Strand Spaces with Choice via a Process Algebra Semantics *

Fan Yang
University of Illinois at Urbana-Champaign, USA
fanyang6@illinois.edu

Santiago Escobar
Universitat Politècnica de València, Spain
sescobar@dsic.upv.es

Catherine Meadows
Naval Research Laboratory, Washington DC, USA
meadows@itd.nrl.navy.mil

José Meseguer
University of Illinois at Urbana-Champaign, USA
meseguer@illinois.edu

Sonia Santiago
University of Illinois at Urbana-Champaign, USA
sosanpi@gmail.com

Abstract

Roles in cryptographic protocols do not always have a linear execution, but may include choice points causing the protocol to continue along different paths. In this paper we address the problem of representing choice in the strand space model of cryptographic protocols, particularly as it is used in the Maude-NPA cryptographic protocol analysis tool.

To achieve this goal, we develop and give formal semantics to a process algebra for cryptographic protocols that supports a rich taxonomy of choice primitives for composing strand spaces. In our taxonomy, deterministic and non-deterministic choices are broken down further. Non-deterministic choice can be either *explicit*, i.e., one of two paths is chosen, or *implicit*, i.e. the value of a variable is chosen non-deterministically. Likewise, deterministic choice can be either an (explicit) *if-then-else* choice, i.e. one path is chosen if a predicate is satisfied, while the other is chosen if it is not, or *implicit deterministic choice*, i.e. execution continues only if a certain pattern is matched. We have identified a class of choices which includes finite branching and some cases of infinite branching, which we address in this paper.

Our main theoretical results are two *bisimulation* results: one proving that the formal semantics of our process algebra is bisimilar to the forwards execution semantics of its associated strands, and another showing that it is also bisimilar with respect to the symbolic backwards semantics of the strands such as that supported by Maude-NPA. At the practical level, we present a prototype implementation of our process algebra in Maude-NPA, illustrate its expressive power and naturalness with various examples, and show how it can be effectively used in formal analysis.

Keywords cryptographic protocol analysis, rewriting-based model checking, narrowing-based reachability analysis, process algebra

1. Introduction

Formal analysis of cryptographic protocols has become of the most successful applications of formal methods to security, with a number of tools available and many successful applications to the analysis of protocol standards. In the course of developing these tools it has become clear that cryptographic protocols that there are certain universal features that can best be handled by accounting for them directly in syntax and semantics of the formal specification language, e.g. unguessable nonces, communication across a network controlled by an attacker, and support for the equational properties of cryptographic primitives. Thus a number of different languages have been developed that include these features.

At the same time, it is necessary to provide support for more commonly used constructs, such as *choice points* that cause the protocol to continue in different ways, and to do so in such a way that they are well integrated with the more specifically cryptographic features of the language. However, in their original form most of these languages do not support choice, or support it only in a limited way.

In particular the strand space model [10], one of the most popular models designed for use in cryptographic protocol analysis, does not support choice in its original form; strands describe linear sequences of input and output messages, without any branching. One response to dealing with this limitation, and to formalizing strand spaces in general has been to embed the strand space model in some other formal system that supports choice, e.g. event-based models for concurrency [4], Petri nets [11], or multi-set rewriting [3]. However, we believe that there is an advantage in introducing choice in the strand space model itself, while proving soundness and completeness with another formal system in order to validate the augmented model. This allows us to concentrate on handling the types of choice that commonly arise in cryptographic protocols. In this paper we describe such a choice model that we have developed for the Maude-NPA cryptographic protocol analysis tool, which uses a strand space semantics. Such an approach allows us to represent not only finitely branching choice, but various types of infinitely branching choice that arise in cryptographic protocols.

Previous to this work, Maude-NPA offered a number of ways of handling choice, but its scope was limited, and a uniform semantics of choice was lacking. Many kinds of branching could be handled by a protocol composition method [20], in which a single parent strand could be composed with one or more child strands. Although it was designed for the modular construction of protocols, it could also, with the appropriate restrictions, be used to express both non-deterministic branching and deterministic branching predicated on pattern matching of output parameters of the parent with the input parameters of the child. However, repurposing composition to branching had its limitations. First of all, it was possible to inadvertently introduce non-deterministic choice into what was intended to be deterministic choice by unwise choice of input and output parameters. Secondly, the limitation to pattern matching

* S. Escobar has been partially supported by the EU (FEDER) and the Spanish MINECO under grants TIN 2015-69175-C4-1-R and TIN 2013-45732-C4-1-P, and by Generalitat Valenciana under grant PROMETEOII/2015/013. J. Meseguer has been partially supported by NSF grant CNS-131910.

ruled out certain types of deterministic choice conditioned on predicates that could not be expressed this way, e.g. disequality predicates. Finally, implementation of choice via composition can also be inefficient, since Maude-NPA must evaluate all possible child strands that match a parent strand.

Maude-NPA, in common with many other cryptographic protocol analysis tools, also offers a type of implicit choice that does not involve branching: non-deterministic choice of the values of certain variables. For example, a strand that describes an initiator communicating with a responder generally uses variables for both the initiator and responder names; this represents a non-deterministic choice of initiator responder identities. However, the semantic implications of this kind of choice were not that well understood, which made it difficult to determine where it could safely be used. Clearly, a more unified treatment of choice was necessary, together with a formal semantics of choice.

In support of this work we have developed a taxonomy of choice in which the categories of deterministic and non-deterministic choice are further subdivided. First of all, non-deterministic choice is subdivided into *explicit* and *implicit* non-deterministic choice. In explicit non-deterministic choice a role chooses either one branch or another at a choice point non- deterministically. In implicit non-deterministic choice a logical *choice variable* is introduced which may be non-deterministically instantiated by the role. Deterministic choice is subdivided into (explicit) *if-then-else* choice and *implicit deterministic choice*. In if-then-else choice a predicate is evaluated. If the predicate evaluates to true one branch is chosen, and if it evaluates to false another branch is chosen. Deterministic choice with more than two choices can be modeled by concatenation of if-then-else choices. In implicit deterministic choice, a term pattern is used as an implicit guard, so that only messages matching such pattern can be chosen i.e., accepted, by the role. Although implicit deterministic choice can be considered a special case of if-then-else choice in which the second branch is empty, it is often simpler to treat it separately. Classifying choice in this way allows us to represent all possible behaviors of a protocol by a finite number of strands modeling possible executions, while still allowing the variables used in implicit non-deterministic and deterministic choice to be instantiated in an infinite number of ways.

Consider the following example, which exhibits all four types of choice:

$$
\begin{array}{ll}
(Init) & ((+(A_? \,;\, B_? \,;\, PubKey) \cdot -(pk(A_?, B_? \,;\, SK)) \\
& \quad ? \\
& \quad (+(A_? \,;\, B_? \,;\, SharedKey) \cdot -(e(key(A_?, B_?), B_? \,;\, SK)) \\
(Resp) & -(A \,;\, B \,;\, TEnc) \cdot \\
& \quad \mathit{if}\ TEnc = PubKey \\
& \quad\quad \mathit{then}\ (+(pk(A, B \,;\, skey(A, B, r')))) \\
& \quad\quad \mathit{else}\ (+(e(key(A, B), B \,;\, skey(A, B, r'))))
\end{array}
$$

In the initiator role the principal names are chosen using implicit nondeterministic choice. This is represented by choice variables of the form $X_?$. The initiator role then uses the explicit nondeterministic choice operator ? to determine whether or not to initiate a public or shared key version of the protocol. The responder role in turn uses implicit deterministic choice to determine whether to proceed after receiving the first message, proceeding only if that message satisfies the pattern specified by $A \,;\, B \,;\, TEnc$, where A, B, and $TEnc$ are *pattern variables*. It then uses if-then-else deterministic choice to decide whether to execute the public or shared key version of the protocol, depending on the value of $TEnc$.

We can see that the possible paths through the protocol can be described using two types of principals for each role: one principal that executes the public key version, and one that executes the shared key version. This allows us to encode choice directly in the Maude-NPA semantics, with one slight addition: strands must now include the predicates evaluated when if-then-else choices are made, and the truth value of each such predicate is treated as a constraint that must be satisfied by the result of any further state transition. We refer to this as the *constrained backwards semantics*, since Maude-NPA performs backwards search.

The problem still remains of verifying that this method of handling choice corresponds to standard notions of the way choice is made. To this end we develop a process algebra semantics that incorporates the different types of choice, and we prove soundness and completeness of the strand space choice semantics with respect to the process algebra semantics. This is nontrivial, since there are two major ways in which the two semantics differ. The first is that a process algebra "forgets" its past, while strands remember theirs. The second is that Maude-NPA uses backwards search, while the process algebra proceeds forward. We deal with these issues by introducing an intermediate semantics, a forward strand space semantics originally introduced in [7] and augmented here with choice operators and operational semantic rules to produce a *constrained forwards semantics*. We first prove soundness and completeness of the process algebra semantics with respect to the forwards semantics, using labeled transitions to keep track of the history of the process algebra execution. We then generalize the soundness and completeness proof in [7] to prove soundness and completeness of the forwards semantics with respect to the backwards strand semantics with choice incorporated.

The rest of the paper is organized as follows. After some preliminaries in Section 2 and a high level introduction of the Maude-NPA tool in Section 3, we first define the process algebra syntax and operational semantics in Section 4. In Section 5 we extend Maude-NPA's strand space syntax to include choice operators. In Section 6 we introduce the constrained forwards semantics and prove bisimilarity of the process algebra semantics and the constrained forwards semantics. In Section 7 we define the constrained backwards semantics for Maude-NPA. We then prove that the constrained backwards semantics is sound and complete with respect to the constrained forwards semantics, and therefore, by the results of Section 6, is sound and complete with respect to the process algebra semantics. Finally, in Section 8 we describe some preliminary experiments we have run using Maude-NPA on various protocols exhibiting both deterministic and non-deterministic choice and give some results. In Section 9 we discuss related and future work, in particular the potential of using the process algebra syntax as a specification language. Finally, we conclude in Section 10.

1.1 Motivating Example

In this section we introduce a protocol that we will use as a running example in this paper. It is a simplified version of the handshake protocol in TLS 1.3 [19] a proposed update to the TLS standard for client-server authentication. This protocol, like most other protocol standards, offers a number of different choices that are applied in different situations. In order to make the presentation and discussion manageable within the confines of a conference paper, we present only a subset here: the client chooses a Diffie-Hellman group, and proposes it to the server. The server can either accept it or request that the client proposes a different group. In addition, the server has the option of requesting that the client authenticates itself. We present the protocol at a high level similar to the style used in [19].

Example 1.1. *We let a dashed arrow* $\dashrightarrow$ *denote an optional message, and an asterisk * denote an optional field.*

1. $C \rightarrow S$: *ClientHello, Key_Share*
   The client sends a Hello message containing a nonce and the Diffie-Hellman group it wants to use. It also sends a Diffie-Hellman key share.
   - *1.1* $S \dashrightarrow C$: *HelloRetryRequest*
     The server may optionally reject the Diffie-Hellman group proposed by the client and request a new one.
   - *1.2* $C \dashrightarrow S$: *DHGroup, Key_Share*
     The client proposes a new group and sends a new key share.
2. $S \rightarrow C$: *ServerHello, Key_Share,*
   {*AuthReq**},{*CertificateVerify*}, {*Finished*}
   The server sends its own Hello message and a Diffie-Hellman key share. It may optionally send an AuthReq to the client to authenticate itself with a public key signature from its public key certificate. It then signs the entire handshake using its own public key in the CertificateVerify field. Finally, in the Finished field it computes a MAC over the entire handshake using the shared Diffie-Hellman key. The {} notation denotes a field encrypted using the shared Diffie-Hellman key.
3. $C \rightarrow S$: {*CertificateVerify**}, {*Finished*}
   If the client received an AuthReq from the server it returns its own CertificateVerify and Finished fields.

