas the LDYIS agents "store" messages. These imply that the size of Υ_{IS} states is considerably smaller than the size of LDYIS states. Unlike in our Υ_{IS} formalism, the aspects of term-ranges which also impacts heavily the size of the models is not explored in LDYIS. We believe that these discrepancies lie in the respective natures of the approaches: the Υ_{IS} formalism is designed for automatic verification with symbolic model checkers, whereas LDYIS is a theoretical platform, suited for bounded model checking. As such, in our Υ_{IS} model in Chapter 3, we model local states more judiciously than in LDYIS, in a fashion aimed at an applied verification with `MCMAS`.

The indistinguishability relation we use in the $\Upsilon_{IS}^{\mathbb{I}E}$ formalisation to model equationally-specified protocols is not the equality relation as in LDYIS. As such, additional to LDYIS, we introduce the new modality of interrogative knowledge to enquire the acknowledgements of facts in light of the cryptographic operations in the protocol. This allows us to model and analyse systematically the class of receiver-opaque protocols, which cannot be readily encoded in LDYIS. Similar comparisons about the indistinguishability relations and the class of protocols covered can be drawn between our approach and the epistemic-based framework in [78].

Our protocol model relaxes the matched send-receive conditions in the LDYIS models and adopts the more standard matching receive [175] conditions. This can be anticipated even from the state modelling, as the Υ_{IS} states do not contain equivalents of the LDYIS addresses or the LDYIS letters. All in all, our receive semantics implies that the Υ_{IS} framework models a more general setting than the one encompassed in LDYIS (i.e., we model untrusted communication [145], as opposed to the trusted channels assumed in LDYIS). It follows that, unlike in LDYIS, binding protocol attacks can theoretically be found using our methodology. In fact, practice has proven that indeed our methodology does capture binding attacks (see Chapter 5, page 209). Also, our model approaches a more general protocol theory than the one in LDYIS, introducing a denotational protocol semantics (see Chapter 3). In this way, we apply our methodology systematically to several classes of protocols. Moreover, unlike the case of LDYIS, we analyse the satisfaction of formulae specifying security requirements comprised in a methodical and automatically generated taxonomy. The verification methodology we employ is symbolic model checking [143]. The LDYIS advanced a bounded model checking procedure for the NSPK model, but no implementation and actual experimentation with it was presented.

As aforementioned, another epistemic-based formalism for cryptographic protocol analysis is presented in [90]. The formalisation in [90] is protocol dependent and restricted to several types of protocols (i.e., onion routing [92], crowds [192]). By contrast, our approach is applicable in an unitary fashion to different classes of protocols (e.g., authentication, key-establishment). Nevertheless, modelling similarities between our work and [90] are more prevalent in the case where we apply our formalisations to more specific investigations (e.g., e-voting, in Chapter 7). The logic language in [90] is based on epistemic logic, whereas the temporal aspect is embedded in the protocol theory. The latter differentiates it from our line of work where the specification language most used is CTLK. However, the semantics adopted in [90] is that of distributed programs [80], which can be assimilated to our interpreted system-based semantics. The practical verification aspects in [90] are scarcely addressed, thus impairing a possible comparison with our verification methods.

A Comparison with Traditional Approaches to Security Protocol Modelling.

In the following we refer to those formalisms of security protocol modelling that have become the *de facto* approaches to protocol analysis over the last two decades (i.e., all but the more recent formalisations [63] driven by applied pi calculus [2]). None of these formalism is based on epistemic logics.

Approaches of formal protocol verification assume either a bounded or an unbounded number of protocol sessions. The latter assumption yields undecidability issues [172] and semi-decidable tools [23,180,33,61]. The former assumption can be employed [67,187] to develop decidable methods for protocol analysis. However, it truncates the space of searching for attacks, i.e., a bounded number of sessions might not exhibit all possible attacks on a protocol.

Like [73, 188, 138, 72, 163, 59] and several formalisms [19, 58] in the state-of-the-art AVISPA tool [9], we take the approach of modelling and verifying a bounded number of concurrent protocol sessions.

Also, our protocol modelling is fully ground (i.e., without variables), unlike some of the recent symbolic or on-the-fly approaches [190,19] to bounded protocol verification. The main implications of taking a fully ground approach to verifying a bounded number of protocol sessions are most prominent in the evaluations of the tools we use (i.e., the performance metrics and/or the attacks found). In a nutshell, the increase in number of protocol sessions modelled worsens the performance of any fully ground approach at a faster rate than that of a symbolic or on-the-fly approach. This has been discussed in detail in Section 8.2.3, where we compare our tools with other protocol analysis software. In our case, a fully ground modelling of the protocols was imposed by the verification tools at hand (i.e., the `MCMAS` model checker for temporal-epistemic logics).

After the decline of authentication logic, rewriting mechanisms [44, 67, 175] became the main focus of protocol verification. From early attempts [153] to the backends of the successful state-of-the-art AVISPA project [9] (e.g, the on-the-fly model checker OFMC [19], the constraint-solving based model checker AtSe [190], the SAT-based model checker SAT-MC [58], TA4SP [25]), protocol verification tools rely mostly on (multiset) rewriting semantics for protocol execution. These tools use multiset rewriting protocol roles in restricted form[5] which can be likened to simplified form of Υ_{IS} agents controlled by the local evolution functions. Nevertheless, differences are apparent (e.g., a multiset rewriting role contains atomic propositions evaluated over a trace, whereas a state of an Υ_{IS} agent contains term-value pairs evaluated in an unwound model).

Formalisms [73] based on multiset rewriting (MSR) also provided many of the theoretical decidability and complexity results about analysing security protocols. Other trace-based formalisms [172, 188] have drawn full correspondences with MSR approaches. If generated under a specific set of substitutions[6] (i.e., via algorithm tr), the protocol model we use is homomorphic with the bounded protocol model [188] and thus, with the multiset rewriting [44] formalism (see Chapter 4). This entails that respective formulations of standard trace-based properties are

[5]For restricted form protocol role in multiset rewriting, see page 41.

[6]These specific substitutions are extending those in M_{CAPSL}; see Chapter 4, page 128.

equally validated/refuted in our model and in the traditional models for authentication protocols (see Chapter 4).

Well-established protocol verification approaches other than rewriting-based are founded on: inductive proofs [162, 163], Horn clauses [24], strand spaces [79] and process algebra [139, 138, 72]. Such protocol modellings are generally dissimilar to our temporal-epistemic approach to security analysis. Nevertheless, a correspondence between strand spaces and general MAS modellings has been drawn in [98].

A Comparison with Upcoming Approaches to Security Protocol Modelling.

Emerging from the more traditional algebraic approaches (e.g., CSP [174]), certain algebraic calculi are the upcoming formalisms for certain areas of protocol verification (e.g., analysis of modern cryptographic primitives). Such calculi are the spi-calculus [3] or, the more recent, applied pi calculus [2]. Whilst these are not extensively applied to authentication and key-establishment protocols, they have succesfully modelled and verified [63] more advanced protocols (e.g., e-voting protocols).

In Chapter 7 we propose a multiagent system model for (e-voting) protocols specified by convergent equational theories. The models for e-voting protocols are similar in principle to the work in [63]. In fact, the quotient indistinguishability relation[7] that we introduce in Chapter 7, page 256, can be assimilated to static-equivalence[8] in applied pi calculus. Furthermore, our novel quotient knowledge modality, introduced in Chapter 7 can be likened to the observational equivalence[9] used in [63].

However, the applied pi approach to protocol modelling views an unbounded number of protocol sessions, whereas our formalism assumes a bounded number of such sessions. The implications by this fact were generally discussed above and specifically discussed in Chapter 5, Section 5.4.3 and Chapter 7, page 283, in the context of the `ProVerif` tool versus our tools.

Moreover, essential specification differences lie in the fact that [63] presents an applied pi calculus methodology, whilst we take an approach based on multiagent systems. The operational semantics

[7]For details on indistinguishability relations, see Chapter 2, page 25.

[8]For details on static equivalence, see Chapter 2, page 61.

[9]For details on observational equivalence, see Chapter 2, page 52.

in applied pi[10] is fully based on the idea of communicating processes. The applied pi operational semantics is tailored on equational theories and it is much more specific than the IS semantics we adopt (i.e., the set of actions a process can perform is coarsely defined by a general, applied pi operational semantics, recalled in Chapter 2). Nevertheless, these aspects make it more protocol-oriented than the IS formalism used in our Υ_{IS} modelling. As a consequence, models in applied pi can look more compact than our Υ_{IS} formalism, partially due to its symbolic nature and suitability for cryptographic modelling. A comparison in terms of requirement specifications follows.

8.2.2 Security Requirement Specifications

In this subsection we will relate our means of specifying security protocol requirements to other approaches in the field.

Lowe's hierarchy [137] is a list of increasingly stronger expressions of authentication requirements. These were introduced using natural language [137]. This hierarchy has been later formalised algebraically in [138, 72] and in linear temporal logic in [59]. Another attempt of formalising authentication requirements is that of Gollman [93]. He started from the ISO standards of these requirements and expressed them by means of authentication logics, using belief operators. For details on such formalisations of security goals, please refer to Section 2.2.3.1.

In our approach, security requirements and properties related to them are expressed as temporal-epistemic formulae. Firstly, for each standard authentication and key-establishment goal we give a taxonomy of temporal-epistemic formulae that relates to the Lowe's hierarchy for authentication. Secondly, we introduce a taxonomy of temporal-epistemic formulae as possible specifications for each BAN-like goal [41] (e.g., comprising doxastic or epistemic assertions). We propose novel specifications based on the distributed knowledge modality [80]. Furthermore, we map security goals pertaining to knowledge into formulae akin to Gollman's formalisations.

In doing so, we come close to the specifications in authentication logics [42, 95, 191, 184]. In particular, we cover the expressivity of the VO logic [191]. The implicit nesting of beliefs in VO, recalled at page 51, is made explicit in our systematic translation of complex, epistemic and

[10]For the operational semantics of applied pi, see [2] or, for a summary, refer to page 46.

doxastic `CAPSL` goals (see Section 3.3.3 and, e.g., Example 3.3.26). In this fashion, mutual agreement [111] is naturally expressed in our formalism by a nesting of two knowledge operators. New aspects of epistemic-based specifications are explored in our methodology. Most prominently, we use the distributed knowledge operator to express both `CAPSL` goals (e.g., see Chapter 3, page 115) and novel, detectability specifications (in Chapter 6). Moreover, the `PD2IS` toolkit support the automatic translation of these, as Chapter 5 explained and Figure 5.6 briefly illustrated. Also, the systematic manner in which we produce the taxonomy of specifications detaches us from the protocol-dependency and inherent ambiguities (e.g., implicit universal quantification, impossibility to express possesion of keys, etc.) present in authentication logics.

At the same time, the aforementioned taxonomies of temporal-epistemic formulae show that our specification language for authentication and key-establishment requirements is more expressive than the reachability specifications used in the rewriting-based formalism [73, 188]. Also, the language we employ is richer than the rewriting-based ones used in the state-of-the-art AVISPA [9] project.

In Chapter 7 we express e-voting requirements in a way which is in keeping with the taxonomy[11] of anonymity-like properties in [189]. This taxonomy refers to Halpern's expressions of secrecy [97] and to a series of information-hiding and anonymity specifications built on top of them. These were recalled at page 52. We use the core of the role interchangeability [189] specification to express VVU, RF and CR and the privacy up to $Ag \setminus \{Attacker\}$ [97] to express VP. Whist we maintain the epistemic aspects as introduced in [189, 97], we add a temporal aspect in keeping with the natural-language expression of e-voting goals (see page 34) and the intuition behind applied pi specifications of voting requirements.

The anonymity-based expressions in Chapter 7 can be partially related to observational equivalence[12] in applied-pi.

It is less clear how our formulations of security requirements relate to expressions of protocol goals based on algebraic equivalences, which were used in algebraic approaches [72, 138] other than applied pi. In [138], the formulations of security goals had, apart from the algebraic aspect, a trace-

[11]This is presented in Chapter 2, page 52.

[12]This is recalled in Chapter 2, page 52.

based formulation; this is correlated to the FDR model checker used in tandem with CASPER [138]. These formulations convey an implicit linear time aspect and embed implicit expressions of past actions, e.g., when *alice* finished her session, *alice* and *bob* will have agreed on data ds. In that sense, our specification language based on computational time logic with future-based operators is different from such algebraic specifications with embedded linear time, past aspects. Whilst it is known that a fragment of CTL with pure past operators (PCTL) can be translated into CTL pure future operators [120], the translation yields exponentially larger, non-intuitive formulae. Moreover, it would imply a perfect recall semantics (rooted in the anchored view of satisfaction [141]) that is not supported by MCMAS and that would exacerbate the size of our protocol models. We therefore rely on a systematic, intuitive mapping of Lowe's hierarchy for agreement onto our CTLK specifications (see Chapter 3, Section 3.3.3).

8.2.3 Methodology and Tools for Security Protocol Verification

We devise a toolkit called PD2IS, which produces one or several interpreted system specifications of a protocol's execution starting from a protocol high-level description given in CAPSL (*Common Authentication Protocol Specification Language*) [64, 66, 151]. Each interpreted system specification is described by PD2IS in the specification language of ISPL (*Interpreted System Programming Language*). ISPL is the input language of the MCMAS model checker [127]. Our PD2IS toolkit is linked to MCMAS. The latter is called for the sequential verification of all ISPL files produced by PD2IS. Hence, the verification of our protocol models is fully automated.

Using translators from a high-level language to a low-level one is common in protocol verification [9, 139, 64, 151]. For instance, the backends of the state-of-the-art AVISPA toolkit [9] use translators from HLSPL [157] to IF [167]. In particular, translators from CAPSL to low-level languages have been employed as connectors to several protocol verification tools and formalisms (e.g., to NRL [35], to Athena [150], to a Prolog-based verification tool for strand spaces [151]). Our PD2IS toolkit is therefore aligned with such connectors. However, the ISPL programs that PD2IS produces are purposely optimised for the use with the MCMAS model checker.

Driven by the nature of verification tools for temporal-epistemic logic, we employ a fully ground (i.e., with no un-instantiated variables) verification of a bounded number of sessions of a security protocol. This is the case of other tools for protocol analysis [138, 144]. By standard verification parameters (e.g., time to find an attack, the largest state-space that we can explore), our methodology outperforms these other tools [138, 144] handling a bounded number of protocol sessions in a fully ground model.

Naturally, the performance of analysing fully ground protocol models worsens as the number of protocol sessions considered increases (hence, the size of the state-space implied). Therefore, approaches [190, 58] that verify a bounded number of protocol sessions modelled symbolically (i.e., with un-instantiated variables) perform better than our fully ground approach when large models are tested. When average-size models (e.g., up to 5 concurrent sessions) are verified, the respective performances of our methodology and that of symbolic, bounded protocol verification [190, 58] are similar. For details, please refer to Chapter 5.

State-of-the-art, on-the-fly methodologies for verifying a bounded number of protocol sessions recently emerged [19] (i.e., methodologies using symbolic models constrained at verification-time). These should theoretically outperform both our approach and traditional, symbolic approaches. Nevertheless, practice [166, 138] has confirmed Lowe's intuition [178] that attacks are normally exhibited/found on small numbers of protocol sessions. On such "small systems" [178] the differences in performance between our framework and on-the-fly approaches to verifying a bounded number of protocol sessions are negligible. We consider that this is mainly due to our `MCMAS`-wise optimisation for the ground, `ISPL` protocol models that `PD2IS` generates. Nevertheless, the `OFMC` on-the-fly model checker [19] obtains better verification times than `PD2IS` and `MCMAS`, in the rare case when larger systems need to be verified. For details, please refer to Chapter 5.

The fact that we verify fully ground models (i.e., without variables) raises a particular interest on the size of specifications and of the unwound models. Indeed, some of the `ISPL` files for the Υ_{IS} specifications generated by `PD2IS` are long (e.g., the FOO'92 [87] protocol analysed in Chapter 7 is described in over 8000 `ISPL` lines). The unwound model is of the order of 10^{11} states. Whilst the specifications for symbolic verification are shorter, the search-space explored by other bounded protocol verification methods is similar (i.e., double exponential in the size of the original specification [188]). Automatic symbolic verification methods do not generate a low-level specification as detailed as the `ISPL` files we use (i.e., their intermediate formats (IF [167]) are still symbolic). As such, the time/CPU spent by `PD2IS` to generate our `ISPL`, fully ground specifications can be related to similar metrics in other state-of-the-art tools (e.g., back-end of AVISPA [9]).

We apply our verification toolkit to protocols drawn from well-established repositories [48, 119]. In finding known attacks (e.g., [136]) against what we call *receiver-transparent*[13] protocols, our MAS methodology performs comparably to existing state-of-the-art tools [9]. For details on these aspects, see Section 5.4.3. Nevertheless, our more expressive specification language allows us to analyse new settings and find novel attacks residing on the knowledge of the protocol participants. For details, see Chapter 5, Table 5.2 and the corresponding discussions.

There are numerous notable protocol-analysis frameworks [149, 34, 144, 138, 138, 163, 79, 181, 44, 73, 188, 59, 116] which, like ours, verify a bounded number of protocol sessions. Theoretically, assuming an unbounded number of protocol sessions can lead to finding more attacks which cannot be exhibited by the more constrained assumptions of a bounded number of protocol sessions. Nevertheless, methods of analysing an unbounded number of protocol sessions [33, 23, 61, 180] are only semi-decidable (i.e., they might not terminate, but if they terminate they provide the correct answer).