2. Preliminaries

We follow the classical notation and terminology from [21] for term rewriting, and from [15] for rewriting logic and order-sorted notions. We assume an order-sorted signature $\mathbf{\Sigma} = (\mathsf{S}, \leqslant, \Sigma)$ with poset of sorts $(\mathsf{S}, \leqslant)$. We also assume an S-sorted family $\mathcal{X} = \{\mathcal{X}_\mathsf{s}\}_{\mathsf{s}\in\mathsf{S}}$ of disjoint variable sets with each $\mathcal{X}_\mathsf{s}$ countably infinite. $\mathcal{T}_\Sigma(\mathcal{X})_\mathsf{s}$ is the set of terms of sort s, and $\mathcal{T}_{\Sigma,\mathsf{s}}$ is the set of ground terms of sort s. We write $\mathcal{T}_\Sigma(\mathcal{X})$ and $\mathcal{T}_\Sigma$ for the corresponding order-sorted term algebras. For a term t, $Var(t)$ denotes the set of variables in t.

A *substitution* $\sigma \in \mathcal{S}ubst(\Sigma, \mathcal{X})$ is a sorted mapping from a finite subset of $\mathcal{X}$ to $\mathcal{T}_\Sigma(\mathcal{X})$. Substitutions are written as $\sigma = \{X_1 \mapsto t_1, \ldots, X_n \mapsto t_n\}$ where the domain of σ is $Dom(\sigma) = \{X_1, \ldots, X_n\}$ and the set of variables introduced by terms $t_1, \ldots, t_n$ is written $Ran(\sigma)$. The identity substitution is denoted *id*. Substitutions are homomorphically extended to $\mathcal{T}_\Sigma(\mathcal{X})$. The application of a substitution σ to a term t is denoted by $t\sigma$. For simplicity, we assume that every substitution is idempotent, i.e., σ satisfies $Dom(\sigma) \cap Ran(\sigma) = \varnothing$. Substitution idempotency ensures $t\sigma = (t\sigma)\sigma$. The restriction of σ to a set of variables V is $\sigma|_V$. Composition of two substitutions σ and σ' is denoted by $\sigma\sigma'$. A substitution σ is a ground substitution if $Ran(\sigma) = \varnothing$.

A Σ-*equation* is an unoriented pair $t = t'$, where $t, t' \in \mathcal{T}_\Sigma(\mathcal{X})_\mathsf{s}$ for some sort $\mathsf{s} \in \mathsf{S}$. Given Σ and a set E of Σ-equations, order-sorted equational logic induces a congruence relation $=_E$ on terms $t, t' \in \mathcal{T}_\Sigma(\mathcal{X})$ (see [16]). The E-equivalence class of a term t is denoted by $[t]_E$ and $\mathcal{T}_{\Sigma/E}(\mathcal{X})$ and $\mathcal{T}_{\Sigma/E}$ denote the corresponding order-sorted term algebras modulo E. Throughout this paper we assume that $\mathcal{T}_{\Sigma,\mathsf{s}} \neq \varnothing$ for every sort s, because this affords a simpler deduction system. An *equational theory* (Σ, E) is a pair with Σ an order-sorted signature and E a set of Σ-equations. The E-*subsumption* preorder $\sqsupseteq_E$ (or just $\sqsupseteq$ if E is understood) holds between $t, t' \in \mathcal{T}_\Sigma(\mathcal{X})$, denoted $t \sqsupseteq_E t'$ (meaning that t is *more general* than t' modulo E), if there is a substitution σ such that $t\sigma =_E t'$; such a substitution σ is said to be an E-*match* from t' to t.

An E-*unifier* for a Σ-equation $t = t'$ is a substitution σ such that $t\sigma =_E t'\sigma$. For $Var(t) \cup Var(t') \subseteq W$, a set of substitutions $CSU_E^W(t = t')$ is said to be a *complete* set of unifiers for the equality $t = t'$ modulo E away from W iff: (i) each $\sigma \in CSU_E^W(t = t')$ is an E-unifier of $t = t'$; (ii) for any E-unifier ρ of $t = t'$ there is a $\sigma \in CSU_E^W(t = t')$ such that $\sigma|_W \sqsupseteq_E \rho|_W$; (iii) for all $\sigma \in CSU_E^W(t = t')$, $Dom(\sigma) \subseteq (Var(t) \cup Var(t'))$ and $Ran(\sigma) \cap W = \varnothing$. If the set of variables W is irrelevant or is understood from the context, we write $CSU_E(t = t')$ instead of $CSU_E^W(t = t')$. An E-unification algorithm is *complete* if for any equation $t = t'$ it generates a complete set of E-unifiers. A unification algorithm is said to be *finitary* and complete if it always terminates after generating a finite and complete set of solutions.

A *rewrite rule* is an oriented pair $l \rightarrow r$, where[1] $l \notin \mathcal{X}$ and $l, r \in \mathcal{T}_\Sigma(\mathcal{X})_\mathsf{s}$ for some sort $\mathsf{s} \in \mathsf{S}$. An *(unconditional) order-sorted rewrite theory* is a triple (Σ, E, R) with Σ an order-sorted signature, E a set of Σ-equations, and R a set of rewrite rules.

The rewriting relation on $\mathcal{T}_\Sigma(\mathcal{X})$, written $t \rightarrow_R t'$ or $t \rightarrow_{p,R} t'$ holds between t and t' iff there exist $p \in Pos_\Sigma(t)$, $l \rightarrow r \in R$ and a substitution σ, such that $t|_p = l\sigma$, and $t' = t[r\sigma]_p$. The subterm $t|_p$ is called a *redex*. The relation $\rightarrow_{R/E}$ on $\mathcal{T}_\Sigma(\mathcal{X})$ is $=_E; \rightarrow_R; =_E$, i.e., $t \rightarrow_{R/E} t'$ iff there exists u, u' s.t. $t =_E u \rightarrow_R u' =_E t'$. Note that $\rightarrow_{R/E}$ on $\mathcal{T}_\Sigma(\mathcal{X})$ induces a relation $\rightarrow_{R/E}$ on the free (Σ, E)-algebra $\mathcal{T}_{\Sigma/E}(\mathcal{X})$ by $[t]_E \rightarrow_{R/E} [t']_E$ iff $t \rightarrow_{R/E} t'$. The transitive (resp. transitive and reflexive) closure of $\rightarrow_{R/E}$ is denoted $\rightarrow^+_{R/E}$ (resp. $\rightarrow^*_{R/E}$).

The $\rightarrow_{R/E}$ relation can be difficult to compute. However, under the appropriate conditions it is equivalent to the R,E relation [13] in which it is enough to compute the relationship on any representatives of two E-equivalence classes. A relation $\rightarrow_{R,E}$ on $\mathcal{T}_\Sigma(\mathcal{X})$ is defined as: $t \rightarrow_{p,R,E} t'$ (or just $t \rightarrow_{R,E} t'$) iff there exist $p \in Pos_\Sigma(t)$, a rule $l \rightarrow r$ in R, and a substitution σ such that $t|_p =_E l\sigma$ and $t' = t[r\sigma]_p$.

Let t be a term and W be a set of variables such that $Var(t) \subseteq W$, the R, E-*narrowing* relation on $\mathcal{T}_\Sigma(\mathcal{X})$ is defined as $t \rightsquigarrow_{p,\sigma,R,E} t'$ ($\rightsquigarrow_{\sigma,R,E}$ if p is understood, $\rightsquigarrow_\sigma$ if R, E are also understood, and $\rightsquigarrow$ if σ is also understood) if there is a non-variable position $p \in Pos_\Sigma(t)$, a rule $l \rightarrow r \in R$ properly renamed s.t. $(Var(l) \cup Var(r)) \cap W = \varnothing$, and a unifier $\sigma \in CSU_E^{W'}(t|_p = l)$ for $W' = W \cup Var(l)$, such that $t' = (t[r]_p)\sigma$. For convenience, in each narrowing step $t \rightsquigarrow_\sigma t'$ we only specify the part of σ that binds variables of t. The transitive (resp. transitive and reflexive) closure of $\rightsquigarrow$ is denoted by $\rightsquigarrow^+$ (resp. $\rightsquigarrow^*$). We may write $t \rightsquigarrow^k_\sigma t'$ if there are $u_1, \ldots, u_{k-1}$ and substitutions $\rho_1, \ldots, \rho_k$ such that $t \rightsquigarrow_{\rho_1} u_1 \cdots u_{k-1} \rightsquigarrow_{\rho_k} t'$, $k \geqslant 0$, and $\sigma = \rho_1 \cdots \rho_k$.

3. Overview of Maude-NPA

Here we give a high-level summary of Maude-NPA. For further information, please see [6].

Given a protocol $\mathcal{P}$, states are modeled as elements of an initial algebra $\mathcal{T}_{\Sigma_{SS_\mathcal{P}}/E_{SS_\mathcal{P}}}$, where $\Sigma_{SS_\mathcal{P}} = \Sigma_{SS} \cup \Sigma_\mathcal{P}$ is the signature defining the sorts and function symbols ($\Sigma_\mathcal{P}$ for the cryptographic functions, Σ_{SS} for strand constructor symbols and for all the state constructor symbols), $E_{SS_\mathcal{P}} = E_\mathcal{P} \cup E_{SS}$ is a set of equations where $E_\mathcal{P}$ specifies the *algebraic properties* of the cryptographic functions and E_{SS} denotes properties of constructors of states. The set of equations $E_\mathcal{P}$ may vary depending on different protocols, but the set of equations E_{SS} is always the same for all protocols. There-

[1] Note that we do not impose here the standard condition $Var(r) \subseteq Var(l)$, since extra variables will be introduced in the righthand side of a rule as we will make explicit in the paper. Rewriting with extra variables in righthand sides is handled by allowing the matching substitution to instantiate these extra variables in any possible way. However, this may produce an infinite number of one-step rewrites from a term due to the infinite number of possible instantiations of an extra variable. In Maude-NPA this is avoided by restriction to topmost rewriting, which can be shown to be complete for the tool.

fore, a state is an $E_{SS_\mathcal{P}}$-equivalence class $[t]_{E_{SS_\mathcal{P}}} \in T_{\Sigma_{SS_\mathcal{P}}/E_{SS_\mathcal{P}}}$ with t a ground $\Sigma_{SS_\mathcal{P}}$-term.

In Maude-NPA a *state pattern* for a protocol P is a term t of sort State (i.e., $t \in T_{\Sigma_{SS_\mathcal{P}}/E_{SS_\mathcal{P}}}(\mathcal{X})_{\mathsf{State}}$) which has the form $\{S_1 \,\&\, \cdots \,\&\, S_n \,\&\, \{IK\}\}$ where $\&$ is an associative-commutative union operator with identity symbol $\varnothing$. Each element in the set is either a *strand* S_i or the *intruder knowledge* $\{IK\}$ at that state.

The *intruder knowledge* $\{IK\}$ belongs to the state and is represented as a set of facts using the comma as an associative-commutative union operator with identity element $empty$. There are two kinds of intruder facts: *positive* knowledge facts (the intruder knows m, i.e., $m{\in}\mathcal{I}$), and *negative* knowledge facts (the intruder *does not yet know* m but *will know it in a future state*, i.e., $m{\notin}\mathcal{I}$), where m is a message expression.

A *strand* [10] specifies the sequence of messages sent and received by a principal executing the protocol and is represented as a sequence of messages

$$[msg_1^{\pm}, msg_2^{\pm}, msg_3^{\pm}, \ldots, msg_{k-1}^{\pm}, msg_k^{\pm}]$$

with $msg_i^{\pm}$ either msg_i^{-} (also written $-msg_i$) representing an input message, or msg_i^{+} (also written $+msg_i$) representing an output message. Note that each msg_i is a term of a special sort Msg.

Strands are used to represent both the actions of honest principals (with a strand specified for each protocol role) and the actions of an intruder (with a strand for each action an intruder is able to perform on messages). In Maude-NPA strands evolve over time; the symbol | is used to divide past and future. That is, given a strand

$$[\, msg_1^{\pm},\ \ldots,\ msg_i^{\pm} \mid msg_{i+1}^{\pm},\ \ldots,\ msg_k^{\pm}\,]$$

, messages $msg_1^{\pm}, \ldots, msg_i^{\pm}$ are the *past messages*, and messages $msg_{i+1}^{\pm}, \ldots, msg_k^{\pm}$ are the *future messages* ($msg_{i+1}^{\pm}$ is the immediate future message). A strand $[msg_1^{\pm}, \ldots, msg_k^{\pm}]$ is shorthand for $[nil \mid msg_1^{\pm}, \ldots, msg_k^{\pm}, nil]$. An *initial state* is a state where the bar is at the beginning for all strands in the state, and the intruder knowledge has no fact of the form $m{\in}\mathcal{I}$. A *final state* is a state where the bar is at the end for all strands in the state and there is no intruder fact of the form $m{\notin}\mathcal{I}$.

Since the number of states $T_{\Sigma_{SS_\mathcal{P}}/E_{SS_\mathcal{P}}}$ is in general infinite, rather than exploring concrete protocol states $[t]_{E_{SS_\mathcal{P}}} \in T_{\Sigma_{SS_\mathcal{P}}/E_{SS_\mathcal{P}}}$ Maude-NPA explores *symbolic strand state patterns* $[t(x_1, \ldots, x_n)]_{E_{SS_\mathcal{P}}} \in T_{\Sigma_{SS_\mathcal{P}}/E_{SS_\mathcal{P}}}(\mathcal{X})$ on the free $(\Sigma_{SS_\mathcal{P}}, E_{SS_\mathcal{P}})$-algebra over a set of variables $\mathcal{X}$. In this way, a state pattern $[t(x_1, \ldots, x_n)]_{E_{SS_\mathcal{P}}}$ represents not a single concrete state but a possibly infinite set of such states, namely all the *instances* of the pattern $[t(x_1, \ldots, x_n)]_{E_{SS_\mathcal{P}}}$ where the variables $x_1, \ldots, x_n$ have been instantiated by concrete ground terms.

The semantics of Maude-NPA is expressed in terms of *rewrite rules* that describe how a protocol moves from one state to another via the intruder's interaction with it. One uses Maude-NPA to find an attack by specifying an insecure state pattern called an *attack pattern*. Maude-NPA attempts to find a path from an initial state to the attack pattern via backwards narrowing (narrowing using the rewrite rules with the orientation reversed). That is, a narrowing sequence from an initial state to an attack state is searched *in reverse* as a *backwards path* from the attack state to the initial state. Maude-NPA attempts to find paths until it can no longer form any backwards narrowing steps, at which point it terminates. If at that point it has not found an initial state, the attack pattern is judged *unreachable*. Note that Maude-NPA places *no bound on the number of sessions*, so reachability is undecidable in general. Note also that Maude-NPA does not perform any data abstraction such as a bounded number of nonces. However, the tool makes use of a number of sound and complete state space reduction techniques that help to identify unreachable and redundant states [8], and thus make termination more likely.

4. A Process Algebra for Protocols with Choice

In this section we define a process algebra that can specify protocols exhibiting choice points. Throughout the paper we refer to this process algebra as the *protocol process algebra*.

The rest of this section is organized as follows. First, in Section 4.1 we define the syntax of the protocol process algebra and state the requirements that a *well-formed process* must satisfy. Then in Section 4.2, we explain how the *protocol specifications* can be defined in this process algebra. In Section 4.3 we then define the *operational semantics* of the protocol process algebra.

4.1 Syntax of Protocol Process Algebra

In the *protocol process algebra* the behavior of both honest principals and the intruder is represented by *labeled processes*. Therefore, a protocol is specified as a set of labeled processes. Each process performs a sequence of actions, namely, sending or receiving a message, and may perform deterministic or non-deterministic choices. The protocol process algebra's syntax is parameterized by a sort Msg of messages and has the following syntax:

$$\begin{aligned}
ProcConf &::= LProc \mid ProcConf \;\&\; ProcConf \mid \varnothing \\
LProc &::= (Role, I, J)\ Proc \\
Proc &::= nil \mid +\,Msg \mid -\,Msg \mid Proc \cdot Proc \mid \\
&\qquad Proc\ ?\ Proc \mid \textit{if } Cond \textit{ then } Proc \textit{ else } Proc \\
Cond &::= Msg \;=\; Msg \mid Msg = Msg
\end{aligned}$$

- $ProcConf$ stands for a *process configuration*, that is, a set of labeled processes. The symbol & is used to denote set union for sets of labeled processes. It is associative-commutative with $\varnothing$ as its identity element.
- $LProc$ stands for a *labeled process*, that is, a process $Proc$ with a label $(Role, I, J)$. $Role$ refers to the role of the process in the protocol (e.g., initiator or responder). I is a natural number denoting the identity of the process, which distinguishes different instances(sessions) of a process specification. J indicates that the action at stage J of the process specification will be the next one to be executed, that is, the first $J-1$ actions of the process for role $Role$ have already been executed. Note that we omit I and J in the protocol specification when both I and J are 0.
- $Proc$ defines the actions that can be executed within a process. $+Msg$, and $-Msg$ respectively denote sending out or receiving a message Msg. We assume a single channel, through which all messages are sent or received by the intruder. "$Proc \cdot Proc$" denotes sequential composition of processes, where symbol $\_\cdot\_$ is associative and has the empty process *nil* as identity. "$Proc\ ?\ Proc$" denotes an explicit nondeterministic choice, whereas "*if* $Cond$ *then* $Proc$ *else* $Proc$" denotes an explicit deterministic choice, whose continuation depends on the satisfaction of the constraint $Cond$.
- $Cond$ denotes a constraint that will be evaluated in explicit deterministic choices. In this work we only consider constraints that are either equalities ($=$) or disequalities ($\neq$) between message expressions.

Let PS, QS, and RS be process configurations, and P, Q, and R be protocol processes. Our protocol syntax satisfies the following *structural axioms*:

$$PS \,\&\, QS = QS \,\&\, PS \quad (1) \qquad PS \,\&\, \varnothing = PS \quad (4)$$
$$(PS \,\&\, QS) \,\&\, RS = (PS \,\&\, QS) \,\&\, RS \quad (2) \qquad P \cdot nil = P \quad (5)$$
$$(P \cdot Q) \cdot R = P \cdot (Q \cdot R) \quad (3) \qquad nil \cdot P = P \quad (6)$$

The specification of the processes defining a protocol's behavior may contain some variables denoting information that the principal executing the process does not yet know, or that will be different in different executions. In all protocol specifications we assume three disjoint kinds of variables:

- ***fresh variables***: these are not really variables in the standard sense, but *names* for *constant values* in a data type $\mathsf{V_{fresh}}$ of unguessable values such as nonces. A *fresh variable* r is always associated with a role $ro \in Role$ in the protocol. For each protocol session i, we associate to r a unique name $r.ro.i$ for a constant in the data type $\mathsf{V_{fresh}}$. What is assumed is that if $r.ro.i \neq r'.ro'.j$ (including the case $r.ro.i \neq r.ro.j$), the values interpreting $r.ro.i$ and $r'.ro'.j$ in $\mathsf{V_{fresh}}$ are both *different* and *unguessable*. In particular, for role $ro \in Role$, the interpretation mapping $I : \{r.ro.i \mid i \in \mathbb{N}\} \rightarrow \mathsf{V_{fresh}}$ is *injective* and *random*. In our semantics, a constant $r.ro.i$ denotes its (unguessable) interpretation $I(r.ro.i) \in \mathsf{V_{fresh}}$. Throughout this paper we will denote this kind of variables as $r, r_1, r_2, \ldots$.
- ***choice variables***: variables first appearing in a *sent message* $+M$, which can be substituted by any value arbitrarily chosen from a possibly infinite domain. A choice variable indicates an *implicit non-deterministic choice*. Given a protocol with choice variables, each possible substitution of these variables denotes a possible continuation of the protocol. We always denote choice variables by uppercase letters postfixed with the symbol "?" as a subscript, e.g., $A_?, B_?, \ldots$.
- ***pattern variables***: variables first appearing in a *received message* $-M$. These variables will be instantiated when matching sent and received messages. *Implicit deterministic choices* are indicated by pattern variables, since failing to match the pattern may lead to the rejection of a message. The pattern plays the implicit role of a guard, so that, depending on the different ways of matching, the protocol can have different continuations. This kind of variables will be written as uppercase letters, e.g. $A, B, N_A, \ldots$.

Note that fresh variables are distinguished from other variables by having a specific sort Fresh. Choice variables or pattern variables can never have sort Fresh.

To guarantee the requirements on different kinds of variables that can appear in a given process, we consider only *well-formed* processes. We this notion more precise by defining a function $wf : Proc \rightarrow Bool$ checking whether a given process is well-formed. A labeled process is *well-formed* if the process in it is well-formed. A process configuration is *well-formed* if all the labeled process in it are well-formed. The definition of wf uses an auxiliary function $shVar : Proc \rightarrow VarSet$, retrieving the "shared variables" of a process, i.e., the set of variables that show up in all branches. Below we define both functions, where P, Q, and R are processes, M is a message, and T is a constraint.

$$shVar(+M \cdot P) = Var(M) \cup shVar(P)$$
$$shVar(-M \cdot P) = Var(M) \cup shVar(P)$$
$$shVar((\mathit{if}\ T\ \mathit{then}\ P\ \mathit{else}\ Q) \cdot R) = Var(T) \cup (shVar(P) \cap shVar(Q)) \cup shVar(R)$$
$$shVar((P\ ?\ Q) \cdot R) = (shVar(P) \cap shVar(Q)) \cup shVar(R)$$
$$shVar(nil) = \varnothing$$

$$wf(P \cdot +M) = wf(P) \quad \mathit{if}\ (Var(M) \cap Var(P)) \subseteq shVar(P)$$
$$wf(P \cdot -M) = wf(P) \quad \mathit{if}\ (Var(M) \cap Var(P)) \subseteq shVar(P)$$
$$wf(P \cdot (\mathit{if}\ T\ \mathit{then}\ Q\ \mathit{else}\ R)) = wf(P \cdot Q) \wedge wf(P \cdot R)$$
$$\mathit{if}\ P \neq nil\ \mathit{and}\ Q \neq nil\ \mathit{and}\ Var(T) \subseteq shVar(P)$$
$$wf(P \cdot (Q\ ?\ R)) = wf(P \cdot Q) \wedge wf(P \cdot R)\ \mathit{if}\ Q \neq nil\ \mathit{or} R \neq nil$$
$$wf(P \cdot nil) = wf(P)$$
$$wf(nil) = True.$$

Remark 1. *Note that the well-formedness property implies that if a process begins with a deterministic choice action if T then Q else R, then all variables in T must be instantiated, and thus only one branch may be taken. For this reason, it is undesirable to specify processes that begin with such an action. Furthermore, note that the well-formedness property avoids explicit choices where both possibilities are the nil process. That is, processes containing either (if T then nil else nil), or (nil ? nil), respectively.*

We illustrate the notion of well-formed process below.