Therefore, it remains to compare the actual performance between bounded and unbounded protocol verification methods when the latter terminates. As Chapter 5 has shown, verification times are almost constant when using unbounded protocol verification irrespective of the number of sessions considered. Therefore, our methodology as well as all other bounded-number-of-sessions approaches to protocol verification are outperformed by [33,23,61,180] if a large number of sessions is considered (e.g., usually over 5 concurrent sessions). Fortunately, as suggested by [166, 178, 138], attacks are usually exhibited on a small number of sessions (e.g., 2–5 concurrent sessions). In these cases, our methodology is comparable with approaches considering an unbounded number of protocol sessions. For details on these aspects, see Section 5.4.3.

Other comparison criteria to be discussed are the use of model checking and of temporal logic in our verification methods. Recently, approaches of model checking of security protocols based on linear temporal logic have emerged [59, 10]. The differences between our methodology and these

[13]For details, see Definition 3.1.3.

approaches with respect to the specification aspect is clear (i.e., CTLK versus LTL). Apart from the aforementioned aspects (i.e., bounded versus unbounded, symbolic versus ground), the comparison in performance lies mainly in the usual distinctions between LTL and CTL model checking [16]. In particular, we are not aware of any methodology other than the one hereby presented to employ logics of time and knowledge in a systematic model for automatic verification of security protocols, based on symbolic model checking.

Protocol verification methods based on model checking approaches, other than those already discussed, use refinement [72] to establish properties of models expressed in CSP [174]. Described briefly at page 48, `FDR` is a model checker used by Lowe to verify security protocol CSP models generated with `CASPER` [138]. The underlying theoretical aspects of a CSP-refinement checker like `FDR` are essentially different to the OBDD-based, symbolic model checker `MCMAS` that we use. Recently, contract signing protocols have been verified with the probabilistic model checker `PRISM` [118]. Again, the underlying mechanisms in the modelling and the verification thereby used (i.e., Markov chains) are essentially different from the methodology presented in this thesis.

8.2.4 Detection of Security Protocol Failures

In [91] a proof-of-concept framework where protocol participants have the ability to conduct active retaliations against the intruder is presented. Hence, the model in [91] is not a Dolev-Yao attacker[14] model, but a *general-attacker* thread model where all agents are potentially able to become attackers. The design views the actual collaboration of protocol participants to mount retaliations against the intruder. A single application is thereby presented. It is based on the NSPK protocol, which is manually extended with a rule for the retaliation step. The data proposed for the retaliation-driven communication is defined ad-hoc by the specifier. In [91], LTL [165] is used to encode the notion of a notion of attack trace extended towards retaliations. No aspects of the participants' knowledge is employed to achieve the retaliation, communication being enforced onto specific participants upon manually designated data.

In Chapter 6 we defined the notion of *detectability*, i.e., if and when groups of agents are

[14]For "Dolev-Yao" attacker, see [71] or Chapter 2, page 35.

theoretically able to detect protocol failures in an MAS protocol model. We thereby presented a systematic methodology of specifying and verifying detectability specifications in any security protocol. We carried out experiments on several automatically generated models for authentication and key establishment protocols. Unlike [91], our formalism refers to a detection at a theoretical level (i.e., agents do not collaborate actively or share actual information to detect an attack). Furthermore, unlike [91], our protocol model is a multiagent system model where knowledge is not static (i.e., it does not restrict itself to possesion of terms), but follows a Kripke semantics. The model is not trace-based, but built on top of an $S5_n$, temporal-epistemic semantics. Henceforth, our model maintains the traditional Dolev-Yao thread model and the aspects of detection on the intruder side are also explored. To sum up, we distinguish ourselves from this line in that we use a knowledge-centred, CTLK-based framework to investigate a systematic and automatic way to model and verify groups of honest agents theoretically detecting attacks and protocol failures in standard Dolev-Yao thread model.

In the nineties, Lowe attempted methods of manual inspection of protocol executions for inter-session errors [136]. The tests thereby applied were simplistic (e.g., based on comparisons of values) and trace-based. In turn, our detectability schema is expressed using CTLK and the specifications do not trivially reduce to simpler, trace-based error-finding tests. Also, our schemata is automatically obtained (i.e., it is not obvious to see its relations with some manual inspection of the protocol). Like [136], our specifications also refers to protocol attacks. Nevertheless, they additionally refer to more general protocol failures (i.e., we introduce the notion of `pre-attack` and study its detectability). Moreover, our methodology has a larger purpose (i.e., agents actively detecting the attack mounted on a protocol).

The detectability methodology takes our approach to verification further into possible intrusion-detection (IDS) [140] techniques. The latter are aimed at actual attack detection. In turn, our formalism refers to the theoretical possibility to detect an attack in real-time. While IDS uses offline training and add-on machinery (e.g., honey pots, filters, firewalls), we rely solely on the protocol participants and the runtime of the protocol. We estimate that the computational overhead of a detectability implementation in an every-day system would be greater than using offline IDS (i.e., the size of the unwound model and the communication links needed between groups of agents).

However, such an implementation would eliminate the false-positive and false-negative yielded by machine learning techniques, normally used in offline IDS.

8.2.5 Knowledge Modulo Cryptographic Equational Theories

In Chapter 7 we presented a methodology of verifying protocols specified under convergent [124] equational theories. This approach has been compared and contrasted with existing, related work at page 258, page 266 and page 270. From the point of view of the knowledge modalities thereby introduce, we reiterate some of these comparisons.

We introduced a notion of knowledge modulo the convergent equational theory. In the models in Chapter 7, the specification of agents is enriched with a set of predicates. These are used to approximated the knowledge modulo the convergent equational theory by another modality called *interrogative knowledge*.

The general idea of enriching the specification of agents with a set of predicates is akin to certain, seminal approaches. As such, the mechanisms and interpretations of this interrogative knowledge are in the style of Pucella's deductive algorithmic[15] knowledge [164]. Nevertheless, differences are apparent, e.g., no actual deductions or actions are required on the agents' side in our models. Then, we tailor the formalisation for convergent equational theories. In that sense, an agent is able to "interrogate" all normal terms and only those (i.e., these *interrogations* relate to Pucella's observations [164]). In our interrogative equational IS, the values of terms in the agent's local state evolve with the equational rewriting embedded in the local evolution functions (i.e., this can be considered as partially encoding Pucella's local algorithms). For a more detailed comparison on the matter, refer to the evaluation presented at page 266.

Agents' observations (i.e., predicates interpreted at local states) have recently been used also in probabilistic knowledge in MAS [100] and revision of belief [86]. The modalities thereby entailed are different from the interrogative knowledge modality introduced in Chapter 7.

The underlying indistinguishability relation modulo the equational theory that we model in Chapter 7 is similar to the notion of static equivalence[16] in applied pi calculus [2]. Furthermore,

[15]This is summarised in Chapter 2, page 61.

[16]This is recalled in Chapter 2, page 61.

our novel quotient knowledge modality, introduced in Chapter 7 can be likened to the observational equivalence[17] used in [63].

The IS-based protocol models presented in Chapter 7 simulate the rewriting entailed by the equational theory. In that sense, these MAS models come closer to the MSR model [73] than the model introduced in Chapter 3.

To create the aforementioned models, we implement another compiler from `CAPSL` to `ISPL`. In order to embed the equational rewriting in the IS model, our tool uses the rewriting simulations present in `CIL` (`CAPSL` Intermediate Language). In fact, we exploited a version of the `CAPSL` to `CIL` translator [151] (summarised in Chapter 2, page 57) which we modified to fit our purposes, i.e., to render some of the IS semantics.

We devise an algorithm of model checking for our interrogative knowledge modality. The implementation of this model checking algorithm follows some of the ideas previously used in `MCMAS-X` [130] (recalled in Chapter 2, page 61). Nevertheless, it takes an on-the-fly approach to the model checking procedure.

With respect to the main criteria above, we compared and contrasted the contributions of this thesis with relevant existing work in the fields of protocol analysis and AI-inspired verification. Moreover, in our multiagent system approach to cryptography analysis we fall into the category of methodologies used for protocol verification via AI-inspired techniques, together with [128,130,109, 132,125]. The (dis)similarities in comparison with [125] have been discussed. With the other lines ([128,130,109,132]) we do not substantially compare otherwise but in the AI-inspired techniques and/or logic languages used.

8.3 Overall Contributions

This work reinstates the seminal ideas of authentication logics to use non-classical logics to model and verify security protocols. Moreover, through well-foundedness, systematisation and automation, it succeeds in the points where the early BAN-like logics failed (e.g., a compelling semantics, protocol-independent language, un-ambiguous formalisations). As preempted in Chapter 1, Sec-

[17]For details on observational equivalence, see Chapter 2, page 52.

tion 1.2, the main contributions of this thesis are:

1. defining the class of receiver-transparent and the class of receiver-opaque protocols and purposely optimising the modelling and verification of each class in the context of temporal-epistemic logic and MAS model checking;
2. well-founded MAS models for security protocol executions;
3. a well-founded notion of cryptographic knowledge to be used in our approach;
4. model checking methodologies for this novel cryptographic knowledge modality;
5. tools for the automatic generation of several flavours of all the MAS models formally introduced; these cater for the automatic generation of temporal-epistemic formulae for each requirement of the underlying protocols;
6. defining the notion of detectability (i.e., groups of agents being theoretically able to detect a protocol attack and/or early signs of the attack) and further studying its applicability in a systematic, automatic way using our MAS setting.

In a nutshell, the above contributions enabled us to:

- model check MAS models for security protocols (e.g., authentication, key-establishment, e-voting protocols) against both standard and novel, temporal-epistemic expressions of their security requirements;
- find attacks in novel settings driven by the epistemic framework;
- exploit the MAS setting to investigate security protocols beyond attack-finding into active attack detection by the parties involved.

8.4 Future Work

In this section we present possible directions of future work related to this thesis.

To the best of our knowledge, in parallel to the investigations reported in this thesis, researchers at the Polish Academy of Sciences have been studying aspects of bounded model checking of temporal-epistemic properties of security protocols. A comparison between this thesis and their approach is envisaged, given the intersection points in the underlying logic formalisations. Moreover, a performance comparison between our symbolic model checking methodology and their bounded model checking is of interest.

In Chapter 4 we presented a proof that relates our methodology to standard, trace-based semantics used in the verification of authentication and key-establishment protocols. However, for e-voting, receiver-opaque protocols our formalism comes closer to process algebra approaches (i.e., cryptographic indistinguishability corresponds to applied pi static equivalence, etc.). The actual formal study of the relations between applied pi modellings and our logic-based Υ_{IS}^{IE} formalisation remains an open research topic.

For certain approaches to symbolic protocol verification [2] computational soundness proofs have been advanced. This implied showing that the guarantees offered by a symbolic verification formalism hold for computational verification models as well (see page 61). Along these lines, it is of interest to investigate the computational soundness of our temporal-epistemic based approach to protocol verification. Similar to these, we are currently investigating the properties enjoyed by the knowledge modalities introduced in Chapter 7 and the axiomatisation of the logics presented therein.

An immediate line of work is to use our detectability framework to automate the synthesis of attack-free security protocols.

At the same time, we are investigating methods of systematic MAS modelling for classes of protocols that are not encompassed by this thesis (e.g., contract signing protocols).

We conclude that this thesis has offered AI-inspired automatic methodologies for protocol verification. It investigated classical lines of protocol testing (e.g., attack-finding) and novel aspects into security protocol analysis (e.g., our MAS-driven detectability techniques).

Bibliography

[1] M. Abadi and V. Cortier. Deciding Knowledge in Security Protocols Under Equational Theories. In J. Diaz, J. Karhumäki, A. Lepistö, and D. Sannella, editors, *Proceedings of the 31st International Colloquium on Automata, Languages and Programming (ICALP'04)*, volume 3142 of *Lecture Notes in Computer Science*, pages 145–160, Turku, Finland, 2004. Springer Berlin/Heidelberg.

[2] M. Abadi and C. Fournet. Mobile Values, New Names, and Secure Communication. In H. Nielson, editor, *Proceedings of the 28th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL'01)*, pages 104–115, London, UK, 2001. ACM Press New York.

[3] M. Abadi and A. D. Gordon. A Calculus for Cryptographic Protocols: the Spi Calculus. In *Proceedings of the 4th ACM conference on Computer and communications security (CCS'97)*, pages 36–47, Zurich, Switzerland, 1997. ACM Press New York.

[4] M. Abadi and P. Rogaway. Reconciling Two Views of Cryptography (The Computational Soundness of Formal Encryption). *Journal of Cryptology*, 20(3):395–395, 2007.

[5] M. Abadi and M. Tuttle. A Semantics for a Logic of Authentication. In L. Logrippo, editor, *Proceedings of the 10th annual ACM Symposium of Principles of Distributed Computing (PODC'91)*, pages 201–216, Montreal, Canada, 1991. ACM Press New York.

[6] R. Alur, T. Henzinger, F. Mang, S. Qadeer, S.Rajamani, and S. Tasiran. MOCHA: Modularity in Model Checking. In A. J. Hu and M. Vardi, editors, *Proceedings of the 10th Interna-*

tional Conference on Computer Aided Verification (CAV'98), volume 1427 of *Lecture Notes in Computer Science*, pages 521–525, Vancouver, Canada, 1998. Springer Berlin/Heidelberg.

[7] R. Alur, T. A. Henzinger, and O. Kupferman. Alternating-Time Temporal Logic. *Journal of the ACM*, 49(5):672–713, 2002.

[8] R. Amadio and D. Lugiez. On the Reachability Problem in Cryptographic Protocols. In C. Palamidessi, editor, *Proceedings of the 11th International Conference on Concurrency Theory (CONCUR'00)*, volume 11 of *Lecture Notes in Computer Science*, pages 380–394, Pennsylvania, US, 2000. Springer Berlin/Heidelberg.

[9] A. Armando, D. Basin, Y. Boichut, Y. Chevalier, L. Compagna, J. Cuéllar, P. Drielsma, P. Héam, O. Kouchnarenkoand, J. Mantovani, S. Mödersheim, D. von Oheimb, M. Rusinowitch, J. Santiago, M. Turuani, L. Viganò, and L. Vigneron. The AVISPA Tool for the Automated Validation of Internet Security Protocols and Applications. In K. Etessami and S. Rajamani, editors, *Proceedings of the 17th International Conference on Computer Aided Verification (CAV'05)*, volume 3576 of *Lecture Notes of Computer Science*, pages 281–285, Edinburgh, UK, 2005. Springer Berlin/Heidelberg.

[10] A. Armando, R. Carbone, L. Compagna, J. Cuéllar, and M. Tobarra. Formal Analysis of SAML 2.0 Web Browser Single Sign-on: Breaking the SAML-based Single Sign-on for Google Apps. In V. Shmatikov, editor, *The 6th ACM Workshop on Formal Methods in Security Engineering (FMSE'08)*, pages 1–10, Virginia, US, 2008. ACM New York.

[11] A. Armando and L. Compagna. An Optimized Intruder Model for SAT-based Model-Checking of Security Protocols. In A. Armando and L. Viganò, editors, *Electronic Notes in Theoretical Computer Science*, volume 125, pages 91–108. Elsevier Science Publishers, 2005.

[12] B. Arnold. *Logic and Boolean algebra.* Prentice-Hall, New York, 1962.

[13] N. Asokan, V. Shoup, and M. Waidner. Asynchronous Protocols for Optimistic Fair Exchange. In *Proceedings of the 1998 IEEE Symposium on Research in Security and Privacy (IEEE-S&P'98)*, pages 86–99, California, US, 1998. IEEE Computer Society Press.

[14] E. Astesiano, M. Bidoit, H. Kirchner, B. Krieg-Brückner, P. Mosses, D. Sannella, and A. Tarlecki. CASL: the Common Algebraic Specification Language. *Theoretical Computer Science*, 286(2):153–196, 2002.

[15] F. Baader and T. Nipkow. *Term Rewriting and All That.* Cambridge University Press, New York, 1998.

[16] C. Baier and J. Katoen. *Principles of Model Checking.* The MIT Press, 2008.

[17] H. Barendregt. *The Lambda Calculus – its Syntax and Semantics*, volume 103 of *Studies in Logic and the Foundations of Mathematics.* North-Holland, 1984.

[18] D. Basin. Lazy Infinite-State Analysis of Security Protocols. In R. Baumgart, editor, *Proceedings of the International Exhibition and Congress on Secure Networking (CQRE'99)*, pages 30–42, Düsseldorf, Germany, 1999. Springer-Verlag London.

[19] D. Basin, S. Mödersheim, and L. Viganò. OFMC: A Symbolic Model Checker for Security Protocols. *International Journal of Information Security*, 4(3):181–208, 2005.

[20] M. Ben-Ari, Z. Manna, and A. Pnueli. The Temporal Logic of Branching Time. In *Proceedings of the 8th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL'81)*, pages 164–176, Williamsburg, Virginia, 1981. ACM New York.

[21] M. Ben-Or, O. Goldreich, S. Micali, and R. Rivest. A Fair Protocol for Signing Contracts. *IEEE Transactions on Information Theory*, 36(1):40–46, 1990.

[22] P. Blackburn, M. de Rijke, and Y. Venema. *Modal Logics.* Cambridge University Press, 2001.

[23] B. Blanchet. An Efficient Cryptographic Protocol Verifier Based on Prolog Rules. In P. Pandya and J. Radhakrishnan, editors, *Proceedings of the 14th IEEE Computer Security Foundations Workshop (CSFW'01)*, pages 82–96, Novia Scotia, Canada, 2001. IEEE Computer Society Press.

[24] B. Blanchet and A. Podelski. Verification of Cryptographic Protocols: Tagging Enforces Termination. In A. Gordon, editor, *Proceedings of the 6th International Conference on Fondations*

of Software Science and Computation Structures (FoSSaCS'03), pages 136–152, Warsaw, Poland, 2003. Springer Heidelberg/Berlin.

[25] Y. Boichut, N. Kosmatov, and L. Vigneron. Validation of Prouvé protocols using the automatic tool TA4SP. In *Proceedings of the 3rd Taiwanese-French Conference on Information Technology (TFIT'06)*, pages 467–480, Nancy, France, 2006. —.

[26] I. Boureanu, M. Cohen, and A. Lomuscio. A Compilation Method for the Verification of Temporal-Epistemic Properties of Cryptographic Protocols. In *Informal Proceedings of the Joint International Workshop on Automated Reasoning for Security Protocols Analysis and Issues in the Theory of Security (ARSPA-WITS'09)*, pages 117–131, York, UK, 2009.

[27] I. Boureanu, M. Cohen, and A. Lomuscio. Model Checking Temporal Epistemic Specifications for Authentication Protocols Compiled into Interpreted Systems. *Journal of Applied Non-Clasical Logics*, 19(4):463–487, 2009.

[28] I. Boureanu, M. Cohen, and A. Lomuscio. PD2IS: Protocol Descriptions to Interpreted Systems, version 0.90. http://www.doc.ic.ac.uk/ibourean/pd2is, 2009.

[29] I. Boureanu, M. Cohen, and A. Lomuscio. PD2**I**-IS: Protocol Descriptions to Interrogative Interpreted Systems, version 0.80. http://www.doc.ic.ac.uk/ibourean/pd2ois, 2010.

[30] I. Boureanu, A. Jones, and A. Lomuscio. Automatic Verification of Temporal-Epistemic Logic under Convergent Equational Theories. submitted.

[31] I. Boureanu, A. V. Jones, and A. Lomuscio. MCMAS-**I** — Model Checker for Interrogative Interpreted Systems. http://www-lai.doc.ic.ac.uk/mcmas/mcmas-o/.

[32] I. Boureanu, A. Lomuscio, and M. Cohen. Model Checking Detectability of Attacks in Multi-agent Systems. In W.van der Hoek, G. Kaminka, Y. Lespérance, M. Luck, and S. Sen, editors, *Proceedings of the 9th International Conference on Autonomous Agents and Multi-Agent systems (AAMAS'10)*, pages 691–698, Toronto, Canada, 2010. IFAAMAS Press.

[33] L. Bozga, Y. Lakhnech, and M. Périn. HERMES: An Automatic Tool for Verification of Secrecy in Security Protocols. In W. Hunt and F. Somenzi, editors, *Proceedings of the 15th International Conference of Computer Aided Verification (CAV'03)*, volume 2725 of *Lecture Notes of Computer Science*, pages 219–222, Colorado, US, 2003. Springer Berlin/Heidelberg.

[34] M. Bozzano and G. Delzanno. Automated Protocol Verification in Linear Logic. In F. Pfenning, editor, *Proceedings of the 4th ACM SIGPLAN International Conference on Principles and Practice of Declarative Programming (PPDP'02)*, pages 38–49, Pennsylvania, US, 2002. ACM Press New York.

[35] S. Brackin, C. Meadows, and J. Millen. CAPSL Interface for the NRL Protocol Analyzer. In *Proceeedings of the 2nd IEEE Workshop on Application-Specific Software Engineering and Technology (ASSET'99)*, pages 64–73, Dallas, US, 1999. IEEE Computer Society, ACM New York.

[36] T. Brauner and S. Ghilardi. First-Order Modal Logic. In P. Blackburn, J. van Benthem, and F. Wolter, editors, *Handbook of Modal Logic*, pages 549–620. Elsevier Science Publishers Ltd. Essex, UK, 2007.

[37] M. Browne, E. Clarke, and O. Grümberg. Characterizing Finite Kripke Structures in Propositional Temporal Logic. *Theoretical Computer Science*, 59(1-2):115–131, 1988.

[38] E. Bryant. Graph-Based Algorithms for Boolean Function Manipulation. *IEEE Transactions Computing*, 35(8):677–691, 1986.

[39] J. Bull and D. Otway. A Nested Mutual Authentication Protocol. *Operating Systems Review*, 33(4):42–47, 1999.

[40] R. Burkhardt. *UML: Unified Modeling Language.* Addison-Wesley, 1997.

[41] M. Burrows, M. Abadi, and R. Needham. Authentication: A practical study in belief and action. In V. Moshe and M. Kaufman, editors, *Proceedings of the 2nd Conference on Theoretical Aspects of Reasoning about Knowledge (TARK'88)*, pages 325–342, California, US, 1988. ACM New York.

[42] M. Burrows, M. Abadi, and R. Needham. A Logic of Authentication. *ACM Transactions on Computer Systems*, 8(1):18–36, 1990.

[43] M. Burrows, M. Abadi, and R. Needham. A Logic of Authentication. Technical report, DEC–SRC, http://www.hpl.hp.com/techreports/Compaq-DEC/SRC-RR-39.html, 1990.

[44] I. Cervesato, N. Durgin, P. Lincoln, J. Mitchell, and A. Scedrov. A Meta-Notation for Protocol Analysis. In P. Syverson, editor, *Proceedings of the 12th IEEE Computer Security Foundations Workshop (CSFW'99)*, pages 55–69, Mordano, Italy, 1999. IEEE Computer Society Press.

[45] B. Chellas. *Modal logic: An Introduction.* Cambridge University Press, 1980.

[46] A. Cimatti, E. M. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, and A. Tacchella. NuSMV2: An Open-Source Tool for Symbolic Model Checking. In E. Brinksma and K. Larsen, editors, *Proceedings of the 14th International Conference on Computer Aided Verification (CAV'02)*, volume 2404 of *Lecture Notes in Computer Science*, pages 359–364, Copenhagen, Denmark, 2002. Springer Heidelberg/Berlin.

[47] Ş. Ciobâcă, S. Delaune, and S. Kremer. Computing Knowledge in Security Protocols under Convergent Equational Theories. In R. Schmidt, editor, *Proceedings of the 22nd International Conference on Automated Deduction (CADE'09)*, Lecture Notes in Artificial Intelligence, pages 355–370, Montreal, Canada, 2009. Springer.

[48] J. Clark and J. Jacob. A Survey of Authentication Protocol Literature. http://www.cs.york.ac.uk/~jac/papers/drareviewps.ps, 1997.

[49] J. Clark and J. Jacob. Clark-Jacobs Library in CAPSL. http://www.csl.sri.com/projects/capsl/, 1999.

[50] E. Clarke, E. Emerson, and A. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 8(2):244–263, 1986.

[51] E. Clarke, O. Grumberg, and D. Peled. *Model Checking.* MIT Press, Massachusetts, US, 1999.

[52] M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, and J. Quesada. The Maude System. In P. Narendran and M. Rusinowitch, editors, *Proceedings of the 10th International Conference on Rewriting Techniques and Applications (RTA'99)*, volume 1631 of *Lecture Notes of Computer Science*, pages 240–243, 1999.

[53] E. Cohen. TAPS: A First-Order Verifier for Cryptographic Protocols. In P. Wadler, editor, *Proceedings of the 13th IEEE workshop on Computer Security Foundations (CSFW'00)*, http://computer.org/proceedings/csfw/0671/0671toc.htm, pages 144–158, Cambridge, UK, 2000. IEEE Computer Society. Online Publication.

[54] M. Cohen. *Logics of Knowledge and Cryptography : Completeness and Expressiveness.* PhD thesis, Royal Institute of Technology (KTH), Stockholm, 2007.

[55] M. Cohen and M. Dam. A Complete Axiomatization of Knowledge and Cryptography. In L. Ong, editor, *Proceedings of the 22nd Symposium on Logic in Computer Science (LICS'07)*, pages 77–88, Wroclaw, Poland, 2007. IEEE Computer Society Press.

[56] M. Cohen, M. Dam, A. Lomuscio, and H. Qu. A Symmetry Reduction Technique for Model Checking Temporal-Epistemi Logic. In C. Boutilier, editor, *Proceedings of the 21st International Joint Conference on Artificial Intelligence, (IJCAI'09)*, pages 721–726, Passadena, US, 2009. AAAI Press.

[57] H. Comon-Lundh and V. Cortier. Computational Soundness of Observational Equivalence. In P. Syverson and S. Jha, editors, *Proceedings of the 15th ACM conference on Computer and communications security (CCS'08)*, pages 109–118, Virginia, US, 2008. ACM New York.

[58] L. Compagna. *SAT-based Model-Checking of Security Protocols.* PhD thesis, Università degli Studi di Genova and the University of Edinburgh (joint programme), September 2005.

[59] R. Corin, S. Etalle, and A. Saptawijaya. A Logic for Constraint-based Security Protocol Analysis. In I. Arce, editor, *Proceedings of the 2006 IEEE Symposium on Security and Privacy (IEEE-S&P'06)*, Oakland, US, 2006. IEEE Computer Society.

[60] V. Cortier. Observational Equivalence and Trace Equivalence in an Extension of Spi-Calculus. Application to Cryptographic Protocols. Technical Report LSV-02-3, Laboratory of Specification and Verification, ENS Cachan, Paris, France, 2002.

[61] C. Cremers. The Scyther Tool: Verification, Falsification, and Analysis of Security Protocols. In A. Gupta and S. Malik, editors, *Proceedings of the 20th International Conference on Computer Aided Verification (CAV'08)*, volume 5123 of *Lecture Notes in Computer Science*, pages 414–418, Princeton, USA, 2008. Springer.

[62] C. Cremers, P. Lafourcade, and P. Nadeau. Comparing State Spaces in Automatic Protocol Analysis. In *Formal to Practical Security*, volume 5458/2009 of *Lecture Notes in Computer Science*, pages 70–94. Springer Berlin / Heidelberg, 2009.

[63] S. Delaune, S. Kremer, and M. Ryan. Verifying Privacy-Type Properties of Electronic Voting Protocols. *Journal of Computer Security*, 17(4):435–487, 2009.

[64] G. Denker and J. Millen. CAPSL and CIL Language Design. Technical Report SRI-CSL-99-02, SRI International Computer Science Laboratory, Washington, US, 1999.

[65] G. Denker and J. Millen. CAPSL Intermediate Language. In N. Heintze and E. Clarke, editors, *Proceedings of the Workshop on Formal Methods and Security Protocols (FMSP'99)*, Trento, Italy, 1999. ACM New York.

[66] G. Denker and J. Millen. CAPSL Integrated Protocol Environment. In *Proceedings of DARPA Information Survivability Conference (DISCEX'00)*, volume 1, pages 207–221, South Carolina, US, 2000. IEEE Computer Society.

[67] G. Denker and J. Millen. Modeling Group Communication Protocols Using Multiset Term Rewriting. *Electronic Notes Theoretical Computer Science*, 71:20–39, 2002.

[68] H. van Ditmarsch, W. van der Hoek, and B Kooi. Playing cards with Hintikka. An introduction to dynamic epistemic logic. *Australasian Journal of Logic*, pages 108–134, 2005.

[69] H. van Ditmarsch, W. van der Hoek, R. van der Meyden, and J. Ruan. Model Checking Russian Cards. In *Proceedings of the 3rd International Workshop of Model Checking and Artificial Intelligence (Mochart'05)*, volume 149(2), pages 105–123. Electronic Notes in Theoretical Computer Science, 2005.

[70] H. van Ditmarsch, J. Ruan, and L. Verbrugge. Sum and Product in Dynamic Epistemic Logic. *Journal of Logic and Computation*, 18(4):563–588, 2006.

[71] D. Dolev and A. Yao. On the Security of Public-Key Protocols. *IEEE Transactionson Information Theory 29*, 29(2):198–208, 1983.

[72] B. Donovan, P. Norris, and G. Lowe. Analyzing a library of security protocols using Casper and FDR. In E. Clarke and N. Heintze, editors, *Proceedings of the Workshop on Formal Methods and Security Protocols (FMSP'99)*, Trento, Italy, 1999. ACM New York.

[73] N. Durgin, P. Lincoln, and J. Mitchell. Multiset Rewriting and the Complexity of Bounded Security Protocols. *Journal of Computer Security*, 12(2):247–311, 2004.

[74] N. Durgin, P. Lincoln, J. Mitchell, and A. Scedrov. Undecidability of Bounded Security Protocols. In E. Clarke and N. Heintze, editors, *In Proceedings of Workshop on Formal Methods and Security Protocols (FMSP'99)*, Trento, Italy, 1999. ACM New York.

[75] N. Durgin, P. Lincoln, J. Mitchell, A. Scedrov, and I. Cervesato. Relating Strands and Multiset Rewriting for Security Protocol Analysis. In P. Syverson, editor, *Proceedings of the 13th IEEE workshop on Computer Security Foundations (CSFW '00)*, http://computer.org/proceedings/csfw/0671/0671toc.htm, pages 35–51, Cambridge, UK, 2000. IEEE Computer Society. Online Publication.

[76] H. Ehrig and B. Mahr. *Fundamentals of Algebraic Specification 1: Equations and Initial Semantics*. Springer-Verlag New York, 1985.

[77] H. Ehrig and B. Mahr. *Fundamentals of Algebraic Specification 2: Module Specifications and Constraints*. Springer-Verlag New York, 1990.

[78] J. van Eijck and S. Orzan. Epistemic Verification of Anonymity. *Electronic Notes Theoretic Computer Science*, 168:159–174, 2007.

[79] T. Fábrega, J. Herzog, and J. Guttman. Strand Spaces: Proving Security Protocols Correct. *Journal of Computer Security*, 7(1):191–230, 1999.

[80] R. Fagin, J. Halpern, Y. Moses, and M. Vardi. *Reasoning about Knowledge*. MIT Press, 1995.

[81] N. Ferguson, B. Schneier, and T. Kohno. *Cryptography Engineering: Design Principles and Practical Applications*. Wiley Publishing, 2010.

[82] International Organization for Standardization. Information technology- Security techniques - Entity authentication mechanisms; Part 3: Entity authentication mechanisms using a public key algorithm (ISO/ IEC 9798-3), August 1993.

[83] International Organization for Standardization. Information technology- Security techniques - Entity authentication mechanisms; Part 2: Mechanisms using symmetric encipherment algorithms (HSO/IEC 9798-2), Second Edition, December 1994.

[84] International Organization for Standardization. Information technology- Security techniques - Entity authentication mechanisms; Part 3: Entity authentication mechanisms using a public key algorithm (ISO/IEC JTC1/SC27/WG2 M51), March 1991.

[85] International Organization for Standardization. Information technology - Security techniques - Entity authentication mechanisms; Part 1: General model (ISO/IEC 9798-1), Second Edition, September 1991.

[86] N. Friedman and J. Halpern. Modeling belief in dynamic systems part ii: revision and update. *Journal of Artificial Intelligence Res.*, 10:117–167, March 1999.

[87] A. Fujioka, T. Okamoto, and K. Ohta. A Practical Secret Voting Scheme for Large Scale Elections. In J. Seberry and Y. Zheng, editors, *Proceedings of Advances in Cryptology, Workshop*

on the Theory and Application of Cryptographic Techniques (AUSCRYPT'92), volume 718 of *Lecture Notes in Computer Science*, pages 244–251, Queensland, Australia, 1992. Springer New York.

[88] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional, 1995.

[89] P. Gammie and R. van der Meyden. MCK: Model Checking the Logic of Knowledge. In R. Alur and D. Peled, editors, *Proceedings of 16th International Conference on Computer Aided Verification (CAV'04)*, volume 3114 of *Lecture Notes in Computer Science*, pages 479–483, Boston,US, 2004. Springer Heidelberg/Berlin.

[90] F. Garcia, I. Hasuo, W. Pieters, and P. van Rossum. Provable Anonymity. In R. Küsters and J. Mitchell, editors, *Proceedings of the 2005 ACM workshop on Formal methods in security engineering (FMSE'05)*, pages 63–72, Alexandria, US, 2005.

[91] G.Bella, S.Bistarelli, and F.Massacci. Retaliation Against Protocol Attacks. *Journal of Information Assurance and Security*, 3:89–102, 2008.

[92] D. Goldschlag, M. Reed, and P. Syverson. Onion Routing. *Communications of the ACM*, 42(2):39–41, 1999.

[93] D. Gollmann. What do we mean by entity authentication? In C. Landwehr, editor, *Proceedings of the 1996 IEEE Symposium on Security and Privacy (IEEE-S&P'96)*, pages 46–54, Washington, US, 1996. IEEE Computer Society.

[94] L. Gong. Using One-Way Functions for Authentication. *SIGCOMM Computer Communication Review*, 19(5):8–11, 1989.

[95] L. Gong, R. Needham, and R. Yahalom. Reasoning About Belief in Cryptographic Protocols. In D. Cooper and T. Lunt, editors, *Proceedings of the 1990 IEEE Symposium on Research in Security and Privacy (IEEE-S&P'90)*, pages 234–248, Oakland, US, 1990. IEEE Computer Society.