Example 4.1. *The behavior of a Client initiating an instance of the handshake protocol from Example 1.1 with the Server, where the Server may or may not request the Client to authenticate itself, may be specified by the well-formed process shown below:*

$$\begin{aligned}
(Client)\ & +(hs; n(C_?, r_1); G_?; gen(G_?); keyG(G_?, C_?, r_2)) \cdot \\
& -(hs; N; G_?; gen(G_?); E; Z(AReq, G_?, E, C_?, r_1, S, HM)) \cdot \\
& \mathit{if}\ (AReq = authreq) \\
& \mathit{then} \\
& +(e(keyE(G_?, E, C_?, r_1), \\
& \quad sig(C, W(HM, AReq, S_?, G_?, E, C_?, r_1)); \\
& \quad\ mac(keyE(G_?, E, , C_?, r_1), \\
& \quad\quad W(HM, AReq, S, G_?, E, C_?, r_1)))) \cdot \\
& \mathit{else} \\
& +(e(keyE(G_?, E, C_?, r_2), \\
& \quad mac(keyE(G_?, E, C_?, r_2), \\
& \quad\ W(HM, AReq, S, G_?, E, C_?, r_1))))
\end{aligned}$$

where $KeyG$, Z and W are macros used to construct messages sent in the protocol. The variables $C_?$ and $G_?$ are choice variables denoting the client and Diffie-Hellman group respectively, and the variables r_1 and r_2 are fresh variables. All other variables are pattern variables. In particular, the variable $AReq$ is a pattern variable which can be instantiated to either $authreq$ or $noauthreq$. The Client makes a deterministic choice whether or not to sign its next message with its digital signature, depending on which value of $AReq$ it receives.

Example 4.2. *The behavior of a Server who may or may not request a retry from a Client in an instance of the handshake protocol from Example 1.1 may be specified as follows:*

$$\begin{aligned}
(Server) : & -(hs; N; G; gen(G); E) \cdot \\
& (((+(hs; retry) \cdot -(hs; N'; G'; gen(G'); E') \cdot \\
& \quad +(hs; n(S_?, r1); G'; gen(G'); keyG(G', S_?, r_2); \\
& \quad\ Z(AReq_?, G', E', S, r_2, S_?, HM))) \\
& ? \\
& (+(hs; n(S_?, r1); G; gen(G); keyG(G, S, r_2); \\
& \quad Z(AReq_?, G, E, S, r_2, S_?, HM)))))
\end{aligned}$$

In this case the server nondeterministically chooses to request or not to request a retry. In the case of a retry it waits for the retry message from the client, and then proceeds with the handshake message

using the new key information from the client. In the case when it does not request a retry, it sends the handshake message immediately after receiving the client's Hello message. The variable r_2 is a fresh variable, while $S_?$ and $AReq_?$ are choice variables. $S_?$ denotes the name of the server, and $AReq_?$ is nondeterministically instantiated to $authreq$ or $noauthreq$.

Finally, we give an example of a process that does not satisfy the well-formedness property.

Example 4.3.

$$\begin{aligned}(Resp) - (pk(B, A; NA))\cdot \\ (+(pk(A, 1; n(B, r))) \ ? \ + (pk(A, 2))) \cdot \\ + (pk(C_?, n(B, r)))\end{aligned}$$

The problem with this process is the fresh variable r appearing in message $+(pk(C_?, n(B, r)))$, since $r \notin shVar(-(pk(B, A; NA)) \cdot (+(pk(A, 1; n(B, r))) \ ? \ + (pk(A, 2))))$ (more specifically, because it does not appear in message $+(pk(A, 2)))$, but $r \in Var(-(pk(B, A; NA)) \cdot (+(pk(A, 1; n(B, r))) \ ? \ + (pk(A, 2))))$.

4.2 Protocol Specification in Process Algebra

Given a protocol $\mathcal{P}$, we define its specification in the protocol process algebra, written $\mathcal{P}_{PA}$, as a tuple of the form $\mathcal{P}_{PA} = ((\Sigma_{PA_\mathcal{P}}, E_{PA_\mathcal{P}}), P_{PA})$, where P_{PA} is a term denoting a *well-formed* process configuration representing the behavior of the honest principals as well as the capabilities of the attacker. That is, $P_{PA} = (ro_1)P_1 \ \& \ \ldots \ \& \ (ro_i)P_i$, where each ro_k, $1 \leqslant k \leqslant i$, is either the role of an honest principal or identifies one of the capabilities of the attacker. P_{PA} cannot contain two processes with the same label, i.e., the behavior of each honest principal, and each attacker capability are represented by a *unique* process. $E_{PA_\mathcal{P}} = E_\mathcal{P} \cup E_{PA}$ is a set of equations with $E_\mathcal{P}$ denoting the protocol's cryptographic properties and E_{PA} denoting the properties of process constructors. The set of equations $E_\mathcal{P}$ may vary depending on different protocols, but the set of equations E_{PA} is always the same for all protocols. $\Sigma_{PA_\mathcal{P}} = \Sigma_\mathcal{P} \cup \Sigma_{PA}$ is the signature defining the sorts and function symbols as follows:

- $\Sigma_\mathcal{P}$ is an order-sorted signature defining the sorts and function symbols for the messages that can be exchanged in protocol $\mathcal{P}$. However, independently of protocol $\mathcal{P}$, $\Sigma_\mathcal{P}$ must always have a sort Msg as the top sort in one of its connected components. We call a sort S a *data sort* iff it is either a subsort of Msg, or there is a message constructor $c : \mathsf{S_1}...\mathsf{S}...\mathsf{S_n} \rightarrow \mathsf{S'}$, with $\mathsf{S'}$ a subsort of Msg. The specific sort Fresh for fresh variables is an example of *data sort*. Choice and pattern variables have sort Msg or any of its subsorts.
- Σ_{PA} is an order-sorted signature defining the sorts and function symbols of the *process algebra infrastructure*. Σ_{PA} corresponds exactly to the BNF definition of the protocol process algebra's syntax in Section 4.1. Although it has a sort Msg for messages, it leaves this sort totally unspecified, so that different protocols $\mathcal{P}$ may use completely different message constructors and may satisfy different equational properties $E_\mathcal{P}$. Therefore, Σ_{PA} will be the same signature for any protocol specified in the process algebra. More specifically, Σ_{PA} contains the sorts for process configurations ($\mathsf{ProcConf}$), labeled processes (LProc), processes (Proc), constraints (Cond), and messages(Msg), as well as the subsort relations $\mathsf{LProc} < \mathsf{ProcConf}$. Furthermore, the function symbols in Σ_{PA} are also defined according to the BNF definition.

Therefore, the syntax $\Sigma_{PA_\mathcal{P}}$ of processes for $\mathcal{P}$ will be in the union signature $\Sigma_{PA} \cup \Sigma_\mathcal{P}$, consisting of the protocol-specific syntax $\Sigma_\mathcal{P}$, and the generic process syntax Σ_{PA} through the shared sort Msg.

4.3 Operational Semantics of the Protocol Process Algebra

Given a protocol $\mathcal{P}$, a *state* of $\mathcal{P}$ consists of a set of (possibly partially executed) *labeled processes*, and a set of terms in the intruder's knowledge $\{IK\}$. That is, a state is a term of the form

$$\{LP_1 \ \& \ \cdots \ \& \ LP_n \mid \{IK\}\}$$

Given a state St of the form shown above, we abuse notation and write $LP_k \in St$ if LP_k is a labeled process in the set $LP_1 \ \& \ \cdots \ \& \ LP_n$.

The intruder knowledge IK models the *single* channel through which all messages are sent and received. We consider an active attacker who has complete control of the channel, i.e, can read, alter, redirect, and delete traffic as well as create its own messages by means of intruder processes.

State changes are defined by a set $R_{PA_\mathcal{P}}$ of *rewrite rules*, such that the rewrite theory $(\Sigma_{PA_\mathcal{P}+State}, E_{PA_\mathcal{P}}, R_{PA_\mathcal{P}})$ characterizes the behavior of protocol $\mathcal{P}$, where $\Sigma_{PA_\mathcal{P}+State}$ extends $\Sigma_{PA_\mathcal{P}}$ by adding state constructor symbols. We assume that a protocol's execution begins with an empty state, i.e., a state with an empty set of labeled processes, and an empty intruder knowledge. That is, the initial state is always of the form shown below:

$$\{\varnothing \mid \{empty\}\}$$

Each transition rule in $R_{PA_\mathcal{P}}$ is labeled with a tuple of the form (ro, i, j, a, n), where:

- ro is the role of the labeled process being executed in the transition.
- i denotes the identifier of the labeled process being executed in the transition. Since there can be more than one process instance of the same role in a process state, i is used to distinguish different instances, i.e., ro and i together uniquely identify a process in a state.
- j denotes the process' step number since its beginning.
- a is a ground term identifying the action that is being performed in the transition. It has different possible values: "$+m$" or "$-m$" if the message m was sent (and added to the intruder's knowledge) or received, respectively; "m" if the message m was sent but did not increase the intruder's knowledge, "?" if the transition performs an explicit non-deterministic choice, or "T" if the transition performs a explicit deterministic choice.
- n is a number that, if the action that is being executed is an explicit choice, indicates which branch has been chosen as the process continuation. In this case n takes the value of either 1 or 2. If the transition does not perform any explicit choice, then $n = 0$.

Below we describe the set of transition rules that define a protocol's execution in the protocol process algebra, that is, the set of rules $R_{PA_\mathcal{P}}$. Note that in the transition rules shown below, PS denotes the rest of labeled processes of the state (which can be $\varnothing$).

- The action of *sending a message* is represented by the two transition rules below. Since we assume the intruder has complete control of the network, it can learn any message sent by other principals. Rule (PA++) denotes the case in which the sent message is added to the intruder's knowledge. Note that this rule can only be applied if the intruder has not already learnt that message. Rule (PA+) denotes the case in which the intruder chooses not to learn the message, i.e., the intruder's knowledge is not modified, and, thus, no condition needs to be checked. Since choice variables denote messages that are nondeterministically chosen, all (possibly infinitely many) admissible ground substitutions for the choice variables are possible behaviors.

$$\{(ro,i,j)\ (+M\cdot P)\ \&\ PS \mid \{IK\}\}$$
$$\longrightarrow_{(ro,i,j,+M\sigma,0)} \{(ro,i,j+1)\ P\sigma\ \&\ PS \mid \{M\sigma{\in}\mathcal{I}, IK\}\}$$
if $(M\sigma{\in}\mathcal{I}) \notin IK$

where σ *is a ground substitution binding choice variables in* M (PA++)

$$\{(ro,i,j)\ (+M\cdot P)\ \&\ PS \mid \{IK\}\}$$
$$\longrightarrow_{(ro,i,j,M\sigma,0)} \{(ro,i,j+1)\ P\sigma\ \&\ PS \mid \{IK\}\}$$
where σ *is a ground substitution binding choice variables in* M (PA+)

- As shown in the rule below, a process can *receive a message* matching a pattern M if there is a message M' in the intruder's knowledge, i.e. a message previously sent either by some honest principal or by some intruder process, that matches the pattern message M. After receiving this message the process will continue with its variables instantiated by the matching substitution, which takes place modulo the equations $E_{\mathcal{P}}$. Note that the intruder can "delete" a message via choosing not to learn it (executing Rule PA+ instead of Rule PA++) or not to deliver it (failing to execute Rule PA-).

$$\{(ro,i,j)\ (-M\cdot P)\ \&\ PS \mid \{M'{\in}\mathcal{I}, IK\}\}$$
$$\longrightarrow_{(ro,i,j,-M\sigma,0)} \{(ro,i,j+1)\ P\sigma\ \&\ PS \mid \{M'{\in}\mathcal{I}, IK\}\}$$
if $M' =_{E_{\mathcal{P}}} M\sigma$ (PA-)

- The two transition rules shown below define the operational semantics of *explicit deterministic choice*s. That is, the operational semantics of an *if* T *then* P *else* Q expression. More specifically, rule (PAif1) describes the *then* case, i.e., if the constraint T is satisfied, the process will continue as P. Rule (PAif2) describes the *else* case, that is, if the constraint T is *not* satisfied, the process will continue as Q. Note that, since we only consider well-formed processes, these transition rules will only be applied if $j \geqslant 1$. Note also that since T has been fully substituted by the time the if-then-else is executed, and the constraints that we considered in this paper are of the form $m \neq_{E_{\mathcal{P}}} m'$ or $m =_{E_{\mathcal{P}}} m'$, the satisfiability of T can be checked by checking whether the corresponding ground equality or disequality holds.

$$\{(ro,i,j)\ ((\textit{if}\ T\ \textit{then}\ P\ \textit{else}\ Q)\ \cdot R)\ \&\ PS \mid \{IK\}\}$$
$$\longrightarrow_{(ro,i,j,T,1)} \{(ro,i,j+1)\ (P\cdot R)\ \&\ PS \mid \{IK\}\}\ \ if\ T$$
(PAif1)

$$\{(ro,i,j)\ ((\textit{if}\ T\ \textit{then}\ P\ \textit{else}\ Q)\ \cdot R)\ \&\ PS \mid \{IK\}\}$$
$$\longrightarrow_{(ro,i,j,T,2)} \{(ro,i,j+1)\ (Q\cdot R)\ \&\ PS \mid \{IK\}\}\ \ if\ \neg T$$
(PAif2)

- The two transition rules below define the semantics of *explicit non-deterministic choice* $P\ ?\ Q$. In this case, the process can continue either as P, denoted by rule (PA?1), or as Q, denoted by rule (PA?2). Note that this decision is made non-deterministically.

$$\{(ro,i,j)\ ((P\ ?\ Q)\cdot R)\ \&\ PS \mid \{IK\}\}$$
$$\longrightarrow_{(ro,i,j,?,1)} \{(ro,i,j+1)\ (P\cdot R)\ \&\ PS \mid \{IK\}\}$$
(PA?1)

$$\{(ro,i,j)\ ((P\ ?\ Q)\cdot R)\ \&\ PS \mid \{IK\}\}$$
$$\longrightarrow_{(ro,i,j,?,2)} \{(ro,i,j+1)(Q\cdot R)\ \&\ PS \mid \{IK\}\}$$
(PA?2)

- The transition rules shown below describe the *introduction of a new process* from the specification into the state, which allows us to support an unbounded session model. Recall that fresh variables are associated with a role and an identifier. Therefore, whenever a new process is introduced: (a) the largest process identifier (i) will be increased by 1, and (b) new names will be assigned to the fresh variables in the new process. The function *MaxProcId*(PS, ro) in the transition rule below is used to get the largest process identifier (i) of role ro in the process configuration PS. The substitution $\rho_{ro,i+1}$ in the transition rule below takes a labeled process and assigns new names to the fresh variables according to the label. More specifically, $(ro, i+1, 1)\ P_k(r_1,\ldots,r_n)\rho_{ro,i+1} = (ro, i+1, 1)\ P_k(r_1,\ldots,r_n)\ \{r_1 \mapsto r_1.ro.i+1, \ldots, r_n \mapsto r_n.ro.i+1\}$. In a process state, a role name together with an identifier uniquely identifies a process. Therefore, there is a unique subset of fresh names for each process in the state. In the rest of this paper we will refer to this kind of substitutions as *fresh substitutions*.

$$\left\{\begin{array}{l}\forall\ (ro)\ P_k \in P_{PA}\\ \{PS \mid \{IK\}\}\\ \longrightarrow_{(ro,i+1,1,A,Num)} \{(ro,i+1,2)\ P'_k\ \&\ PS \mid \{IK'\}\}\\ \mathsf{IF}\ \{(ro,i+1,1)\ P_k\rho_{ro,i+1} \mid \{IK\}\}\\ \longrightarrow_{(ro,i+1,1,A,Num)} \{(ro,i+1,2)\ P'_k \mid \{IK'\}\}\\ \textit{where}\ \rho_{ro,i+1}\ \textit{is a fresh substitution},\\ i = \textit{MaxProcId}(PS, ro)\end{array}\right\}$$
(PA&)

Note that A denotes the action of the state transition, and can be of any of the forms explained above. The function *MaxProcId* is defined as below:

$$\textit{MaxProcId}(\emptyset, ro) = 0$$
$$\textit{MaxProcId}((ro,i,j)P\&PS, ro) = max(\textit{MaxProcId}(PS, ro), i)$$
$$\textit{MaxProcId}((ro',i,j)P\&PS, ro) = \textit{MaxProcId}(PS, ro) \quad \textit{if}\ ro \neq ro'$$

where PS denotes a process configuration, P denotes a process, and ro, ro' denote role names.

Therefore, the behavior of a protocol in the process algebra is defined by the set of transition rules $R_{PA_{\mathcal{P}}} = \{$(PA++), (PA+), (PA-), (PAif1), (PAif2), (PA?1), (PA?2)$\} \cup$ (PA&)

5. Constrained Protocol Strands with Choice

To specify and analyze protocols with choices in Maude-NPA, in this section we extend Maude-NPA's strand notation by adding new symbols to support explicit choices. We refer to the strands in this extended syntax as *constrained protocol strands*.

In Section 5.1 we describe the syntax for constrained protocol strands. Then, in Section 5.2 we define a mapping from a protocol specification in the protocol process algebra, as described in Section 4.2, to a specification based on constrained protocol strands.

5.1 Constrained Protocol Strands Syntax

In this section we extend Maude-NPA's syntax by adding *constrained messages*, which are terms of the form $\{Cstr, Num\}$, where $Cstr$ is a constraint, and Num is a natural number that identifies the continuation of the protocol's execution, among the two possibilities after an explicit choice point. More specifically, we extend the Σ_{SS} part of the signature $\Sigma_{SS_{\mathcal{P}}}$ of the Maude-NPA's syntax we defined in Section 3 as follows:

- A new sort Cstr represents the constraints allowed in constrained messages, that are, the constant "?" for explicit non-deterministic choice, and equality constructor ("_ = _"), and

disequality constructor ("_ ≠ _") among message expressions for explicit deterministic choice:

$$? : \rightarrow \mathsf{Cstr} . \qquad \_ = \_ : \mathsf{Msg\ Msg} \rightarrow \mathsf{Cstr} .$$
$$\_ \neq \_ : \mathsf{Msg\ Msg} \rightarrow \mathsf{Cstr} .$$

- A new sort CstrMsg for constrained messages, such that CstrMsg < SMsg, where SMsg is an existing Maude-NPA sort denoting signed messages (i.e., messages with + or -). Therefore, now a strand is a sequence of output, input and constrained messages.
- A new operator $\{\_,\_\}$ constructs constraint messages as follows:

$$\{\_,\_\} : \mathsf{Cstr\ Nat} \rightarrow \mathsf{CstrMsg} .$$

We will refer to this extended signature as $\Sigma_{CstrSS_{\mathcal{P}}}$. Note that the protocol signature $\Sigma_{\mathcal{P}}$ is contained in $\Sigma_{SS_{\mathcal{P}}}$, therefore in $\Sigma_{CstrSS_{\mathcal{P}}}$. Furthermore, in the constrained semantics we allow each honest principal or intruder capability strand to be *labeled* by the "role" of that strand in the protocol (e.g. `(Client)` or `(Server)`). Therefore, strands are now terms of the form $(ro, i)[u_1, \ldots, u_n]$, where ro denotes the role of the strand in the protocol, i is a unique identifier distinguishing different instances of the strands of the same role, and each u_i can be a sent or received message, i.e., a term of the form $M^{\pm}$, or a constraint message of the form $\{Cstr, Num\}$. We often omit i, or both ro and i for clarity when they are not relevant.

5.2 Protocol Specification using Constrained Protocol Strands

The behavior of a protocol involving choices can be specified using the syntax presented in Section 5.1 as described below.

Definition 1 (Protocol specification)**.** *Given a protocol $\mathcal{P}$, we define its specification by means of constrained protocol strands, written $\mathcal{P}_{CstrSS}$, as a tuple of the form $\mathcal{P}_{CstrSS} = ((\Sigma_{CstrSS_{\mathcal{P}}}, E_{SS_{\mathcal{P}}}), P_{CstrSS})$, where $\Sigma_{CstrSS_{\mathcal{P}}}$ is the protocol's signature (see Section 5.1), and $E_{SS_{\mathcal{P}}} = E_{\mathcal{P}} \cup E_{SS}$ is a set of equations as we defined in Section 3, where $E_{\mathcal{P}}$ denotes the protocol's cryptographic properties and E_{SS} denotes the protocol-independent properties of constructors of strands. That is, the set of equations $E_{\mathcal{P}}$ may vary depending on different protocols, but the set of equations E_{SS} is always the same for all protocols. P_{CstrSS} is a set of constrained protocol strands as defined in Section 5.1, representing the behavior of the honest principals as well as the capabilities of the attacker. That is, P_{CstrSS} is a set of labeled strands of the form: $P_{CstrSS} = \{(ro_1)[u_{1,1}, \ldots, u_{1,n_1}]\ \&\ \ldots\ \&\ (ro_m)[u_{m,1}, \ldots, u_{m,n_m}]\}$, where, for each ro_k such that $1 \leqslant k \leqslant i$, ro_k is either the role of an honest principal, or identifies one of the capabilities of the attacker. We note that P_{CstrSS}* may contain several strands with the same label*, each defining one of the possible paths of such a principal.*

The protocol specification described above can be obtained by *transforming* a specification in the process algebra of Section 4.2 as follows. Given a protocol $\mathcal{P}$, its specification in the process algebra P_{PA}, consists of a set of *well-formed* labeled processes. We transform a term denoting a set of labeled processes into a term denoting a set of constrained protocol strands by the mapping *toCstrSS*. The intuitive idea is that, since our process contains no recursion, each process can be "deconstructed" as a set of constrained protocol strands, where each such strand represent a possible execution path of the process.