[96] J. Halpern and Ron van der Meyden. A Logical Reconstruction of SPKI. *Journal of Computer Security*, 11(4):581–613, 2004.

[97] J. Halpern and K. O'Neill. Anonymity and Information Hiding in Multiagent Systems. *Journal Computer Security*, 13(3):483–514, 2005.

[98] J. Halpern and R. Pucella. On the Relationship between Strand Spaces and Multi-Agent Systems. In P. Samarati, editor, *Proceedings of the 8th ACM conference on Computer and Communications Security (CCS'01)*, pages 106–115, Pennsylvania, USA, 2001. ACM New York.

[99] J. Halpern and R. Pucella. Modeling Adversaries in a Logic for Security Protocol Analysis. In *Proceedings of the Workshop on Formal Aspects of Security (FASec'02)*, volume 2629 of *Lecture Notes in Computer Science*, pages 115–132, London, UK, 2002. Springer Berlin/Heidelberg.

[100] J. Halpern and R. Pucella. Evidence with uncertain likelihoods. *Synthese*, 171(1):111–133, 2009.

[101] C. Hamblin. One-Valued Logic. *The Philosophical Quarterly*, 17(66):38–45, 1967.

[102] J. Heather, G. Lowe, and S. Schneider. How to Prevent Type Flaw Attacks on Security Protocols. *Journal of Computer Security*, 11(2):217–244, 2003.

[103] W. van der Hoek and J. Meyer. A Complete Epistemic Logic for Multiple Agents: Combining Distributed and Common Knowledge. In M. Bacharach, L. Girard-Varet, P. Mongin, and H. Shin, editors, *Epistemic Logic and the Theory of Games and Decisions*, pages 35–68. Kluwer, Dordrecht, 1997.

[104] G. Hughes and M. Cresswell. *A New Introduction to Modal Logics*. Routledge, 1998.

[105] M. Huth and M. Ryan. *Logic in Computer Science. Modelling and Reasoning about Systems.* Cambridge University Press, 1995.

[106] T. Hwang, N. Lee, C. Li, M. Ko, and Y. Chen. Two Attacks on Neuman-Stubblebine Authentication Protocols. *Information Processing Letters*, 53(2):103–107, 1995.

[107] F. Jacquemard, M. Rusinowitch, and L. Vigneron. Compiling and Verifying Security Protocols. In M. Parigot and A. Voronkov, editors, *Proceedings of the 7th International Conference Logic for Programming, Artificial Intelligence, and Reasoning (LPAR'00)*, volume 1995 of *Lecture Notes of Computer Science*, pages 131–160, Reunion Island, France, 2000. Springer Berlin/Heidelberg.

[108] A. V. Jones and A. Lomuscio. Distributed BDD-based BMC for the Verification of Multi-Agent Systems. In W.van der Hoek, G. Kaminka, Y. Lespérance, M. Luck, and S. Sen, editors, *Proceedings of the 9th International Conference on Autonomous Agents and Multi-Agent systems (AAMAS'10)*, pages 675–682, Toronto, Canada, 2010. IFAAMAS Press.

[109] M. Kacprzak, A. Lomuscio, A. Niewiadomski, W. Penczek, F. Raimondi, and M. Szreter. Comparing BDD and SAT Based Techniques for Model Checking Chaum's Dining Cryptographers Protocol. *Fundamenta Informaticae*, 72(1-3):215–234, 2006.

[110] M. Kacprzak, W. Nabialek, A. Niewiadomski, W. Penczek, A. Pólrola, M. Szreter, B. Woźna, and A. Zbrzezny. VerICS 2007 - a Model Checker for Knowledge and Real-Time. *Fundamenta Informaticae*, 85(1-4):313–328, 2008.

[111] A. Kehne, J. Schönwälder, and H. Langendörfer. A Nonce-Based Protocol for Multiple Authentications. *SIGOPS Operating Systems Review*, 26(4):84–89, 1992.

[112] R. Kemmerer. Using Formal Verification Techniques to Analyze Encryption Protocols. In *Proceeding of the 1987 IEEE Symposium on Security and Privacy (IEEE-S&P'87)*, pages 134–139, California, US, 1987. IEEE Computer Society.

[113] D. Knuth. *The Art of Computer Programming, Volume I: Fundamental Algorithms*. Addison-Wesley, 1968.

[114] S. Kripke. Semantic Analysis of Modal Logic (Abstract). *Journal of Symbolic Logic*, 24:323–324, 1959.

[115] O. Kupferman, M. Y. Vardi, and P. Wolper. An Automata-Theoretic Approach to Branching-Time Model Checking. *Journal of the ACM*, 47(2):312–360, 2000.

[116] M. Kurkowski, W. Penczek, and A. Zbrzezny. SAT-based Verification of Security Protocols via Translation to Networks of Automata. In *Proceedings of the 4th International Workshop on Model checking and Artificial Intelligence (Mochart'07)*, Lecture Notes In Artificial Intelligence, pages 146–165, Riva del Garda, Italy, 2007. Springer Berlin/Heidelberg.

[117] R. Kusters and T. Wilke. Automata-Based Analysis of Recursive Cryptographic Protocols. In V. Diekert and M. Habib, editors, *Proceedings of the 21st Annual Symposium on Theoretical Aspects of Computer Science (STACS'04)*, volume 2996 of *Lecture Notes in Computer Science*, pages 382–393, Montpellier, France, 2004. Springer Berlin/Heidelberg.

[118] M. Kwiatkowska, G. Norman, and D. Parker. A Framework for Verification of Software with Time and Probabilities. In K. Chatterjee and T. Henzinger, editors, *Formal Modeling and Analysis of Timed Systems*, volume 6246 of *Lecture Notes in Computer Science*, pages 25–45. Springer Berlin/Heidelberg, 2010.

[119] Laboratoire Spécification et Vérification (LSV), École Normale Supérieure Cachan. SPORE: Security Protocols Open Repository. http://www.lsv.ens-cachan.fr/Software/spore/. A Library of Cryptographic Protocols Descriptions.

[120] F. Laroussinie. *Logiques temporelles avec passé pour la spécification et la vérification des systèmes réactifs.* PhD thesis, Institut National Polytechnique de Grenoble, France, 1994.

[121] B. Lee, C. Boyd, E. Dawson, K. Kim, J. Yang, and S. Yoo. Providing receipt-freeness in mixnet-based voting protocols. In *Proceedings of Information Security and Cryptology (ICISC03).*

[122] C. Lewis and C. Langford. *Symbolic Logics.* The Century Co., New York, 1932.

[123] D. Lewis. Counterpart Theory and Quantified Modal Logic. In M. Loux, editor, *The Possible and the Actual*, pages 110–128. Cornell University Press, New York, US, 1979.

[124] J. Loeckx, H. Ehrich, and M. Wolf. Algebraic Specification of Abstract Data Types. In S. Abramsky, D. Gabbay S. Thomas, and E. Maibaum, editors, *Handbook of Logic in Computer Science*, volume 5, pages 217–316. Clarendon Press, 2000.

[125] A. Lomuscio and W. Penczek. LDYIS: a Framework for Model Checking Security Protocols. *Fundamenta Informaticae*, 85:359–375, 2008.

[126] A. Lomuscio, H. Qu, and F. Raimondi. MCMAS 1.0. http://www-lai.doc.ic.ac.uk/mcmas/, 2009.

[127] A. Lomuscio, H. Qu, and F. Raimondi. MCMAS: A Model Checker for Multi-Agent Systems. In *Proceedings of the 21st International Conference on Computer Aided Verification (CAV'09)*, volume 5643 of *Lecture Notes in Computer Science*, pages 682–688. Springer, 2009.

[128] A. Lomuscio, H. Qu, M. Sergot, and M. Solanki. Verifying Temporal and Epistemic Properties of Web service Compositions. In A. Bouguettaya, I. Krüger, and T. Margaria, editors, *In Proceedings of the 5th International Conference on Service Oriented Computing (ICSOC'08)*, volume 5364 of *Lecture Notes in Computer Science*, Sydney, Australia, 2008. Springer Berlin/Heidelberg.

[129] A. Lomuscio and F. Raimondi. The Complexity of Model Checking Concurrent Programs against CTLK Specifications. In P. Stone and G. Weiss, editors, *Proceedings of the 5th International Joint Conference of Autonomous agents and multiagent systems (AAMAS'06)*, pages 548–550, Hakodake, Japan, 2006. ACM Press.

[130] A. Lomuscio, F. Raimondi, and B. Wozna. Verification of the Tesla protocol in MCMAS-X. *Fundamenta Informaticae*, 79(3-4):473–486, 2007.

[131] A. Lomuscio and M. Sergot. Deontic Interpreted Systems. *Studia Logica*, 75(1):63–92, 2003.

[132] A. Lomuscio and M. Sergot. A Formalisation of Violation, Error Recovery, and Enforcement in the Bit Transmission Problem. *Journal of Applied Logic*, 2(1):93–116, 2004.

[133] A. Lomuscio and M. Solanki. Mapping OWL-S Processes to Multi Agent Systems: A Verification Oriented Approach. In I. Awan, editor, *Proceedings of the 2009 International Conference on Advanced Information Networking and Applications Workshops (WAINA'09)*, pages 488–493, Bradford, UK, 2009. IEEE New York.

[134] A. Lomuscio and B. Woźna. A Combination of Explicit and Deductive Knowledge with Branching Time: Completeness and Decidability Results. In M. Baldoni, U. Endriss, A. Omicini, and P. Torroni, editors, *Proceeding of the 3rd International Workshop on Declarative Agent Languages and Technologies (DALT'05)*, volume 3904 of *Lecture Notes of Artificial Intelligence*, pages 188–204, Utrecht, The Netherlands, 2005. Springer Berlin/Heidelberg.

[135] G. Lowe. An Attack on the Needham-Schroeder Public-Key Authentication Protocol. *Information Processing Letters*, 56(3):131–133, 1995.

[136] G. Lowe. Some New Attacks Upon Security Protocols. In M. Merritt, editor, *Proceedings of the 9th IEEE workshop on Computer Security Foundations (CSFW'96)*, pages 162–169, County Kerry, Ireland, 1996. IEEE Computer Society.

[137] G. Lowe. A Hierarchy of Authentication Specifications. In S. Foley and J. Millen, editors, *In Proceedings of the 10th IEEE workshop on Computer Security Foundations (CSFW'97)*, pages 31–43, Massachusetts, US, 1997. IEEE Computer Society.

[138] G. Lowe. CASPER: A Compiler for the Analysis of Security Protocols. *Journal of Computer Security*, 6(1-2):53–84, 1998.

[139] G. Lowe and A. Roscoe. Using CSP to Detect Errors in the TMN Protocol. *Software Engineering*, 23(10):659–669, 1997.

[140] T. Lunt. A Survey of Intrusion Detection Techniques. *Computers and Security*, 12(4):405–418, 1993.

[141] Z. Manna and A. Pnueli. The Anchored Version of the Temporal Framework. In J. de Bakker, W. de Roever, and G. Rozenberg, editors, *Linear Time, Branching Time and Partial Order*

in Logics and Models for Concurrency, volume 354 of *Lecture Notes in Computer Science*, pages 201–284. Springer Berlin/Heidelberg, 1989.

[142] K. Mano, Y. Kawabe, H. Sakurada, and Y. Tsukada. Role Interchange for Anonymity and Privacy of Voting. *Journal of Logics and Computation*, exq013:1–38, 2010.

[143] K. McMillan. *Symbolic Model Checking.* Kluwer Academic Publishers, 1993.

[144] C. Meadows. The NRL Protocol Analyzer: An Overview. *Journal of Logic Programming*, 26:113–131, 1996.

[145] A. Menezes, P. van Oorschot, and S. Vanstone. *Handbook of Applied Cryptography.* CRC Press, 1996.

[146] R. van der Meyden. Constructing Finite State Implementations of Knowledge-Based Programs with Perfect Recall. In N. Foo and R. Goebel, editors, *Proceedings from the Workshop on Intelligent Agent Systems, Theoretical and Practical Issues (PRICAI'96)*, volume 1209 of *Lecture Notes In Computer Science*, pages 135–151, Cairns, Australia, 1996. Springer-Verlag UK.

[147] R. van der Meyden and K. Su. Symbolic model checking the knowledge of the dining cryptographers. In G. Dinolt and R. Focardi, editors, *Proceedings of the 17th IEEE Computer Security Foundation Workshop (CSFW'04)*, pages 280–291, Washington, US, 2004. IEEE Computer Society.

[148] J. Meyer and W. van der Hoek. *Epistemic Logic for Computer Science and Artificial Intelligence.* Cambridge University Press, New York, US, 1995.

[149] J. Millen. The Interrogator: A Tool for Cryptographic Protocol Security. In *Proceedings of the 1984 IEEE Symposium on Security and Privacy (IEEE-S&P'84)*, pages 134–141, Oakland, US, 1984. IEEE Computer Society.

[150] J. Millen. A CAPSL Connector to Athena. In H. Veith, N. Heintze, and E. Clarke, editors, *Proceedings of the Workshop on Formal Methods and Computer Security (FMCS'00)*, Chicago, US, 2000.

[151] J. Millen. Common Authentification Protocol Specification Language. http://www.csl.sri.com/projects/capsl/, 2001.

[152] J. Misra. The Muddy Children Puzzle. http://www.cs.utexas.edu/users/misra/Notes.dir/MuddyChildren.pdf, 1998.

[153] J. Mitchell, M. Mitchell, and U. Stern. Automated Analysis of Cryptographic Protocols using Mur-phi. In C. Landwehr, editor, *Procedings of the 1997 IEEE Symposium on Security and Privacy (IEEE-S&P'97)*, pages 141–151, Oakland, US, 1997. IEEE Computer Society.

[154] Y. Moses, D. Dolev, and J. Halpern. Cheating Husbands and Other Stories. A Case Study of Knowledge, Action, and Communication. Unpublished Manuscript.

[155] R. Needham and M. Schroeder. Using Encryption for Authentication in Large Networks of Computers. *Communications of the ACM*, 21:993–999, 1978.

[156] D. Nessett. A Critique of the Burrows, Abadi and Needham Logic. *SIGOPS Operating Systems Review*, 24(2):35–38, 1990.

[157] D. von Oheimb. The High-Level Protocol Specification Language HLPSL developed in the EU project AVISPA. In M. Hofmann and H. Loidl, editors, *Proceedings of International Summer School On Applied Semantics (APPSEM'05)*, Frauenchiemsee, Germany, 2005. Springer-Verlag New York.

[158] T. Okamoto. An Electronic Voting Scheme. In N. Terashima and E. Altman, editors, *Proceedings of Advanced IT Tools, IFIP World Conference on IT Tools*, pages 21–30, Canberra, Australia, 1996. ACM New York.

[159] T. Okamoto. Receipt-Free Electronic Voting Schemes for Large Scale Elections. In B. Christianson, B. Crispo, M. Lomas, and M. Roe, editors, *Security Protocols*, volume 1361 of *Lecture Notes in Computer Science*, pages 25–35. Springer Berlin/Heidelberg, 1998.

[160] R. Parikh. Knowledge and the Problem of Logical Omniscience. In Z. Ras and M. Zemankova, editors, *Proceedings of the International Syposium on Methodologies for Intelligent Systems (ISMIS)*, pages 432–439, North-Carolina, US, 1987. North-Holland/Elsevie.

[161] R. Parikh and R. Ramanujam. Distributed Processes and the Logic of Knowledge. In R. Parikh, editor, *Proceedings of Logic of Programs*, volume 193 of *Lecture Notes in Computer Science*, pages 256–268, New York, US, 1985. Springer.

[162] L. Paulson. The Inductive Approach to Verifying Cryptographic Protocols. *Journal of Computer Security*, 6:85–128, 1998.

[163] L. Paulson. Proving Security Protocols Correct. In G. Longo, editor, *Proceeding of the 14th Annual Symposium on Logic in Computer Science (LICS'99)*, pages 370–383, Trento, Italy, 1999. IEEE Computer Society Press.

[164] S. Petride and R. Pucella. Perfect Cryptography, S5 Knowledge, and Algorithmic Knowledge. In D. Samet, editor, *Proceedings of the 11th conference on Theoretical Aspects of Rationality and Knowledge (TARK'07)*, pages 239–247, Brussels, Belgium, 2007. ACM New York.

[165] A. Pnueli. The Temporal Logic of Programs. In J. Carlyle, editor, *Proceedings of the 18th Annual Symposium on Foundations of Computer Science (SFCS '77)*, pages 46–57, Washington, USA, 1977. IEEE Computer Society.

[166] AVISPA Project. AVISS – Deliverable D.6.1: List of Selected Problems. http://www.avispa-project.org/delivs/6.1/d6-1.ps, 2005.

[167] AVISPA Project. IF- Intermediate Format. http://www.avispa-project.org/delivs/2.3/d2-3.pdf, 2005.

[168] R. Pucella and V. Weissman. A Logic for Reasoning about Digital Rights. In I. Cervesato and S. Schneider, editors, *Proceedings of the 15th Computer Security Foundations Workshop (CSFW'02)*, pages 282–294, Nova Scotia, Canada, 2002. IEEE Computer Society.

[169] F. Raimondi. *Model Checking Multi-Agent Systems*. PhD thesis, University of London, 2006.