The mapping *toCstrSS* is defined in Definition 2 below.

Definition 2 (Mapping *toCstrSS*)**.** *Given a labeled process LP and a process configuration LPS, we define the mapping toCstrSS : $\mathcal{T}_{\Sigma_{PA_{\mathcal{P}}}}(\mathcal{X}) \rightarrow \mathcal{T}_{\Sigma_{CstrSS_{\mathcal{P}}}}(\mathcal{X})$ as:*

$$\mathit{toCstrSS}(LP\ \&\ LPS) = \mathit{toCstrSS}^*(LP, nil)\ \&\ \mathit{toCstrSS}(LPS)$$
$$\mathit{toCstrSS}(\varnothing) = \varnothing$$

where $\varnothing$ is the empty set of strands. toCstrSS is an auxiliary mapping that maps a term denoting a labeled process to a term that denotes a set of constrained protocol strands. It takes two arguments: a labeled process, and a temporary store that keeps a sequence of messages. More specifically, toCstrSS* : $\mathcal{T}_{\Sigma_{PA_{\mathcal{P}}}}(\mathcal{X}) \times \mathcal{T}_{\Sigma_{CstrSS_{\mathcal{P}}}}(\mathcal{X}) \rightarrow \mathcal{T}_{\Sigma_{CstrSS_{\mathcal{P}}}}(\mathcal{X})$ is defined as follows:*

$$\begin{aligned}
&\mathit{toCstrSS}^*((ro,i,j)\ nil, L) = (ro,i)\ [\ L\ ]\\
&\mathit{toCstrSS}^*((ro,i,j)\ +M\ .\ P, L)\\
&\qquad = \mathit{toCstrSS}^*((ro,i,j)\ P, (L,\ +M))\\
&\mathit{toCstrSS}^*((ro,i,j)\ -M\ .\ P,\ L)\\
&\qquad = \mathit{toCstrSS}^*((ro,i,j)\ P, (L,\ -M))\\
&\mathit{toCstrSS}^*((ro,i,j)\ (\mathit{if}\ T\ \mathit{then}\ P\ \mathit{else}\ Q)\ .\ R, L)\\
&\qquad = \mathit{toCstrSS}^*((ro,i,j)\ P\ .\ R, (L,\ \{T,1\}))\ \&\\
&\qquad\quad \mathit{toCstrSS}^*((ro,i,j)\ Q\ .\ R, (L,\ \{\neg T,2\}))\\
&\mathit{toCstrSS}^*((ro,i,j)\ (P\ ?\ Q)\ .\ R, L)\\
&\qquad = \mathit{toCstrSS}^*((ro,i,j)\ P\ .\ R, (L,\ \{?,1\}))\ \&\\
&\qquad\quad \mathit{toCstrSS}^*((ro,i,j)\ Q\ .\ R, (L, \{?,2\}))
\end{aligned}$$

where P, Q, and R denote processes, M is a message, T is a constraint, and L denotes a list of messages, i.e., input, output or constraint messages.

Note that toCstrSS does not modify output and input messages, since messages are actually terms in $\mathcal{T}_{\Sigma_{\mathcal{P}}/E_{\mathcal{P}}}(\mathcal{X})$ in both the protocol process algebra, and the constrained forwards semantics. toCstrSS can be used both as a map between specifications, and as a map from process configurations and strand sets appearing in states.

We illustrate *toCstrSS* with the example below.

Example 5.1. *If we apply the mapping toCstrSS to the process in Example 4.2 we obtain the following term which denotes a set of strands:*

$$\begin{aligned}
&(\mathit{Server})\ [\ \{?,1\},\ -(hs; N; G; gen(G); E),\\
&\qquad + (hs; retry),\\
&\qquad - (hs; N'; G'; gen(G'); E')),\\
&\qquad + (hs; n(S_?, r1); G'; gen(G'); keyG(G', S_?, r_2);\\
&\qquad\quad Z(AReq_?, G', E', S, r_2, S_?, HM))]\ \&\\
&(\mathit{Server})\ [\ \{?,2\}, -(hs; N; G; gen(G); E),\\
&\qquad + (hs; n(S_?, r1); G; gen(G); keyG(G, S, r_2);\\
&\qquad\quad Z(AReq_?, G, E, S, r_2, S_?, HM))]
\end{aligned}$$

A protocol specification in the protocol process algebra can then be transformed into a specification of that protocol in the constrained protocol strands described below using *toCstrSS*.

Definition 3 (Specification mapping)**.** *Given a protocol $\mathcal{P}$ and its protocol process algebra specification $\mathcal{P}_{PA} = ((\Sigma_{PA_{\mathcal{P}}}, E_{\mathcal{P}} \cup E_{PA}), P_{PA})$, where*

$$P_{PA} = (ro_1)P_1\ \&\ \ldots\&(ro_n)P_n$$

its specification by means of constrained protocol strands is $\mathcal{P}_{CstrSS} = ((\Sigma_{CstrSS_{\mathcal{P}}}, E_{\mathcal{P}} \cup E_{SS}), P_{CstrSS})$ with $P_{CstrSS} = \mathit{toCstrSS}(P_{PA})$.

6. Constrained Forwards Strand Semantics

In this section we extend Maude-NPA's rewriting-based forwards semantics in [7] by adding new transition rules for constrained messages. We refer to this extended forwards semantics as *constrained forwards strand semantics*. We show that the process algebra semantics and the constrained forwards strand semantics are label bisimilar. Therefore, protocols exhibiting choices can be specified and executed in an equivalent way in both semantics.

In constrained forwards strand semantics, state changes are defined by a set $R_{CstrF_{\mathcal{P}}}$ of *rewrite rules*, such that the rewrite theory $(\Sigma_{CstrSS_{\mathcal{P}}}, E_{SS_{\mathcal{P}}}, R_{CstrF_{\mathcal{P}}})$ characterizes the behaviors of protocol $\mathcal{P}$.

The set of transition rules $R_{CstrF_{\mathcal{P}}}$ is an extension of the transition rules $R_{F_{\mathcal{P}}}$ in [7]. The transition rules are generated from the protocol specification. A *state* consists of a multiset of partially executed strands and a set of terms denoting the intruder's knowledge. The main differences between the sets $R_{CstrF_{\mathcal{P}}}$ and $R_{F_{\mathcal{P}}}$ are: (i) new transition rules are added in $R_{CstrF_{\mathcal{P}}}$ to appropriately deal with constraint messages, (ii) strands are labeled with the role name, together with the identifier for distinguishing different instances, as explained in Section 5.1, (iii) transitions are also labeled, similarly as in the protocol process algebra, (iv) the global counter for generating fresh variables is deleted from the state, instead, special unique names are assigned to fresh variable, which simplifies our notation.

In the constrained forwards strand semantics we label each transition rule similarly as in Section 4.3, that is, using labels of the form (ro, i, j, a, n), where ro, i, a, and n are as explained in Section 4.3, and j in this case is the position of the message that is being exchanged in the state transition. Also, similar to Section 4.3, for transitions that send out messages containing choice variables, all (possibly infinitely many) admissible ground substitutions for the choice variables are possible behaviors. A similar mechanism for distinguishing different fresh variables is used as that explained in Section 4.3. Since messages are introduced into strands in the state incrementally, we instantiate the fresh variables incrementally as well. Recall that fresh variables always first show up in a sent message. Therefore, each time a sent message is introduced into a strand in the state, we assign new names to the fresh variables in the message being introduced. The function *MaxStrId* for getting the max identifier for a constrained strand of a certain role is similar to *MaxProcId* in Section 4.3.

Since now messages in a strand can be sent or received messages, i.e. terms of the form m^+ or m^-, as well as constraint messages $\{Cstr, Num\}$, we represent them in the rules below simply as terms of the form u_i when their exact form is not relevant. We will use the precise form of the message when disambiguation is needed.

Before explaining the new transition rules for constraint messages, we show how the transition rules in [7] are labeled.

$$\left\{\begin{array}{l}\forall\,(ro)\,[u_1,\ldots,u_{j-1},u_j^+,u_{j+1},\ldots,u_n]\in P_{CstrSS}\,\wedge\, j{>}1:\\ \{SS\,\&\,\{IK\}\,\&\,(ro,i)\,[u_1,\ldots,u_{j-1}]\}\\ \quad\rightarrow_{(ro,i,j,(u_j\rho_{ro,i}\sigma)^+,0)}\\ \{SS\,\&\,\{u_j\rho_{ro,i}\sigma{\in}\mathcal{I},IK\}\,\&\,(ro,i)\,[u_1,\ldots,u_{j-1},(u_j\rho_{ro,i}\sigma)^+]\}\\ \qquad\mathsf{IF}\ (u_j\rho_{ro,i}\sigma{\in}\mathcal{I})\notin IK\\ \textit{where } \sigma \textit{ is a ground substitution binding choice variables in } u_j,\\ \rho_{ro,i}=\{r_1\mapsto r_1.ro.i,\ldots,r_n\mapsto r_n.ro.i\} \textit{ is a fresh substitution.}\end{array}\right\} \quad \text{(F++)}$$

$$\left\{\begin{array}{l}\forall\,(ro)\,[u_1,\ldots,u_{j-1},u_j^+,u_{j+1},\ldots,u_n]\in P_{CstrSS}\,\wedge\, j{>}1:\\ \{SS\,\&\,\{IK\}\,\&\,(ro,i)\,[u_1,\ldots,u_{j-1}]\}\\ \quad\rightarrow_{(ro,i,j,u_j\rho_{ro,i}\sigma,0)}\\ \{SS\,\&\,\{IK\}\,\&\,(ro,i)\,[u_1,\ldots,u_{j-1},(u_j\rho_{ro,i}\sigma)^+]\}\\ \textit{where } \sigma \textit{ is a ground substitution binding choice variables in } u_j,\\ \rho_{ro,i}=\{r_1\mapsto r_1.ro.i,\ldots,r_n\mapsto r_n.ro.i\} \textit{ is a fresh substitution.}\end{array}\right\} \quad \text{(F+)}$$

$$\left\{\begin{array}{l}\forall\,(ro)\,[u_1^+,\ldots,u_n]\in P_{CstrSS}:\\ \{SS\,\&\,\{IK\}\}\rightarrow_{(ro,i+1,j,(u_1\rho_{ro,i+1}\sigma)^+,0)}\\ \{SS\,\&\,(ro,i+1)\,[(u_1\rho_{ro,i+1}\sigma)^+]\,\&\,\{u_1\rho_{ro,i+1}\sigma{\in}\mathcal{I},IK\}\}\\ \qquad\mathsf{IF}\ (u_1\rho_{ro,i+1}\sigma{\in}\mathcal{I})\notin IK\\ \textit{where } \sigma \textit{ is a ground substitution binding choice variables in } u_1,\\ \rho_{ro,i+1}=\{r_1\mapsto r_1.ro.i+1,\ldots,r_n\mapsto r_n.ro.i+1\}\\ \textit{is a fresh substitution, } i=\textit{MaxStrId}(SS,ro).\end{array}\right\} \quad \text{(F++\&)}$$