[170] F. Raimondi and A. Lomuscio. MCMAS-X - An Extension of MCMAS to Explicit Knowledge. http://www.cs.ucl.ac.uk/staff/f.raimondi/mcmas-x.tar.gz.

[171] R. Ramanujam and S. Suresh. A Decidable Subclass of Unbounded Security Protocols. In R. Gorrieri and R. Lucchi, editors, *Proceedings of IFIP WG 1.7 and ACM SIGPLAN Workshop on Issues in the Theory of Security (WITS'03)*, pages 11–20, Warsaw, Poland, 2003. ACM New York.

[172] R. Ramanujam and S. Suresh. Undecidability of Secrecy for Security Protocols. *Manuscript*, 2003.

[173] R. Rivest, A. Shamir, and L. Adleman. A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. *Communications of the ACM*, 21:120–126, 1978.

[174] A. Roscoe. Modelling and Verifying Key-exchange Protocols using CSP and FDR. In C. Landwehr, editor, *Proceedings of the 1995 IEEE Computer Security Foundations Workshop (IEEE-S&P'95)*, pages 98–107, Oakland, US, 1995. IEEE Computer Society Press.

[175] M. Rusinowitch and M. Turuani. Protocol Insecurity with Finite Number of Sessions is NP-complete. In P. Pandya and J. Radhakrishnan, editors, *Proceedings of the 14th IEEE Workshop on Computer Security Foundations (CSFW'01)*, pages 174–187, Nova Scotia, Canada, 2001. IEEE Computer Society.

[176] K. Schneider. *Verification of Reactive Systems: Formal Methods and Algorithms*. Springer-Verlag, 2004.

[177] S. Schneider. Verifying Authentication Protocols in CSP. *IEEE Transactions of Software Engineering*, 24(9), 1998.

[178] S. Schneider, B. Roscoe, M. Goldsmith, P. Ryan, and G. Lowe. *Modelling and Analysis of Security Protocols*. Addison-Wesley, UK, 2001.

[179] F. Somenzi. CUDD: CU Decision Diagram Package - Release 2.3.1. http://vlsi.colorado.edu/ /~fabio/{CUDD}/cuddIntro.html, 2004.

[180] D. Song, S. Berezin, and A. Perrig. Athena: A Novel Approach to Efficient Automatic Security Protocol Analysis. *Journal of Computer Security*, 9(1-2):47–74, 2001.

[181] M. Stehr, I. Cervesato, and S. Reich. Multiset Rewriting: MSR1, MSR2- Protocol Specification Languages. http://formal.cs.uiuc.edu/stehr/msr_eng.html.

[182] P. Syverson and I. Cervesato. The Logic of Authentication Protocols. In R. Focardi and R. Gorrieri, editors, *Proceedings of the International School on Foundations of Security Analysis and Design (FOSAD'00)*, volume 2170 of *Lecture Notes in Computer Science*, pages 63–136. Springer, 2000.

[183] P. Syverson, C. Meadows, and I. Cervesato. Dolev-Yao is no better than Machiavelli. In P. Degano, editor, *Proceedings of the 1st IFIP WG 1.7 and ACM SIGPLAN Workshop on Issues in the Theory of Security (WITS'00)*, pages 87–92, Geneva, Switzerland, 2000. ACM New York.

[184] P. Syverson and P. van Oorschot. A Unified Cryptographic Protocol Logic. *Naval Research Laboratory Publication*, 5540(227), 1996.

[185] P. Syverson and S. Stubblebine. Group Principals and the Formalization of Anonymity. In J. Wing, J. Woodcock, and J. Davies, editors, *Proceedings of the World Congress on Formal Methods (FM'99) in the Development of Computing Systems*, volume 1 of *Lecture Notes for Computer Science*, pages 814–833, Toulouse, France, 1999. Springer Berlin/Heidelberg.

[186] The European Network and Information Security Agency (ENISA). Study on the Costs of DNSSEC Deployment. http://www.enisa.europa.eu/act/res/technologies/tech/dnsseccosts, 2009.

[187] F. Ţiplea, C. Bîrjoveanu, and C. Enea. Decidability and Complexity Results for Security Protocols. In E. Clarke, M. Minea, and F. Tiplea, editors, *Proceedings of NATO Workshop on Verification of Infinite-state Systems with Applications to Security (VISSAS 2005)*, Electronic Notes in Theoretical Computer Science, Timisoara, Romania, 2005. Elsevier.

[188] F. Ţiplea, C. Enea, C. Bîrjoveanu, and I. Boureanu. Security Protocols With Freshness Check is NEXPTIME-complete. *Journal of Computer Security*, 16(6):689–712, 2008.

[189] Y. Tsukada, K. Mano, H. Sakurada, and Y. Kawabe. Anonymity, Privacy, Onymity, and Identity: A Modal Logic Approach. In R. Bilof, editor, *Proceedings of the 12th IEEE International Conference on Computational Science and Engineering (CSE'09)*, volume 3, pages 42–51, Vancouver, Canada, 2009. IEEE Computer Society.

[190] M. Turuani. The CL-AtSe Protocol Analyser. In F. Pfenning, editor, *Proceedings of the 17th International Conference on Rewriting Techniques and Applications (RTA'06)*, volume 4098 of *Lecture Notes in Computer Science*, pages 277–286, Washington, US, 2006. Springer-Verlag.

[191] Paul C. van Oorschot. Extending Cryptographic Logics of Belief to Key Agreement Protocols. In E. Bertino, editor, *Proceedings of the 1st of the ACM Conference on Computer and Communications Security (CCS'93)*, pages 232–243, Virginia, US, 1993. ACM New York.

[192] S. Vaudenay. *A Classical Introduction to Cryptography.* Springer, 2006.

[193] G. Wedel and V. Kessler. Formal Semantics for Authentication Logics. In E. Bertino, H. Kurth, G. Martella, and E. Montolivo, editors, *Proceedings of the 4th European Symposium on Research in Computer Security (ESORICS'96)*, volume 1146 of *Lecture Notes in Computer Science*, pages 219–241, Rome,Italy, 1996. Springer-Verlag.

[194] T. Woo and S. Lam. A Lesson on Authentication Protocol Design. *SIGOPS Operating Systems Review*, 28(3):24–37, 1994.

Appendix A

Details on Proofs and Implementations

In this appendix we provide details of some lemmas and proofs in Chapter 3. We also detail certain aspects of the implementation of the `PD2IS` tool presented in Chapter 5.

A.1 Additional Proofs for Chapter 3

In this section we give various proofs and details of statements made in Chapter 3.

I. Proof of Lemma 3.3.17 in Chapter 3

We reiterate the Lemma 3.3.17 in Chapter 3:

Let $A \in Ho$, $\sigma(A\text{-}role) \in \Sigma_{Pr}$ and $ag_A \in Ag$ correspond to $\sigma(A\text{-}role)$. Let $r \in Rules^A$, $r = i.A \to B : t$ (for some $B \in Ho$), $l \in L_{ag_A}$ with $l = \langle i, view \rangle$ and $view_0$ be the initial view of ag_A. Then, there exists $v \in \mathcal{R}_t^{ag_A}$ such that $construct^{\mathbb{I}^\sigma}(t, view, view_0) = (t, v)$.

Proof Assume by reductio ad absurdum that $construct^{\mathbb{I}^\sigma}(t, view, view_0) = (t, \perp)$. Then, by the denotation of symbol *construct* (given in Section 3.3.1, page 79), it follows that:

1. there exists $t_0 \in Sub(t) \cap \mathcal{T}_0$ such that $(t_0, \perp) \in view$

   or

2. $\sigma[v/t_0]$, but $consistent^{\mathbb{I}^{[v/t_0]}}(t, view_0) = false$.

Let us consider first the case 1. Since $(t_0, \perp) \in \mathit{view}$ and, in the hypothesis, $r = i.A \rightarrow B : t$, then it follows that $t_0 \notin \mathit{OwnedAtoms}^A$ and, further that $t_0 \in i\text{-}\mathit{LearnedAtoms}^A$. But since $l = (i, \mathit{view})$, by the local evolution function E_{ag_A} it follows that there exist $j \in \mathit{Steps}^A$, $j < i$, $l' = (j, \mathit{view}') \in L_{ag_A}$ where $\mathit{set}^{\mathbb{I}}(\mathit{view}', t')$ assigns $t_0 \in \mathit{Sub}(t') \cap \mathcal{T}_0$ to some value $v' \neq \perp$. If this is not the case then, by the local evolution function E_{ag_A}, the step j would not have been incremented. Hence, ag_A would not be in step i. (1)

Let us consider first the case 2. If $\mathit{consistent}^{\mathbb{I}^\sigma}(t, \mathit{view}_0) = \mathit{false}$, it means that there exists $t', t'' \in \mathcal{N} \cup \mathcal{K}_0$ such that $\sigma(t') = \sigma(t'')$. Since *consistent* refers itself only to view_0, then it follows $t', t'' \in \mathit{OwnedAtoms}^A$. But by the initial setup of ag_A (see Remark 3.1.9) these would be assigned to different values. So, this case is refuted. (2)

So, by (1) and (2) above the lemma follows. ∎

II. On the Dolev-Yao Analysis and Synthesis in M_{IS}

Let us first give once more the procedural expression[1] of the analysis cycles 2–5 in the local evolution function of the Environment agent. In comparison to the original version in See Chapter 3, page 103, we add some explanatory comments to the procedure.

```
//initialisation of the model
stop_analz:=false;
for(i:=0;i<#maxSubstComposites;i++)
          flag_analz[i]:=false;
          already_analz[i]:=false;
endfor

//Dolev-Yao analysis cycles in the model
label l1:
while(!stop_analz)
  count1:=0;
  foreach [x'/t']
     // [x'/t'] is in the possesion of the intruder
     if([x'/t'] in analz\_log)
          flag_analz[count_1]:=analz(values_log, analz_log,x');
          count_1++;
```

[1]See Chapter 3, page 103.

```
     // [x'/t'] is NOT in the possesion of the intruder
     else if([x'/t'] not in analz\_log)
          flag_analz[count_1]:=false;
          count_1++;
     endif
  endfor
  noNewAnalz_counter:=0;
  allAnalz_counter:=0;
  for(i:=0;i<#maxSubstComposites;i++)

        // composite i has not been succesfully analysed
        if (flag_analz[i]==false)
          noNewAnalz_counter++;

         // composite i has been succesfully analysed and for the first time
        else if (flag_analz[i]==true and already_analz[i]==false)
          already_analz[i]:=true;
          // a new analysis cycle is triggered
          continue l1;

        // composite i has been succesfully analysed in previous analz cycles too
        else if(flag_analz[i]==true and already_analz[i]==true)
           //potential for all composite to have been analysed
          allAnalz_counter++;
        endif
  endfor

  // no new composite was analysed
  if(noNewAnalz_counter==#maxSubstComposites)
      stop_analz:=true;

  // all composites can be analysed succesfully
  if(allAnalz_counter==#maxSubstComposites)
      stop_analz:=true;
  endif
endwhile
```

Lemma A.1.1 *The analysis actions triggered at a local state $l \in L_{Env}$ in the evolution function E_{Env} of the Environment agent simulate the closure of $l \in L_{Env}$ under analysis with respect to the denotations given. I.e., a fixed-point of $(analz(\ldots analz(l_{Env}, \ldots)), \ldots), \ldots)$ is reached.*

Proof Let us suppose by reductio ad absurdum that a fixed point is not reached at local state l_{Env}, upon applying a series of *analz* actions.

Then, by the procedural expression above, it follows that neither the condition $noNewAnalz\_counter{=}{=}\#maxSubstComposites$ nor the condition $allAnalz\_counter{=}{=}\#maxSubstComposites$ is ever satisfied.

Let us consider the case of $noNewAnalz\_counter{=}{=}\#maxSubstComposites$ never being satisfied. $(*)$

From $(*)$ it follows that there is always a value for i such that $flag\_analz[i] == true$.

By the denotation $analz^{\mathbb{I}}$, it follows that there is always a composite t' whose value x' can be analysed at some reached *analz_log* and *values_log*. But for this to be true, it means that at the previous *while* cycle the analysis of some composite $[x/t]$ enriched *values_log* with $[k/v]$ such that $last(t') = k$ and $last(x') = v'$. Then, by the procedure the index of the composite $[x/t]$ will be marked in *already_analz*.

As the above should hold infinitely often (hence, we are working under the hypothesis of $noNewAnalz\_counter{=}{=}\#maxSubstComposites$ never being satisfied), then either there are an infinite number of composite-values $[x'/t']$ as above or all composites are eventually marked in *already_analz*. The first case cannot occur, as the number of ranges for a composite and the number of composites are both finite. If the second situation is the case, then the condition $allAnalz\_counter{=}{=}\#maxSubstComposites$ is eventually satisfied. This contradicts our original reduction ad absurdum assumption.

Note the the *if* branches and the *for* loops involved are exhaustive with respect to the conditions they imply.

The other case (i.e., $allAnalz\_counter{=}{=}\#maxSubstComposites$ never being satisfied) is treated analogously.

Therefore, the *while* loop stops and either all $[x'/t']$ have been succesfully analysed at a state l_{Env} or no $[x'/t']$ can be further analysed at a state l_{Env}. ▮

From the above it follows that there always exists a natural constant $k \geq 1$ and there exists a natural number n such that $1 \leq n \leq k \times \#max\_SubstComposites$ such that *analz* actions are applied n times at some state l_{Env} until the closure of the Dolev-Yao analysis under this bounded model is reached.

A.2 Additional Details for Chapter 5

In this section we give various details on the implementation of the `PD2IS` toolkit, presented in Chapter 5

Generating the Generalised Cartesian Product. In Chapter 5 we detail the implementation of the `PD2IS` toolkit. It homomorphically instantiates a fully ground IS-based model Υ_{IS} starting from a symbolic `CAPSL` description of a protocol. In doing so, it needs to compute the corresponding ranges of messages and composites starting from the ranges of the comprising atomic terms. It also needs to express all possible, valid combinations in certain expressions within the evolutions and protocol functions of the IS-based model (e.g., expression $(*)$ in P_{ag_A}, as per Chapter 3, page 95).

Consider n sequences R_1, R_2, …, R_n, each corresponding to an atomic range ($n \geq 2$). The Cartesian product $R_1 \times \ldots \times R_n$ is the sequence of all possible n-tuples where the first element is from R_1, the second element is from R_2, etc. In the following, we will explain the implementation of how to obtain the aforementioned Cartesian product $R_1 \times \ldots \times R_n$. The actual coding of `PD2IS` was done using the `Java 5.0` programming language. The details presented hereby are not written in the pure `Java`, but in a more imperative-like syntax, easier to understand and to explain.

Let us first explain the intuition behind the procedure of generating the Cartesian product $R_1 \times \ldots \times R_n$. To begin with, consider the simple case of only two ranges, e.g., two sequences $R_1 = \{a, b\}$ and $R_2 = \{x, y, z\}$. Let an iterator go through the first range and a nested iterator go through the second range. If we concatenate the value of the first to the value of the second, we will obtain $\{(a, x), (a, y), (a, z), (b, x), (b, y), (b, z)\}$. Hence, we will obtain the ordered 2-tuples giving indeed the Cartesian product of $R_1 \times R_2$. The code below implements this intuition.

```
List R_1, R_2;
List result = new List ();
for first in R_1
```

```
    for second in R_2
       result=result.add (concat(first, second));
return result;
```

So, by the code above, in the list *result* will have the Cartesian product of $R_1 \times R_2$. Of course, the procedure can be generalised to R_1 and R_2 being sequences of any length. In fact, if we consider a third range R_3, the Cartesian product of $R_1 \times R_2 \times R_3$ can be obtained using the same procedure. To exemplify, let $R_3 = \{m, n\}$. From before, *result* is $\{(a, x), (a, y), (a, z), (b, x), (b, y), (b, z)\}$. Then, by applying the procedure above, $R_1 \times R_2 \times R_3$ equals $result \times R_3$; hence, it is $\{(a, x, m), (a, x, n), (a, y, m), (a, y, n), (a, z, m), (a, z, n), (b, x, m), (b, x, n), (b, y, m), (b, y, n), (b, z, m), (b, z, n)\}$. In more detail, we compute first $R_1 \times R_2$ into the variable *result*, then *result* is attributed to R_1 and R_3 is assigned to R_2 and we re-apply the previous procedure.

The following procedure shows this generalisation precisely, i.e., the way to compute the product $R_1 \times R_2 \times \ldots \times R_n$, for R_i a range of values, $i \in \{1, \ldots n\}$ and $n \geq 2$.

```
Iterable<Iterable<T>> genCartesianProduct;
Iterable<Iterable<T>> result = new List<Iterable<T>();
foreach(var tuple in genCartesianProduct) {
  var seq2 = tuple;
  for seq1 in result
       for item in seq2
          seq1 = seq1.concat(item);
          result = result. add(seq1);
}
return result;
```

The above is presented in a way which is intuitively close to the concept of lists in functional programming languages. Equally, it is close to a notion of *accumulator* (e.g., in the Microsoft .Net platform). We create the generalised Cartesian product in the following way: *1)* we begin with an empty accumulator; *2)* at each step, the new accumulator is obtained by adding the current item in the ultimate accumulator to the penultimate accumulator. At every step of the way, the accumulator will be the Cartesian product of all the sequences processed so far.

We now discuss some aspects related the above, in the context of our `PD2IS` toolkit. The first aspect is the need to use methods of producing the Cartesian product like above in many different contexts, under possibly distinct requirements. For instance, we need to calculate both the σ-restricted ranges for all A^σ-*agent* and the unrestricted ranges for the Environment. Also, we need to expand the ground expressions of the evolution functions and protocol functions differently within different instances, etc. The second aspect is the possible need to produce ordered pairs, i.e., the range of $\{Na, A\}$ is given by $\mathcal{R}_N \times \mathcal{R}_A$ and not by $\mathcal{R}_A \times \mathcal{R}_N$. The third aspect is the need to eliminate the symmetry from some of the ranges generated as above. The fourth aspect is the need to constrain these ranges to produce certain protocol-models (see Chapter 5).

For all of the above reasons, we actually implement a hierarchy of `GeneralisedPermutations` classes and interfaces. These classes facilitate also the indexing of variables, i.e., the range R_1 corresponds to the variable N_a, the range R_2 corresponds to the variable A, etc. Furthermore, indexing ranges with other than by variables is facilitated through these classes. All these classes implement the generic methodology presented above for the calculation of the Cartesian product to construct non-atomic ranges. However, each extend the behaviour to cope with one or several of the aspects aforementioned. The indexing is used as legend to enforce certain constraints, eliminate symmetry, etc. For instance, a constraint can be that all sequences where the value of index 1 is equal with the value at index 3 should be dismissed from the end result. A symmetry elimination can be formulated like: all two sequences where the first contains value x at index 1 and value y at index 3 and the second contains value y at index 1 and value x at index 3 should be eliminated from the end result. The classes are open-source, coded in `Java 5` and are available with `PD2IS` at [28].

Appendix B

An Extension to Chapter 4

In this appendix we provide details of some lemmas and proofs in Chapter 4. In relation to Chapter 4, we also present an additional proof (Theorem B.3.1) concerning the correctness of the Υ_{IS} formalisation with respect to a standard semantics for **CAPSL***-described protocols. Alternative proofs (Theorem B.3.2 and Theorem B.4.3) that M_{IS} and M_{CAPSL} are aligned in the validation/refutation of traditional* **CAPSL** *goals are also presented.*

B.1 Introductory Notions

In this section we present some background notions used in the lemmas and proofs in Chapter 4 and in this appendix. Mainly, we recall the concept of homomorphism [124] and the concept of stuttering equivalence [37].

We first recall the definition of a homomorphism [124]. We will recall the general notion of homomorphism between universal algebras (see Chapter 7 for a review on these algebraic notions). The definition of homomorphism on universal algebras can be transferred to other mathematical objects (e.g., groups, transition systems, etc). In the case of transition systems, the operations are given by the transition relations.

Definition B.1.1 (Homomorphism) *Let S be a set of sort and Σ be a S-sorted signature. Let*

$\mathbf{A} = (A, \Sigma^A), \mathbf{B} = (B, \Sigma^B)$ *be two* Σ*-algebras. Let an indexed function* $h : A \to B$*, i.e.,* $h = (h_s | s \in S)$*,* $h_s : A_s \to B_s$*, for any* $s \in S$*. The function* h *is a* homomorphism *between* $\mathbf{A}$ *and* $\mathbf{B}$ *if*

$(\forall(\omega, s) \in S^* \times S \,|\, \omega = (s_1, \ldots, s_n))(\forall\sigma \in \Sigma_{\omega,s})(\forall a_1 \in A_{s_1}) \ldots (\forall a_n \in A_{s_n})$*, it is the case that* $h_s(\sigma^A(a_1, \ldots, a_n)) = \sigma^B(h_{s_1}(a_1), \ldots, h_{s_n}(a_n))$.

We now recall the definition in [37] for the notion of stuttering equivalent.

Definition B.1.2 (Stuttering Equivalence) *Given two structures* M *and* M' *with the same set of atomic propositions, a sequence of equivalence relations* $E_0, E_1, \ldots$ *on* $S \times S'$ *is defined as follows:*

1. $s\, E_0\, s'$ *if and only if* $V(s) = V(s')$.

2. $s\, E_{n+i}\, s'$ *if and only if*

   A. *for every path* π *in* M *that starts in* s*, there is a path* π' *in* M' *such that it starts in* s'*, a partition* $B_1, B_2, \ldots$ *of* π *and a partition* $B'_1, B'_2, \ldots$ *of* π' *such that for all* $j = \overline{1, n}$*,* B_j *and* B'_j *are both non-empty and finite, and every state in* B_j *is* E_n*-related to every state in* B'_j*, and*

   B. *for every path* π' *in* M' *that starts in* s' *there is a path* π *in* M *starting in* s *and satisfies the same condition as in A.*

If such a sequence of relations is identified between the structures M *and* M'*, then the structures are called* stuttering equivalent*.*

B.2 Helpers for Certain Lemmas in Section 4.3

Informal Explanations on the proof of Lemma 4.3.8.

Recall Lemma 4.3.8:

Let Pr be an RT protocol, M^{Pr}_{CAPSL} be its bounded protocol model and Υ^{Pr}_{IS} be obtained from M^{Pr}_{CAPSL} via algorithm *tr*. Let M^{Pr}_{IS} be the unwinding of Υ^{Pr}_{IS}. For $\sigma(A\text{-}role) \in \Sigma_{Pr}$, $\sigma(A) \neq \mathbf{I}$, $t \in \mathcal{T}$, let $e = (\omega|_A, \sigma, i)$ with $act(e) = A!B : (M)t$ and let $s, s' \in S$ such that $s[e\rangle s'$. Let $\sigma^+(A\text{-}role) \mapsto_{\Upsilon_{IS}} ag_A$, $\sigma^+ \cong \sigma$, $g, g' \in G$ and $g\overline{a'}g'$ be the application of a joint action $\overline{a'} \in Act$ at g in M^{Pr}_{IS} such that $e\,\tilde{v}\,\overline{a'}$. Let $g'' \in G$ and $\beta_{Analz}{=}\overline{a}_1 \ldots \overline{a}_n$ be a g'-*analz* complete sequence of Dolev-Yao actions in M^{Pr}_{IS} with $g'\beta_{Analz}g''$. Then, $s'\,\tilde{p}\,g''$.

We are going to explain the ideas behind the proof of this lemma. To prove that $s^{'}\,\tilde{p}\,g^{''}$ we need to tackle the three parts in the definition of the $\tilde{p}$-relation.

Informally, the first part expresses that principal A can analyse at s' the same atomic terms as ag_A has at $g^{''}$ (up to the renaming of values, i.e., $\sigma^{+} \cong \sigma$). To begin with, states s and g were in the $\tilde{p}$-relation already, i.e., A could analyse at s the same atomic terms and ag_A had at g.
In M_{CAPSL}, the state s' has expanded with respect to state s only by the set M of atomic terms needed to generate t (i.e., $act(e) = A!B : (M)t$).
In M_{IS}, all terms in $Generated^{A}$ (hence, those in $M \subseteq Generated^{A}$ included) have been embedded into the state s_{0_A} under σ^{+} (i.e., the elements of M in Υ_{IS} are respectively equivalent up to renaming to the elements of M in M_{CAPSL}).
The state g'_{ag_A} does not change with respect to atomic terms in comparison to g, but it already contains $\sigma^{+}(M)$ by persistence of values in $Generated^{A}$, assigned in the initial setup of Υ_{IS}.
The state g''_{ag_A} is the same as g'_{ag_A}, as only the intruder is active during the sequence β_{Analz} of actions.
Therefore, principal A can indeed analyse at s' the same atomic terms as ag_A has at $g^{'}$ or $g^{''}$ (up to the renaming of some values). So, part (1) of state s being in the $\tilde{p}$-relation with $g^{''}$ is ensured.

Informally, part (2) of s' being in the $\tilde{p}$-relation with g'' means that if the intruder is not able to analyse an atomic term t_0 at s' in M_{CAPSL}, nor is he able to do so at state g'' in M_{IS}.
In M_{IS}, at $g^{''}$ the closure of g' under $analz^{\mathbb{I}}$ occurs. Under the set Σ^{+} of substitutions, at g'' the intruder has analysed all possible atomic terms given the current stage of the execution.
In M_{CAPSL}, the closure under $analz$ is instantaneous at s'.
In M_{CAPSL}, at the state s' the intruder knows t, which he did not at s.
In M_{IS}, at state g' the intruder acquires t, which he did not at g.
By the fact that $s\,\tilde{p}\,g$, the intruder analysed at s what he analysed at g.
Hence, with respect to the Dolev-Yao analysis, s' and g' were the identical up to the renaming of values. Closure under Dolev-Yao analysis happened at both, in the respective models. Most importantly, (the closure under) analysis operations follows the same, traditional semantics [163] in both models.
By all of the above, it follows that if the intruder is not able to analyse an atomic term t_0 at s' in M_{CAPSL}, nor is he able to do so at state g'' in M_{IS}.

Proving part (3) of s' being in the $\tilde{p}$-relation with g'' follows an identical reasoning to the one behind part (2), presented above.

In Section 4.3 we formulate and prove all these formally.

Informal Explanations on the proof of Lemma 4.3.21.

Recall Lemma 4.3.21: *L*et Pr be an RT protocol and M^{Pr}_{CAPSL} be its bounded protocol model. Let Υ^{Pr}_{IS} be obtained from M^{Pr}_{CAPSL} via algorithm tr. Let M^{Pr}_{IS} be the unwinding of the Υ^{Pr}_{IS}. Let $\sigma(A\text{-}role) \in \Sigma$, $\sigma(A) \neq \mathbf{I}$, $s, s' \in S$, $e = (\omega|_A, \sigma, i)$ be a receive event in M^{Pr}_{CAPSL} such that $act(e) = A?B : (M)t$ and $s[e\rangle s^{'}$. Let $t \in Msg$, $x \in \mathcal{R}_t$, $\sigma^+ \in \Sigma^+$, $\sigma^+ \cong \sigma$, $\sigma^+(A\text{-}role) \mapsto_{\Upsilon_{IS}} ag_A$, $g, g' \in G$ and $g\overline{a}g^{'}$ be the application of a joint action $\overline{a}$ in M^{Pr}_{IS} such that $e\,\tilde{v}'\,\overline{a}$, where $\overline{a}_{ag_A} = receive$ and $\overline{a}_{Env} = transmit(ag_A, t, x)$. Then, $s^{'}\,\tilde{p}\,g^{'}$ and $s^{'}\,\tilde{p}'\,g^{'}$.

The informal explanations behind its proof are given in the following.

In M_{CAPSL}, the state s'_A "grows" with t in comparison with s (i.e., $act(e) = A?B : t$).
In Υ_{IS}, if $out\_match^{\mathbb{I}}$ returns $true$ at g, then g is updated to g' according to the denotation of $set^{\mathbb{I}}$. To begin with, s and g were in the $\tilde{p}$-relation. Because of that, it will easily follow that $out\_match^{\mathbb{I}}$ returns $true$ at g.
On the intruder's part of the states there is no update, in any of the models. So, the only concern comes from the part of the states allocated to principal A and agent ag_A, respectively. In that respect, agent A should have in M_{IS} at state g' the same atomic terms as the principal A would be able to analyse in M_{CAPSL} at the state s'.
Given the above, the only thing that could prevent this from holding would be if $set^{\mathbb{I}}$ applied at g had a different meaning that the simple $s' = s \cup \{t\}$ with respect to what atomic terms are to be analysable in s'_A or contained in g'_{ag_A}. However, as we apply M_{IS} and M_{CAPSL} only to the class $\mathcal{P}$ of receiver-transparent protocols, the latter is not the case (i.e., t would be such that principal A will be able to analyse all the atomic subterms in M_{CAPSL}, as the denotation of $set^{\mathbb{I}}$ dictates in Υ_{IS}).
As we already mentioned, there no updates to the intruder's share of the states, so the rest of the proof (i.e., regarding Dolev-Yao analysis and synthesis) follows trivially.
In Section 4.3 we formulate and prove all these formally.

B.3 Deciding Secrecy in M_{IS} Using Traditional Trace-Based Methods

In Section 4.2.1 we gave an algorithm called tr that produces Υ_{IS}^{Pr} from M_{CAPSL}^{Pr}. We recall that M_{IS}^{Pr} is the unwinding of the interpreted system Υ_{IS}^{Pr}. In Section 4.3 we proved that the events and actions in M_{CAPSL}^{Pr} and M_{IS}^{Pr} are such that they preserve "good" relations between the certain stages in the corresponding computations. This section comes to complete the series of lemmas in Section 4.3. In that sense, this section will show that the $\tilde{p}$-relation (see Definition 4.3.2) and the $\tilde{p}'$-relation are preserved all along the respective computations of the two models. We hereby show that the algorithm $A1$, used in [188] to decide initial secrecy in M_{CAPSL}, can at the same time decide secrecy in M_{IS}.

Such a proof is not required to prove the existence of a homomorphism between the two models. However, it could be used in the proof of homomorphism and it is a natural continuation of all the lemmas in Section 4.3. We therefore give it here, in this Appendix.

We recall the algorithm $A.1$ in [188] for deciding initial secrecy.

Algorithm $A.1$, deciding initial secrecy in M_{CAPSL}, [188], page 695.

```
let $E^{'}$ be the set of all $(T,k)$-*events*;
$\xi := \lambda$; $s := s_0$;
repeat
$E := E^{'}$;
$E^{'} := \emptyset$ ;//events that could not be applied
*bool* := 0;
while $E \neq \emptyset$ do
    begin
    choose $e \in E$;
    $E := E \setminus \{e\}$;
    if $(s,\xi)[e > (s^{'},\xi e)$ then
        begin
            $s := s^{'}$; $\xi := \xi e$; *bool* := 1;
        end
    else $E^{'} := E^{'} \cup \{e\}$;
end
until *bool* = 0;
if $(\bigcup_{Ai \in Ho} Secret_A) \cap analz(s_{\mathbb{I}}) \neq \emptyset$
then "leaky" else "non-leaky"
```

In the next theorem, we are going to show that any run ξ constructed by algorithm $A1$ in

M_{CAPSL} corresponds to a run α in M_{IS} such that all along the runs states are consistently in the $\tilde{p}$ and $\tilde{p}'$ relation. This will enable us to show that the $\asymp$-corresponded goals are satisfied/refuted in both models, without using any homomorphic properties of the models.

Theorem B.3.1 *Let Pr be an RT protocol, M^{Pr}_{CAPSL} be its bounded protocol model and Υ^{Pr}_{IS} be the output of algorithm tr given M^{Pr}_{CAPSL}. Let M^{Pr}_{IS} be the unwinding of the interpreted system Υ^{Pr}_{IS}. Let ξ be a (maximal) run in M^{Pr}_{CAPSL}, as reported by the algorithm $A.1$ in [188]. For each event e in ξ, associate the action(s) in M^{Pr}_{IS} that algorithm tr would produce. Let α be the resulting sequence of actions in M^{Pr}_{IS}. Then, α describes a computation in M^{Pr}_{IS}. Moreover, the resulting states along ξ and α are in the $\tilde{p}$-relation and in the $\tilde{p}'$-relation.*

Proof By structural induction on the run ξ in the hypothesis.

Base-case: Assume the initialisation in $A.1$: $s = s_0$, $\xi = \emptyset$.

By the construction in the algorithm tr, $\exists\ g_0 \in G_0$, $s_0\, \tilde{p}_0\, g_0 \overset{Remark\ 4.3.3}{\Longrightarrow} s_0\, \tilde{p}'\, g_0$.

The first event $e = (\omega|_A, \sigma, 1)$ chosen in the first cycle of the *while* loop in $A.1$ must be a send event with $act(e) = A!B : (M)t$ under some substitution σ, where $A, B \in Ho, t \in Msg, M \subseteq Generated^A$. Then tr dictates a joint action $\overline{a}$, under σ^+ for the *A-role*, such that: *1).* $e\, \tilde{u}\, \overline{a}$; *2).* $s_0[e\rangle s'$ in M_{CAPSL}, it is the case that $g_0 \overline{a} g'$ in M_{IS}.

In this context, $\alpha = \overline{a}$ is a computation in M_{IS}. So, $\xi = e$ is the current run in M_{CAPSL} as per $A.1$ and $\alpha = \overline{a}$ is the computation in M_{IS}.

Then, by tr, automatically in M_{IS} we have $g' \beta_{Analz} \gamma_{Forge} g''$.

Therefore, the current run in M_{CAPSL} is $\xi = e$ and the correspondent current computation in M_{IS} is $\alpha = \overline{a}\beta_{Analz}\gamma_{Forge}$

By the fact that $s_0\, \tilde{p}\, g_0$, $e\tilde{u}\overline{a}$, Lemma 4.3.16, Lemma 4.3.18 and Lemma 4.3.8, it follows that $s'\tilde{p}g''$ and $s'\tilde{p}'g''$. Therefore, the base-case is proven. (For simplicity, in the above we assumed $\sigma(A) \neq \mathbf{I}$; the case where $\sigma(A) = \mathbf{I}$ is handled similarly).

Inductive step: assume any $s \in S$, $g \in G$, s along ξ and g along α (i.e., $s_0[\xi\rangle s$ and $g_0 \alpha g$) such that $s\tilde{p}g$ and $s\tilde{p}'g$. $(*)$

Now, assume the algorithm $A.1$ chooses at the next *while* cycle an event e' under σ'.