$$\left\{\begin{array}{l}\forall\,(ro)\,[u_1^+,\ldots,u_n]\in P_{CstrSS}:\\ \{SS\,\&\,\{IK\}\}\rightarrow_{(ro,i+1,j,u_1\rho_{ro,i+1}\sigma,0)}\\ \quad\{SS\,\&\,(ro,i+1)\,[(u_1\rho_{ro,i+1}\sigma)^+]\,\&\,\{IK\}\}\\ \textit{where } \sigma \textit{ is a ground substitution binding choice variables in } u_1,\\ \rho_{ro,i+1}=\{r_1\mapsto r_1.ro.i+1,\ldots,r_n\mapsto r_n.ro.i+1\}\\ \textit{is a fresh substitution, } i=\textit{MaxStrId}(SS,ro).\end{array}\right\} \quad \text{(F+\&)}$$

$$\left\{\begin{array}{l}\forall\,(ro)\,[u_1,\ldots,u_{j-1},u_j^-,u_{j+1},\ldots,u_n]\in P_{CstrSS}\,\wedge\, j>1:\\ \{SS\,\&\{u_j{\in}\mathcal{I},IK\}\,\&\,(ro,i)\,[u_1,\ldots,u_{j-1}]\}\\ \rightarrow_{(ro,i,j,u_j^-,0)}\\ \{SS\,\&\,\{u_j{\in}\mathcal{I},IK\}\,\&\,(ro,i)\,[u_1,\ldots,u_{j-1},u_j^-]\}\end{array}\right\} \quad \text{(F-)}$$

$$\left\{\begin{array}{l}\forall(ro)\,[u_1^-,u_2,\ldots,u_n]\in P_{CstrSS}:\\ \{SS\,\&\,\{u_1{\in}\mathcal{I},IK\}\}\rightarrow_{(ro,i+1,1,u_1^-,0)}\\ \{SS\,\&\,(ro,i+1)\,[u_1^-]\,\&\,\{u_1{\in}\mathcal{I},IK\}\}\\ \textit{where } i=\textit{MaxStrId}(SS,ro)\end{array}\right\} \quad \text{(F-\&)}$$

The constrained forwards strand semantics extends Maude-NPA's forwards semantics in [7] by adding transition rules to handle constraint messages, i.e, messages of the form $\{Cstr, Num\}$, where Num can be either 1 or 2. First, we add the two transition rules below for the cases when such a constrained message comes from explicit choices. Note that, as a consequence of the well-formedness, the constraints introduce no new variables, and since the constraints that we consider are of the form $m \neq_{E_{\mathcal{P}}} m'$ or $m =_{E_{\mathcal{P}}} m'$, the satisfiability of $Cstr$ can be checked by checking whether the corresponding ground equality or disequality holds.

$$\left\{\begin{array}{l}\forall\,(ro)\,[u_1,\ldots,u_{j-1},\{Cstr,Num\},u_{j+1},\ldots,u_n]\in P_{CstrSS}\\ \wedge j>1:\\ \{SS\,\&\{IK\}\,\&\,(ro,i)\,[u_1,\ldots,u_{j-1}]\}\\ \rightarrow_{(ro,i,j,T,Num)}\\ \{SS\,\&\,\{IK\}\,\&\,(ro,i)\,[u_1,\ldots,u_{j-1},\{Cstr,Num\}]\}\\ \qquad\mathsf{IF}\ Cstr\end{array}\right\} \quad \text{(Fif)}$$

$$\left\{\begin{array}{l}\forall\,(ro)\,[u_1,\ldots,u_{j-1},\{?,Num\},u_{j+1},\ldots,u_n]\in P_{CstrSS}\\ \wedge j>1:\\ \{SS\,\&\{IK\}\,\&\,(ro,i)\,[u_1,\ldots,u_{j-1}]\}\\ \rightarrow_{(ro,i,j,?,Num)}\\ \{SS\,\&\,\{IK\}\,\&\,(ro,i)\,[u_1,\ldots,u_{j-1},\{?,Num\}]\}\end{array}\right\} \quad \text{(F?)}$$

The following set of transition rules adds to the state a new strand whose first message is a constraint message of the form $\{?, Num\}$:

$$\left\{\begin{array}{l}\forall\,(ro)\,[\,\{?, Num\}, u_2, \ldots, u_n] \in P_{CstrSS}:\\ \{SS\,\&\,\{IK\}\} \rightarrow_{(ro, i+1, 1, ?, Num)}\\ \{SS\,\&\,(ro, i+1)\,[\,\{?, Num\}\,]\,\&\,\{IK\}\}\\ \quad \textit{where } i = \textit{MaxStrId}(SS, ro)\end{array}\right\} \qquad \text{(F?\&)}$$

Definition 4. *Let $\mathcal{P}$ be a protocol with signature $\Sigma_{CstrSS_\mathcal{P}}$ and equational theory $E_{SS_\mathcal{P}}$. We define the* constrained forwards rewrite theory characterizing *$\mathcal{P}$ as $(\Sigma_{CstrSS_\mathcal{P}}, E_{SS_\mathcal{P}}, R_{CstrF_\mathcal{P}})$ where $R_{CstrF_\mathcal{P}}$ = (F++) ∪ (F+) ∪ (F++&) ∪ (F+&) ∪ (F-) ∪ (F-&) ∪ (Fif) ∪ (F?) ∪ (F?&).*

6.1 Bisimulation between Constrained Forwards Strand Semantics and Process Algebra Semantics

In this section we show that the process algebra semantics and the constrained forwards strand semantics are label bisimilar. We first define PA-State and FW-State, the respective notions of state in each semantics.

Definition 5 (PA-State)**.** *Given a protocol $\mathcal{P}$, a* PA-State *of $\mathcal{P}$ is a state in the protocol process algebra semantics that is* reachable *from the initial state. The initial PA-State is $P_{init} = \{\varnothing \mid \{empty\}\}$.*

Definition 6 (FW-State)**.** *Given a protocol $\mathcal{P}$, a* FW-State *of $\mathcal{P}$ is a state in the constrained forwards strand semantics that is* reachable *from the initial state. The initial FW-State is $F_{init} = \{\varnothing\ \&\ \{empty\}\}$.*

The bisimulation relation is defined based on reachability, i.e., if a PA-State and a FW-State are in the relation $\mathcal{H}_{State}$, then they both can be reached from their corresponding initial states by the same label sequence. Note that we only consider states that are reachable from the initial states.

Definition 7 (Relation $\mathcal{H}_{State}$)**.** *Given a protocol $\mathcal{P}$, the relation $\mathcal{H}_{State}$ is defined as: $\mathcal{H}_{State} = \{(Pst, Fst) \in$ PA-State $\times$ FW-State $\mid \exists$ label sequence α s.t. $P_{init} \rightarrow_\alpha Pst$, $F_{init} \rightarrow_\alpha Fst\}$.*

Recall that a process can be "deconstructed" by the mapping *toCstrSS* into a set of constrained protocol strands, each representing a possible execution path. If a PA-State Pst and a FW-State Fst are related by $\mathcal{H}_{State}$, then an important observation is that there is a *duality* between individual processes in Pst and strands in Fst: if there is a process in the Pst describing a role's *continuation* in the future, there will be a corresponding strand in Fst describing the part of the process that has *already been executed*, and vice versa. Another observation is that, since the intruder's knowledge is extracted from the communication history, following the definition of $\mathcal{H}_{State}$, the states Pst and Fst have the same communication history, therefore they have the same intruder's knowledge. These observations lead us to the following bisimulation result, whose proof can be found in Appendix A.

Theorem 1 (Bisimulation)**.** *$\mathcal{H}_{State}$ is a bisimulation.*

7. Constrained Backwards Strand Semantics

In this section we extend Maude-NPA's symbolic backwards semantics with rules for constrained messages of the form described in Section 5.1, so that it can analyze protocols exhibiting explicit choices. We refer to this extended backwards semantics as *constrained backwards strand semantics*. We then show that the *constrained backwards strand semantics* is sound and complete with respect to the constrained forwards strand semantics presented in Section 6, and the process algebra semantics presented in Section 4. This result allows us to use Maude-NPA for analyzing protocols exhibiting choice, including both implicit and explicit choices, and in particular any protocol specified using the *protocol process algebra*.

The strand space model used in the constrained backwards strand semantics is the same as the one already used in Maude-NPA [6], except for the following differences:

- Maude-NPA explores *constrained states* as defined in [9], that is, states that have an associated a constraint store. More specifically, a *constrained state* is a pair $\langle St, \Psi\rangle$ consisting of a state expression St and a *constraint*, i.e., a set Ψ understood as a conjunction $\Psi = \bigwedge_{i=1}^{n} c_i$ of constraints.
- Strands are now of the form $[u_1, \ldots, u_i \mid u_{i+1}, \ldots u_n]$, where each u_k can be of one of these forms: (i) m^+ if it is a sent message, (ii) m^- if it is a received message, or (iii) $\{Cstr, Num\}$ if it is a constrained message.

State changes are described by a set $R^{-1}_{CstrB_\mathcal{P}}$ of *rewrite rules*, so that the rewrite theory $(\Sigma_{CstrSS_\mathcal{P}}, E_{SS_\mathcal{P}}, R^{-1}_{CstrB_\mathcal{P}})$ characterizes the behavior of protocol $\mathcal{P}$ modulo the equations $E_{SS_\mathcal{P}}$ for *backwards* execution. The set of rules $R^{-1}_{CstrB_\mathcal{P}}$ is obtained as follows. First, we adapt the set of rules $R^{-1}_{B_\mathcal{P}}$ in [6] to constrained states, which is an embedding of rules in $R^{-1}_{B_\mathcal{P}}$, as shown below:

$$\begin{array}{l}\langle\{SS\,\&\,[L \mid M^-, L']\,\&\,\{M{\in}\mathcal{I}, IK\}\}, \Psi\rangle\\ \quad\rightarrow \langle\{SS\,\&\,[L, M^- \mid L']\,\&\,\{M{\in}\mathcal{I}, IK\}\}, \Psi\rangle\end{array} \qquad \text{(B-)}$$

$$\begin{array}{l}\langle\{SS\,\&\,[L \mid M^+, L']\,\&\,\{IK\}\}, \Psi\rangle\\ \quad\rightarrow \langle\{SS\,\&\,[L, M^+ \mid L']\,\&\,\{IK\}\}, \Psi\rangle\end{array} \qquad \text{(B+)}$$

$$\begin{array}{l}\langle\{SS\,\&\,[L \mid M^+, L']\,\&\,\{M{\notin}\mathcal{I}, IK\}\}, \Psi\rangle\\ \quad\rightarrow \langle\{SS\,\&\,[L, M^+ \mid L']\,\&\,\{M{\in}\mathcal{I}, IK\}\}, \Psi\rangle\end{array} \qquad \text{(B++)}$$

$$\begin{array}{l}\forall\,[l_1, u^+, l_2] \in \mathcal{P}:\\ \langle\{\{SS\,\&\,[\,l_1 \mid u^+, l_2\,]\,\&\,\{u{\notin}\mathcal{I}, IK\}\}, \Psi\rangle\\ \rightarrow \langle\{SS\,\&\,\{u{\in}\mathcal{I}, IK\}\}\}, \Psi\rangle\end{array} \qquad \text{(B\&)}$$

where L and L' are variables denoting a list of strand messages, *IK* is a variable for a set of intruder facts ($m{\in}\mathcal{I}$ or $m{\notin}\mathcal{I}$), *SS* is a variable denoting a set of strands, and l_1, l_2 denote a list of strand messages.