Then, the algorithm *tr* dictates a joint action $\overline{a}^{'}$ under ${\sigma^{'}}^{+}$ for the underlying role. The event $e^{'}$ and the joint action $\overline{a}^{'}$ will be either in relation $\tilde{u}$ or in relation $\tilde{u}'$ (depending on whether they are send or receive events, respectively).

If it a send event, then *tr* would also enforce an complete intercept-analz-forge sequence. But, given $(*)$ and Lemma 4.3.16 and Lemma 4.3.18, Lemma 4.3.8, it follows that the resulting states will be in relations $\tilde{p}$ and $\tilde{p}'$.

If it is a receive event, then by Lemma 4.3.21 it follows that the resulting states will be in relations $\tilde{p}$ and $\tilde{p}'$.

Then, by lemmas above, the resulting states will always be in relations $\tilde{p}$ and $\tilde{p}'$. Therefore, the inductive step is proven too. ∎

From Theorem B.3.1 it follows that algorithm $A1$ from [188] (in tandem with algorithm *tr* in this thesis) can construct not only runs of M_{CAPSL}, but also runs of M_{IS}. Moreover, the states along these computations are proven to be systematically in the $\tilde{p}$ and $\tilde{p}'$ relation. Theorem B.3.2 will use that to show that $A1$ can do even more: if it decides secrecy in M_{CAPSL} on such runs, it decides it for M_{IS} as well. In other words, Theorem B.3.2 uses Theorem B.3.1 to prove preservation of satisfaction/refutation of $\asymp$-corresponded goals.

Theorem B.3.2 *Let Pr be an RT protocol, M_{CAPSL}^{Pr} be its bounded protocol model and Υ_{IS}^{Pr} be the output of algorithm tr. Let M_{IS}^{Pr} be the unwinding of the interpreted system Υ_{IS}^{Pr}. Let r be an atomic* `CAPSL` *goal of Pr such that $\rho_{CAPSL}(r) \asymp \rho_{IS}(r)$. If M_{CAPSL}^{Pr} validates $\rho_{CAPSL}(r)$, then M_{IS}^{Pr} validates f, for all $f \in \rho_{IS}(r)$. If M_{CAPSL}^{Pr} refutes $\rho_{CAPSL}(r)$, then there exists $f \in \rho_{IS}(r)$ such that M_{IS}^{Pr} refutes f.*

Proof Given the definition of relation $\asymp$ (i.e., Definition 4.2.5), let D be a `CAPSL` description for an RT protocol Pr, with the `CAPSL` goal $r =$ `SECRET t` and with a `CAPSL` assertion of type
`HOLDS: A t.`

For M_{CAPSL} to refute $\rho_{CAPSL}(r)$, then there is an arbitrary run ξ in M_{CAPSL} found leaky (by algorithm $A1$ [188]). Assume that $s_0[\xi\rangle s$, where $s_0, s \in S$. This means that $\sigma(t) \in analz(\sigma_1(s_{\mathbf{I}}))$, for some $\sigma(A\text{-}role) \in \Sigma$ and $\sigma_1 \in \Sigma|_{\mathbf{I}}$. **(I)**

But by Theorem B.3.1, there exists a computation α in M_{IS}, $g_0 \alpha g$ and $s \tilde{p} g$. By part (3) of the definition of $\tilde{p} \subseteq S \times G$ and (**I**), it is the case that $\Sigma^+(t) \in g_{Env}.values\_log$. So, there exists $\sigma^+(A\text{-}role) \in \Sigma^+$, $\sigma^+(A\text{-}role) \mapsto ag_A$ and $g_{Env}.POOL.t = ag_A.t \quad (*)$.

So, by $(*)$ and by the semantics of M_{IS}, there exist a path where eventually $g \in G$ with $g_{Env}.POOL.t = ag_A.t$ is reached. $(**)$

Now recall the expression $(S1)$ for $\rho_{IS}(r)$: $\bigwedge_{i \in \cup ag_A} AG(Environment.POOL.var \neq ag_A.var)$ (refer to Chapter 3, page 111, for details).

By $(**)$ and the semantics of M_{IS}, it follows that M_{IS} refutes $(S1)$. (**II**).

From (I) and (**II**), it follows that if M^{Pr}_{CAPSL} refutes $\rho_{CAPSL}(r)$, then there exists a formulation of $\rho_{IS}(r)$, namely $(S1)$, which is refuted by M^{Pr}_{IS}.

For M_{CAPSL} to validate $\rho_{CAPSL}(r)$, then there is no run in M_{CAPSL} found leaky (by algorithm $A1$ [188]). Let ξ be an arbitrary maximal run such that ξ is constructed by algorithm $A1$ [188] and $s_0[\xi\rangle s$. Since ξ is not leaky, $\sigma(t) \notin analz(\sigma_1(s_{\mathbf{I}}))$, for any $\sigma(A\text{-}role) \in \Sigma$ and $\sigma_1 \in \Sigma|_{\mathbf{I}}$. (**III**)

But by Theorem B.3.1, there exists a computation α in M_{IS}, $g_0 \alpha g$ and $s \tilde{p} g$. By part (2) of the definition of $\tilde{p} \subseteq S \times G$ and (I), it is the case that $\Sigma^+(t) \notin g_{Env}.values\_log$. So, for all $\sigma^+(A\text{-}role) \in \Sigma^+$, $\sigma^+(A\text{-}role) \mapsto ag_A$ and $g_{Env}.POOL.t \neq ag_A.t$.

So for any run possibly constructable in M_{IS}, its maximal state g is such that $g_{Env}.POOL.t \neq ag_A.t$. Given the persistence of value-setting in M_{IS}, it follows that any run possibly constructable in M_{IS}, it is such that any of its state g has the property $g_{Env}.POOL.t \neq ag_A.t. \quad (***)$

Again, recall the expression $(S1)$ for $\rho_{IS}(r)$ (refer to Chapter 3, page 111). So, by $(***)$ and by the semantics of M_{IS}, M_{IS} validates $(S1)$. (**IV**).

Now recall the expression $(S2)$ for $\rho_{IS}(r)$: $\bigwedge_{i \in \cup ag_A} AG(\neg K_{Env}(Holds(A, var)))$ (refer to Chapter 3, page 111).

Remember that we are considering an arbitrary computation α in M_{IS}. For M_{IS} to validate also $(S2)$ it is needed that $g \models \neg K_{Env}(Holds(A,t))$, for all $g \in G$ along such arbitrary α.

For this to hold in M_{IS}, it should be that there exists $g' \in G$, $g \sim_{Env} g'$, $g' \not\models Holds(A,t)$. (**iv**)

Recall that at the maximal state g of α, it is the case that $g_{Env}.POOL.t \neq ag_A.t$ (i.e., $(***)$). Hence, $Holds^{\mathbb{I}}(A,t)$ is false at such g_{Env}. And trivially $g \sim_{Env} g$. So, (**iv**) holds indeed.

Hence, if no run is leaky in M_{CAPSL}, then M_{IS} validates $(S2)$ too (**V**).

From (**III**), (**IV**), (**V**), it follows that if M^{Pr}_{CAPSL} validates $\rho_{CAPSL}(r)$, then M^{Pr}_{IS} validates all formulations of $\rho_{IS}(r)$.

The case for r an agreement goal is similar, the reasoning being about two roles at a time (e.g., A, $B \in Ho$) and ξ a maximal run with respect to one of them (e.g., the A-*role*). ▮

Theorem B.3.2 shows the preservation of satisfaction/refutation of the $\asymp$-corresponded goals in the M_{CAPSL} and M_{IS} models. This guarantees that our M_{IS} semantics is aligned with standard trace-based semantics. This alignment only refers to atomic goals. Also, the proof certify that the algorithm *tr* of obtaining Υ_{IS} out of M_{CAPSL} is correct.

In Section 4.2.1 we have given an algorithm called *tr* that produces M^{Pr}_{IS} from M^{Pr}_{CAPSL}. Then, using the relations and lemmas given in Section 4.3, we have hereby shown that this production enforces the alignment of validation/refutation of $\asymp$-correspondent `CAPSL` goals in the two models. Our proofs above are based on structural induction on the *tr*-corresponded computations in the two models and the relations established between points on these computations. In Section 4.4, we show the alignment of validation/refutation of $\asymp$-correspondent `CAPSL` goals in the two models by means of algebraic structures and homomorphic relations.

B.4 Stuttering Equivalence between M_{CAPSL} and M_{IS}

In this section we are going to show that there is more than a homomorphism/epimorphism between M_{CAPSL} and M_{IS}; we will show that the two models are actually stuttering equivalent [37].

The relation $\tilde{p}$ is close to the one underlining this stuttering equivalence between the two models. The relation $\tilde{p}$ is too fine to be the stuttering equivalence relation. However, if we dismiss the third point in the definition of the $\tilde{p}$-relations then we obtain a relation which indeed underlines a stuttering equivalence between the two models.

Let us call this relation *st*. We begin by restating this relation.

Definition B.4.1 (Relation *st* between States in M_{CAPSL} and in M_{IS}) *Let M^{Pr}_{CAPSL} and Υ^{Pr}_{IS} be the models in algorithm tr, as above and let M^{Pr}_{IS} be the unwinding of Υ^{Pr}_{IS}. Let ξ be an arbitrary*

(T,k)-run in M^{Pr}_{CAPSL} and α be a computation in M^{Pr}_{IS}. Let $s \in S$ be a non-initial state of ξ and $g \in G$ be a non-initial state of α. State $s \in S$ is in the relation $st \subseteq S \times G$ with $g \in G$, *$s\, st\, g$, if:*

for $A \in Ho$, $\sigma(A\text{-}role) \in \Sigma$ with $\sigma(A) \neq \mathbf{I}$, $\sigma^+ \in \Sigma^+$, $\sigma^+ \cong \sigma$, $\sigma^+(A\text{-}role) \mapsto_{\Upsilon_{IS}} ag$,

for $Gen \subset Generated^A$, for $t \in Atoms^A$ arbitrary:

(1). $[\sigma(t)/t] \in analz(\sigma(s_A \cup Gen)) \Rightarrow [\Sigma^+(t)/t] \in g_{ag}$

for $\sigma_1 \in \Sigma|_{\mathbf{I}}$, $\sigma_1^+ \in \Sigma^+, \sigma_1^+ \cong \sigma$, $\sigma_2^+ \supset \sigma_1^+$, with $\sigma_2^+(\mathbf{I}) \mapsto_{\Upsilon_{IS}} Env$:

(2). $\forall t \in T_0$, $\Sigma(t) \notin analz(\sigma_1(s_I)) \Rightarrow \Sigma^+(t) \notin g_{Env}.values\_log$

Two states $s \in S$ and $g \in G$ are in the st-relation if honest agents has the same terms at g as they analyse at s and the intruder lacks the same terms at g as it lacks at s.

We note that this relation shaped by the formulation of goals only: i.e., t is secret at s if $t \in analz(s_A) \setminus analz(s_I)$.

Remark B.4.2 *The formulation of initial secrecy and agreement on initials in the two models (i.e., g goal such that $\rho_{CAPSL}(g) \asymp \rho_{IS}(g)$) are invariant with respect to the relation st.*

Also, note that the initial states in the two models are in the st-relation. I.e., the algorithm tr produces the states of M_{IS} in such a relation with those of M_{CAPSL}.

In Section B.1 we recalled the definition of stuttering equivalence originally given in [37].

Theorem B.4.3 *Let M^{Pr}_{CAPSL} and Υ^{Pr}_{IS} be the models in algorithm tr, as above and let M^{Pr}_{IS} be the unwinding of Υ^{Pr}_{IS}. Then, the relation st induces a stuttering equivalence between the M^{Pr}_{CAPSL} and the M^{Pr}_{IS} model.*

Proof (Sketch)

Part 1 of the proof follows from the fact that in M_{CAPSL} there are no actual atomic propositions.

We now prove part 2 of the definition.

Consider a fragment of computation in M_{CAPSL} where a send event has been applied. Then, the correspondence with M_{IS} is illustrated below.

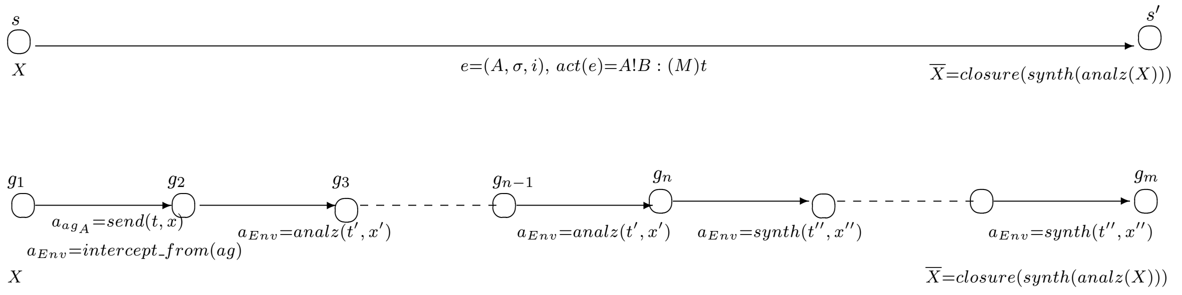


Figure B.1 Fragments of Computation in M_{CAPSL} and in M_{IS}

For such a fragment, assume that $s\,st\,g_1$. This is a reasonable assumption since states along all M_{CAPSL} and M_{IS} are in the $\tilde{p}$-relation (i.e., Theorem B.3.1) and, by the way algorithm *tr* works, if $s\,\tilde{p}g$ then it is implied that $s\,st\,g$. $(*)$

Then, for such a fragment, take $B_1 = \{s\}$, $B_2 = \{s'\}$ and $B_1' = \{g_1\}$, $B_2 = \{g_2, \ldots, g_n, g_{n+1}, \ldots, g_m\}$.

For all the equivalence relations $E_0, \ldots, E_{n+1}$ in the definition of stuttering equivalence, we considered only the *st*-relation.

Given $(*)$, for part A we have to prove that $s'\,st\,g_2$, $s'\,st\,g_3$, …, $s'\,st\,g_n$, $s'\,st\,g_{n+1}$, …, $s'\,st\,g_m$. All the points in the definition of *st* follows for all these states in the same way it did when we were proving the preservation of the $\tilde{p}$-relation (See Lemma 4.3.8 and Lemma 4.3.16).

We make a side note. Since the closure of analysis in M_{IS} has not yet occurred at $g_2, \ldots, g_n$, it is the third point in the definition of $\tilde{p}$-relation that would have not followed for states (s', g_2), …, (s', g_n). But, this third point was dropped in the definition of the *st*-relation precisely for this reason.

So, part A of point 2 follows follows.

Part B of point 2 follows as part A above and by the construction in algorithm *tr* (i.e., all states and actions in M_{IS} as in Figure B.1 are constructed for events and runs in M_{CAPSL}).

A similar reasoning can be made about receiving events in M_{CAPSL}. As paths alternate in this way only (see Section 4.3), it follows that the models are stuttering equivalent with respect to the relation *st*). ▮

From Theorem B.4.3, Remark B.4.2 and the well-established theory of stuttering equivalence [37] it follows that the $\asymp$-related goals are indeed validated/refuted in the M_{CAPSL} model if and only if they are in the M_{IS} model.

With Theorem B.3.2 and Theorem B.4.3 we have shown, by other means than those in Chapter 4, that our M_{IS} model is aligned with standard, trace-based models for `CAPSL`-described protocols (as far as traditional goal-specifications go).

Appendix C

Automatic Generation and Verification of Models for Receiver-Opaque Protocols

In this appendix we present mainly an extension of the methodology used in Chapter 7 to model and verify receiver-opaque protocols. We present a tool that automatically generates `ISPL` programs encoding the $\Upsilon_{IS}^{\mathbb{I}E}$ formalisations of `CAPSL`-described receiver-opaque protocols. We discuss the results of verifying these models against specifications of their security goals in logics of time and interrogative knowledge.

C.1 Introduction

In Chapter 7 we presented an IS-based formalism for encoding protocols specified by convergent equational theories. These protocols are receiver-opaque[1](RO).

For sake of clarity, the main, practical application of Chapter 7 focused on e-voting protocols only. Following existing theoretical results applicable to e-voting [1], we thereby restrained the intruder model to a passive Dolev-Yao attacker. However, in our bounded-size instantiation approach, we can experiment on stronger threat-models (i.e., towards an active Dolev-Yao intruder). We can

[1]For details, see Definition 3.1.5.

also extend the technique to be applied to other classes of protocols (e.g., advanced authentication protocols). This appendix will expand on these matters.

Another characteristic of Chapter 7 is that it focused on the theoretical model and it did not detail on the implementation of a tool for the automatic generation of the models. Also, it only discussed the experiments yielded by automatically generating and verifying the FOO'92 [87] e-voting protocol.

In this appendix we will present extensions of the ideas in Chapter 7, thus diminishing its aforementioned limitations. We will also describe a tool that automatically generates models for ROP, described in `CAPSL` via equational theories, which are more general than those in Chapter 7. Stronger thread-models than the ones in Chapter 7 will be comprised in the generated models. In the last part of the appendix, we will evaluate the results of using this methodology to verify several ROP.

C.2 Protocol Model

To a large extent, the protocol model employed in this appendix to encode ROP is the one already described in Chapter 7. Therefore, we do not re-insist here on the model per se. In turn, we underline the modifications to be operated in order to make the $\varUpsilon_{IS}^{\mathbb{I}E}$ formalisation applicable to the entire class of ROP protocols (i.e., not only to those protocols expressed by convergent cryptographic equational theories).

The model in Chapter 7 can be extended to work for an equational theory that is not necessarily convergent, but only terminating[2]. To do so, we need to consider predicates for terms[3] not only for normal terms as in Chapter 7, but for all the terms in the local states of an agent. In the above fashion, the interrogative knowledge will mean a cryptographically correct "inspection" of *all* the terms in the agents' possesion. By "correct", we intend that the evaluation of the predicates for terms will be accordance with the interpretation in the denotational algebra of the underlying equational theory. The evaluation of the predicates would naturally evolve at the reachable states,

[2]For details, see page 243 for details.

[3]For details, see Definition 7.3.11.

as the terms in the local states get re-assigned by the evolution function. This adjustment maintains the result in Theorem 7.3.15 (i.e., it maintains the approximation of quotient knowledge through interrogative knowledge). Theorem 7.3.15 was based on the unique homomorphism between a term algebra and its quotient algebra modulo the equational theory. Once we encode interrogations for all the terms of any agent, we will be resorting simply to the identity homomorphism in the term algebra of the theory (Σ, E).

The obvious trade-off in enlarging the interrogation-sets[4] is the size of the models (i.e., encoding interrogations not only for normal terms, but for all the terms yields an increase in the size of the interrogation-sets and therefore in the size of the $\Upsilon_{IS}^{\mathbb{I}E}$ models).

Note that all protocols used in practice are described by a terminating equational theory. Hence, the $\Upsilon_{IS}^{\mathbb{I}E}$ formalism modified as above has the benefit of allowing us to model any protocol in an IS formalisation, using a correct cryptographic indistinguishability relation.

C.3 Model-Generation Tool

Our desideratum is to start from a `CAPSL` description of an equationally specified protocol and automatically generate the $\Upsilon_{IS}^{\mathbb{I}E}$ models (amended as above). In order to encode the equational rewriting, we employ the intermediate format of `CAPSL`, the `CIL` language. For details on `CIL`, see page 57.

C.3.1 A `CAPSL` to `CIL`-like Translator

As we mentioned in Chapter 7, we have modified the `CAPSL` to `CIL` translator [65] into a translator from `CAPSL` to a `CIL`-like language. This `CIL`-like language has a syntax that we have designed such that it brings the `CAPSL`-intermediate format closer to an IS semantics. In engineering the `CIL`-like language, we did not modify the semantics of `CIL` (i.e., our `CIL`-like language still describes the equational rewriting involved in each protocol role). We only reformatted the final aspect of a `CIL` file. This final output is much closer to the `ISPL` language[5].

[4]For details, see Definition 7.3.4.

[5]For details on `ISPL`, see page 28.

In the `CIL`-like format, the rewriting within each protocol-role is encoded in a way which is in keeping with the grammar for an agent's evolution function in `ISPL` (i.e., ⟨postconditions⟩ if ⟨preconditions⟩). We have modified amongst others the name of the actions, their parameters, to suit the $\Upsilon_{IS}^{\mathbb{I}E}$ formalisation, rather than the original, MSR-based format of `CIL`.

In Chapter 2, page 54, we presented several details on the RO protocol due to Gong [94]. It is an authentication protocol, but its underlying cryptographic primitives are based on one-way hash functions rather than on simple encryption/decryptions. Hence, the original Υ_{IS} formalisation in Chapter 3 and the `PD2IS` toolkit are not suited to model and verify this protocol (i.e., the indistinguishability relation thereby used is too coarse). In the following, we present an excerpt of the Gong protocol, compiled in this `CIL`-like intermediate format by means of our modified translator.

Example C.3.1 (CIL-like Format for the Gong Protocol [94])

```
rule ( facts ( state ( roleB , 3 , terms ( B , S , Nb , Na , A , Kh , Ns ) ) )
      if facts ( state ( roleB , 2 , terms ( B , S , Nb , Na , A ) ),
               forwardmsg __ from __ UNK __ to __ B ___ terms (
               Ns , xor ( sha ( cat ( Ns , cat ( Nb , cat ( A , csk ( B ) ) ) ) ),
               Kh ) , sha ( cat ( Kh , csk ( B ) ) ) ) ) and ids ( ) )
```

This `CIL`-like excerpt in Example C.3.1 shows that we added a special $forward$ action to the `CIL` syntax. It symbolises that the intruder overhears all the communication and, in this case, transmits to the *B-role* participant the 2-nd step message (i.e., $roleB, 2\ldots$).

In our translator, we process the messages and explicate their format as the ordered list of their inner parts. To do so, we process the equational theory in the `CAPSL` description of protocol (i.e., the `TYPESPEC` section of the `CAPSL` description). In the case of the Gong protocol, when the *B-role* participant "expects" a second step message, it ought to receive the nonce Ns, the result of $xor(sha(cat(Ns, cat(Nb, cat(A, csk(B))))), Kh)$ and the hash-value $sha(cat(Kh, csk(B)))$. By applying the equation annotating step 2 of the protocol (see Example 2.2.5), the *B-role* participant updates its state to the third step tuple where he "possesses" Kh and Ns.

We used the standard `CAPSL` to `CIL` translator [65] and also compiled the Gong protocol in the original `CIL` syntax. Excerpts of that file were given in Example 2.2.6. The interested reader can compare the above `CIL`-like excerpt with the original `CIL` output, by revisiting Example 2.2.6.

C.3.2 The `PD2I-IS` Toolkit

The `PD2I-IS` (Protocol `D`escription to `I`nterrogative `I`nterpreted Systems) toolkit is a similar software to `PD2IS`, but it applies to protocols which are essentially receiver-opaque (i.e., not reducible[6] to receiver-transparent protocols). `PD2I-IS` starts from a `CAPSL` protocol description, containing a `TYPESPEC` section specifying a terminating equational theory, and it produces the $\Upsilon_{IS}^{\mathbb{I}E}$ formalisation in `ISPL`. Unlike `PD2IS`, it encompasses the aforementioned translator to obtain an intermediate `CIL`-like file. It then processes this `CIL`-like file into an `ISPL` representation of the $\Upsilon_{IS}^{\mathbb{I}E}$ formalisation. It now becomes clear that the alteration of `CIL` for its use as an intermediary between `CAPSL` and `ISPL` has viewed the ease in processing the rewriting aspects of the protocol into a final `ISPL` file denoting the multi-session execution of an equationally specified protocol.

The architecture of `PD2I-IS` is similar to that of `PD2IS` (see Chapter 5). However, the state of agents are generated not according to the Υ_{IS} formalisation, but following the more intricate $\Upsilon_{IS}^{\mathbb{I}E}$ formalisation (i.e., local states comprise terms and not only atomic terms). The local evolution functions of honest agents are a fully ground instantiation of the `CIL`-like format described before. This instantiation is in accordance to the input supplied by the user (i.e., a fixed scenario or a loose scenario[7] in a XML file as in `PD2IS`). As `PD2IS`, `PD2I-IS` can generate both constrained and un-constrained models for fixed and loose scenarios (see Section 5.2, for details).

The $\Upsilon_{IS}^{\mathbb{I}E}$ formalisations generated by `PD2I-IS` can comprise an active Dolev-Yao intruder encoded in the Environment agent or, like in Chapter 7, only a passive intruder. In the former case, we implement the generation of an active Dolev-Yao intruder that applies the equational theory. Its states and actions are as in Chapter 3 and their generation as in `PD2IS`. The novel aspects are in the protocol and evolution functions. The Dolev-Yao analysis and synthesis implemented adhere to the underlying equational theory (i.e., not simply to decryption and encryption). In `PD2I-IS`,

[6]For details, see Section 5.1.

[7]For details, see page 192 and/or Figure 5.4.

messages and composites are tagged with the equations that apply to them (i.e., upon the `CAPSL` protocol description) and the active intruder's analysis and synthesis is accordingly restricted by these tags.

In `PD2I-IS`, we generate interrogation-sets, i.e., local predicates for each term in the equational theory. If the underlying theory is convergent, we restrict the interrogation-sets generated to contain predicates for the normal terms only (as per Chapter 7).

The authentication goals are translated into formulae following the same systematisation as in Chapter 3. Other goals (e.g., e-voting requirements) are not essentially supported by the `CAPSL` grammar. However, we implement the systematisation presented in Chapter 7 for vote-privacy, receipt-freeness and coercion-resistance.

`PD2I-IS` is linked to the `MCMAS-I`[8] model checker [31] that supports the `ISPL` specification and the verification of interrogative knowledge operators. Therefore, the $\Upsilon_{IS}^{\mathbb{I}E}$ formalisations generated with `PD2I-IS` are automatically verified by `MCMAS-I`. Details on the possible counterexample found (i.e., protocol attacks) are returned by `PD2I-IS`.

`PD2I-IS` is coded in `Java` and in `Maude`. It is currently in a beta-version. This version is available at [29].

C.4 Evaluation of the Methodology

In this section we first evaluate the use of the `PD2I-IS` toolkit. Then, we discuss the results of using the `MCMAS-I` model checker [31] to verify models for ROP generated with `PD2I-IS` against specifications of their security goals in logics of time and interrogative knowledge.

C.4.1 On the Model-Generation with `PD2I-IS`

The $\Upsilon_{IS}^{\mathbb{I}E}$ formalisations are in general larger than the Υ_{IS} formalisation. One reason is the additional interrogation-sets. This is due in part to the additional predicates for terms. However, some of the composites implied by the equational theory (e.g., $xor(sha(cat(Ns, cat(Nb, cat(A, csk(B))))), Kh)$

[8]For details, see Chapter 7, page 269.

in the Gong protocol, featured in Example C.3.1) are of considerable length themselves. Hence, the possible range for these terms is usually large. In ROP, few atoms are owned[9] by agents and most composites are opaque. Therefore, the restricted ranges[10] for terms in each agent do not contribute considerably to diminish the range of a message. Moreover, the value-space for actions is even larger than the range of the composites (e.g., consider the action $forwardmsg\_from\_UNK\_to\_B\_terms(Ns,$ $xor(sha(cat(Ns, cat(Nb, cat(A, csk(B))))), Kh), sha(cat(Kh, csk(B))))))$. Therefore, the local states, the local evolution functions are also larger in the $\Upsilon_{IS}^{\mathbb{I}E}$ formalisation than in the Υ_{IS} formalisation.

The above justifies the reasons behind our very distinct, separate designs in the handling of RT and RO protocols, respectively.

All of the above also imply that the generation time for $\Upsilon_{IS}^{\mathbb{I}E}$ formalisations is sometimes even impractical and, generally, much larger than that of simpler Υ_{IS} formalisations.

In the context of the above, we often resort to intermediate writing to disk[11] when we generate the $\Upsilon_{IS}^{\mathbb{I}E}$ formalisations using `PD2I-IS`. We have successfully generated `ISPL` code for several constrained $\Upsilon_{IS}^{\mathbb{I}E}$ formalisations of sessions comprising up to three agents and the Dolev-Yao intruder for the Gong authentication protocol [94], the KSL repeated-authentication protocol [111] and the Okamato e-voting protocol [159]. Nevertheless, the generation of some of these constrained models takes up to 10 minutes.

The automatic generation of fully ground IS representations of multi-session executions for ROP is resource-consuming. Nevertheless, this generation is systematic and fully automatic. Therefore, we consider that it is preferable to other formalisms [2, 63] that tackle the modelling of ROP in a pen-and-paper, ad-hoc manner. Also, `PD2I-IS` provides a means to automatic verification of cryptographic knowledge in a well-founded semantics for epistemic modellings.

C.4.2 On Attack-Finding with `PD2I-IS`

In this subsection we are going to report on the finding of attack on ROP modelled automatically with `PD2I-IS`.

[9]For details, see Chapter 3, page 69.

[10]For details, see Definition 3.3.13.

[11]For details, see page 200.

The machine used for the following evaluation was based on an Intel Core 2 Duo processor clocked at 3.00 GHz, with a 6144 KiB cache. The machine ran 32-bit Fedora Core 12, kernel 2.6.32.10.

`PD2I-IS` provides a systematic method for model checking MAS models for ROP protocols. Some of these protocols (e.g., KSL) have been already verified by the reduction of their description to RTP. Others are essentially receiver-opaque (i.e., cannot be easily reduced to a RTP) and their MAS models have been analysed using the `PD2I-IS` only. For the first category, we can compare and contrast the performance of verifying them as ROP or as RTP. For the latter category, we will simply evaluate the verification result in a standalone manner.

The KSL Protocol [111] under `PD2I-IS`. For the KSL protocol, we were able to find the same attack reported in Chapter 5, page 207 and those discussed in Chapter 6. Whilst in Chapter 5 we reduced the KSL protocol to an RTP, here we automatically generated its ROP execution models with `PD2I-IS`.

The time reported in Chapter 5 to find its attacks was of 17 seconds, whereas the maximum time needed with `PD2I-IS` was 3 days. The state-space of the unwound model was in the order of 10^6.

We conclude that whilst some trade-offs[12] are to be considered, the generation and verification of MAS models for RTP where possible is a more promising line than that of producing and analysing MAS models for ROP.

The Okamoto Protocol [159] under `PD2I-IS`. The Okamoto protocol [159] is an e-voting protocol similar to the FOO'92 protocol presented in Chapter 7. Its underlying equational theory is not convergent, but it is terminating.

We generated similar models to $\mathcal{M}_1$–$\mathcal{M}_3$ classes of e-voting $\Upsilon^{\mathbb{I}E}_{IS}$ formalisations[13], where the intruder was an insider (i.e., not an active Dolev-Yao attacker). Practice has shown that an active Dolev-Yao attacker makes the analysis of the generated `ISPL` files impractical (i.e., the files cannot

[12]For details, see page 182.

[13]For details, see Chapter 7, page 277

be parsed by the `MCMAS-I`-parser into unwound models). With `PD2I-IS`, we generated the VP, VVU RF, CR formulations of e-voting goals in `ISPL` using the interrogative knowledge modality. `MCMAS-I` was called for the verification of the models against these formulae. We found that vote-privacy and receipt-freeness were satisfied, whereas coercion-resistance can be violated. Table C.1 reports this and also the metrics involved in the verification with `MCMAS-I` (i.e., state-space of unwound models, verification time, etc.)

Table C.1 Verifying Okamoto Models Generated with `PD2OIS`

Model	Formula	Memory (KiB)	Time (s)	States
$\mathcal{M}_1$	VP/VVU (holds)	196032	$\sim 7 \times 10^5$	$\sim 10^{12}$
$\mathcal{M}_2$	RF (holds)	195452	$\sim 7 \times 10^5$	$\sim 10^{12}$
$\mathcal{M}_3$	CR (fails)	196922	$\sim 7 \times 10^5$	$\sim 10^{12}$

To sum up, `PD2I-IS` enabled us to generate models for e-voting, ROP in a systematic and automatic way. With `MCMAS-I` and its underlying methodology we were able to verify security requirements essentially pertaining to knowledge (e.g., coercion-resistance, receipt-freeness) in a sound, well-founded epistemic-based semantics. We believe that our fully-fledged approach is preferable to the partially systematic formalisation of [63] in modelling and verifying e-voting. However, the verification times yielded by our methods remain large (mainly due to the fully ground methodology embraced).

The Gong Protocol [94] under `PD2I-IS`. We were able to generate the $\Upsilon_{IS}^{\mathbb{I}E}$ formalisations for the Gong protocol, encompassing an active Dolev-Yao model. Moreover, `MCMAS-I` is able to build the corresponding unwound $M_{IS}^{\mathbb{I}E}$ model. Unfortunately, the verification of the $M_{IS}^{\mathbb{I}E}$ models for the Gong protocol with `MCMAS-I` has been halted after 3 weeks of `MCMAS-I` running, by a `CUDD` [179] library[14] internal error. We were unable to resolve this error.

In light of the above, comparisons with other state-of-the-art tools handling such protocols is not possible. However, we are not aware of tools that would not treat such advanced protocols in a

[14]For details, see page 28.

bespoke way (i.e., but rather treat their modelling in a systematic and uniform manner, as in the approach hereby presented).

C.5 Conclusions

In this appendix we have systematised and implemented a methodology to analyse receiver-opaque protocols. This is based on an approach previously presented in Chapter 7. In this appendix we present a tool called `PD2I-IS`. For receiver-opaque protocols described in `CAPSL` under terminating equational theories, `PD2I-IS` automatically generates IS-based formalisations in `ISPL`. These formalisations are denoted $\Upsilon_{IS}^{\mathbb{I}E}$ and have been systematically described in Chapter 7 and in the present appendix.

The receiver-opaque protocols hereby analysed are more complicated than those used in Chapters 3–6 in that the underlying cryptography is much more intricate than simple decryption/encryption.

With this appendix, we have now demonstrated that we can apply our MAS-based methodology for modelling and verifying a very large class of security protocols. Essentially, our methodology can be applied to all traditional cryptographic protocols. The un-handled remainder consists of less conventional protocols, like those used in contract-signing and auctioning.

Whilst the methodology for generating and verifying MAS models for ROP protocol is well-founded, systematic and automatic, it is clearly outperformed by the approach we took on RTP. Nevertheless, in the formalisms and tools [9, 63] that handle such advanced ROP usually resort to bespoke, ad-hoc modellings. In that sense, with `PD2I-IS` and the hereby presented approach, we have shown that we were able to develop a concept, uniform platform to handle the verification of intricate cryptographic protocol in a systematic and automatic way.