Then, we define new transition rules for constrained messages. That is, we add the reversed version of the following rules:

$$\begin{array}{l}\langle\{SS\,\&\,\{IK'\}\,\&\,(ro)[L \mid \{?, Num\}, L']\}, \Psi\rangle\\ \quad\rightarrow \langle\{SS\,\&\,\{IK'\}\,\&\,(ro)[L, \{?, Num\} \mid L']\}, \Psi\rangle\end{array} \qquad \text{(B?)}$$

$$\begin{array}{l}\langle\{SS\,\&\,\{IK\}\,\&\,(ro)[L \mid \{M =_{E_\mathcal{P}} M, Num\}, L']\}, \Psi\rangle\\ \quad\rightarrow \langle\{SS\,\&\,\{IK\}\,\&\,(ro)[L, \{M =_{E_\mathcal{P}} M, Num\} \mid L']\}, \Psi\rangle\end{array} \qquad \text{(Bif=)}$$

$$\begin{array}{l}\langle\{SS\,\&\,\{IK\}\,\&\,(ro)[L \mid \{M \neq M', Num\}, L']\}, (\Psi \wedge M \neq M')\rangle\\ \quad\rightarrow \langle\{SS\,\&\,\{IK\}\,\&\,(ro)[L, \{M \neq M', Num\} \mid L']\}, \Psi\rangle\\ \text{if } (\Psi \wedge M \neq_{E_\mathcal{P}} M') \text{ is satisfiable in } \mathcal{T}_{\Sigma_{CstrSS_\mathcal{P}}/E_\mathcal{P}}(\mathcal{X})\end{array} \qquad \text{(Bif}{\neq}\text{)}$$

Rule (B?) processes a constraint message denoting an explicit non-deterministic choice with constant "?". The constraint store is not changed and no satisfiability check is required.

Rules (Bif=) and (Bif≠) deal with constrained messages associated to explicit deterministic choices. Since the only constraints we allow in explicit deterministic choices are equalities and disequalities, rule (Bif=) is for the case when the constraint is an equality, rule (Bif≠) is for the case when the constraint is a disequality. The equality constraint is solved by $E_\mathcal{P}$-unification. The constraint in a *constrained state* is therefore a *disequality constraint*, i.e., $\Psi = \bigwedge_{i=1}^{n} u_i \neq_{E_\mathcal{P}} v_i$. The *semantics* of such a constrained state,

written $[\![\langle St, \Psi \rangle]\!]$ is the set of all ground substitution instances of the form:

$$[\![\langle St, \Psi \rangle]\!] = \{St\theta \mid \theta \in [\mathcal{X} \to \mathcal{T}_{\Sigma_\mathcal{P}}] \wedge u_i\theta \neq_{E_\mathcal{P}} v_i\theta, 1 \leqslant i \leqslant n\}$$

The disequality constraints are then solved the same way as in [9].

Definition 8. *Let $\mathcal{P}$ be a protocol with signature $\Sigma_{CstrSS_\mathcal{P}}$ and equational theory $E_\mathcal{P}$. We define the* constrained backwards rewrite theory characterizing *$\mathcal{P}$ to be $(\Sigma_{CstrSS_\mathcal{P}}, E_{SS_\mathcal{P}}, R^{-1}_{CstrB_\mathcal{P}})$ where $E_{SS_\mathcal{P}}$ is same as explained in Section 3. $R^{-1}_{CstrB_\mathcal{P}}$ is the result of reversing the rewrite rules* {(B-), (B+), (B++), (B?), (Bif=), (Bif≠)}∪ (B&).

7.1 Soundness and Completeness of Constrained Backwards Strand Semantic

The soundness and completeness proofs generalize the proofs in [7]. Recall that the state in the constrained states of constrained backwards strand semantics is a symbolic strand state, i.e., a state with variables. A state in the forwards strand semantics is a ground strand state, i.e., a state without variables. The lifting relation defines the instantiation relation between symbolic and ground states. We first extend the lifting relation in [7] with constraints and constrained messages. Note that the u_i in the definition below can be sent messages, received messages, or constrained messages.

Definition 9 (Lifting Relation)**.** *Given a protocol $\mathcal{P}$, a constrained symbolic strand state $CstrS = \langle S, \Psi \rangle$ and a ground strand state s, we say that s* lifts *to $CstrS$, or that $CstrS$* instantiates *to s with a* ground *substitution $\theta : (Var(S) - \{SS, IK\}) \to \mathcal{T}_{\Sigma_\mathcal{P}}$, written $CstrS >^\theta s$ iff*

- *for each strand $:: r_1, \ldots, r_m :: [u_1, \ldots u_{i-1} \mid u_i, \ldots, u_n]$ in S, there exists a strand $[v_1, \ldots v_{i-1}]$ in s such that $\forall 1 \leqslant j \leqslant i-1$, $v_j =_{E_\mathcal{P}} u_j\theta$.*
- *for each positive intruder fact $w \in \mathcal{I}$ in S, there exists a positive intruder fact $w' \in \mathcal{I}$ in s such that $w' =_{E_\mathcal{P}} w\theta$, and*
- *for each negative intruder fact $w \notin \mathcal{I}$ in S, there is no positive intruder fact $w' \in \mathcal{I}$ in s such that $w' =_{E_\mathcal{P}} w\theta$.*
- *$E_\mathcal{P} \models \Psi\theta$.*

In the following we show the soundness and completeness of constrained backwards strand semantics w.r.t. the constrained forwards strand semantics. Several auxiliary results and the proofs of Theorems 2 and 3 below can be found in Appendix B.

Extending the proofs in [7], we first proved how the lifting of a ground state to a symbolic state induces a lifting of a forwards rewriting step in the forwards semantics to a backwards narrowing step in the backwards semantics, i.e., the completeness of one-step transition. Theorem 2 below then follows straightforwardly.

Theorem 2 (Completeness)**.** *Given a protocol $\mathcal{P}$, two ground strand states s, s_0, a constrained symbolic strand state $CstrS$ and a substitution θ s.t. (i) s_0 is an initial state, (ii) $s_0 \to^n s$, and (iii) $CstrS >^\theta s$. Then there exists a constrained symbolic initial strand state $CstrS_0$, two substitutions μ and θ', and $k \leqslant n$, s.t. $CstrS_0 \overset{k}{\leftsquigarrow}_\mu CstrS$, and $CstrS_0 >^{\theta'} s_0$.*

The Soundness Theorem from [7] can also be extended to constrained backwards and forwards strand semantics. It shows that the backwards symbolic reachability analysis is *sound* with respect to the forwards rewriting-based strand semantics.

Theorem 3 (Soundness)**.** *Given a protocol $\mathcal{P}$, two constrained symbolic strand states $CstrS_0, CstrS'$, an initial ground strand state s_0 and a substitution θ s.t. (i) $CstrS_0$ is a symbolic initial state, and (ii) $CstrS_0 \overset{*}{\leftsquigarrow} CstrS'$, and (iii) $CstrS_0 >^\theta s_0$. Then there exists a ground strand state s' and a substitution θ', s.t. (i) $s_0 \to^* s'$, and (ii) $CstrS' >^{\theta'} s'$.*

The soundness and completeness results in Theorems 3 and 2 together with the bisimulation proved in Theorem 1 show that the backwards symbolic reachability analysis is *sound* and *complete* with respect to the process algebra semantics.

Theorem 4 (Soundness)**.** *Given a protocol $\mathcal{P}$, two constrained symbolic strand states $CstrS_0, CstrS$, the initial FW-State F_{init}, a substitution θ, and the initial PA-State P_{init} s.t. (i) $CstrS_0$ is a symbolic initial strand state, and (ii) $CstrS_0 \overset{*}{\leftsquigarrow}_\mu CstrS$, and (iii) $CstrS_0 >^\theta F_{init}$. Then there exists a FW-State Fst such that $CstrS >^{\theta'} Fst$, and therefore, there is a PA-State Pst such that $Pst\ \mathcal{H}_{State}\ Fst$.*

Theorem 5 (Completeness)**.** *Given a protocol $\mathcal{P}$, a PA-State Pst, a FW-State Fst, a constrained symbolic strand state $CstrS$ s.t. (i) $Pst\ \mathcal{H}_{State}\ Fst$, (ii) $CstrS >^{\theta'} Fst$. Then there is a backwards symbolic execution $CstrS_0 \overset{*}{\leftsquigarrow}_\mu CstrS$ s.t. $CstrS_0$ is a symbolic initial strand state as defined in Section 3, and $CstrS_0 >^\theta F_{init}$.*

8. Protocol Experiments

In this section we describe some preliminary experiments[2] that we have performed on protocols with choice using a prototype extension of the Maude-NPA cryptographic protocol analysis tool. To validate our approach, we have chosen both simple and complex protocols exhibiting either nondeterministic choice or deterministic choice or both.

8.1 Choice of Encryption Type

This protocol allows either public key encryption or shared key encryption to be used for Alice to communicate with Bob. Alice initiates the conversation by sending out a message containing the chosen encryption mode, then Bob replies by sending an encrypted message containing his session key. The encryption mode is chosen nondeterministically by Alice. Therefore, it exhibits an *explicit nondeterministic choice*. Below we show the protocol description: the first one reflects the case in which public key encryption (denoted by $PubKey$) is chosen.

1. $A \to B : A; B; PubKey$
2. $B \to A : pk(A, B; SK)$
3. $A \to B : pk(B, A; SK; N_A\}$
4. $B \to A : pk(A, B; N_A)$

The second one reflects the case in which a shared key encryption (denoted by $SharedKey$) is chosen.

1. $A \to B : A; B; SharedKey$
2. $B \to A : shk(key(A, B), B; SK)$
3. $A \to B : shk(key(A, B), A; SK; N_A)$
4. $B \to A : shk(key(A, B), B; N_A)$

Note that A and B are names of principals, SK denotes the session key generated by B, and N_A denotes a nonce generated by A.

This protocol can alternatively be specified by treating the encryption mode as a choice variable which can be either public key encryption or shared key encryption, the continuation of the protocol will then be an explicit deterministic choice depending on the value of this choice variable. For our purposes of evaluating our approach on protocols with explicit nondeterministic choice, we use explicit nondeterministic choice here. We analyzed whether the intruder can learn the session key generated by Bob, when either the

[2] Available at `http://www.fan-yang.com/publications/choice.html`

public key encryption or shared key encryption is chosen, assuming both the principals are honest. For this property, Maude-NPA terminated without any attack being found.

8.2 Rock-Paper-Scissors

To evaluate our approach on protocols with explicit deterministic choices, we have used a simple protocol which simulates the famous Rock-Paper-Scissors game, in which Alice and Bob are the two players of the game. In this game, Alice and Bob commit to each other their hand shapes, which are later on revealed to each other after both players committed their hand shapes. The result of the game is then agreed upon between the two players according to the rule: rock beats scissors, scissors beats paper and paper beats rock. They finish by verifying with each other they both reached the same conclusion. Thus, at the end of the protocol each party should know the outcome of the game and whether or not the other party agrees to the outcome. This protocol exhibits *explicit deterministic choice* because the result of the game depends on the evaluation of the committed hand shapes according to the game rule. Note that this protocol also exhibits *implicit nondeterministic choice*, since the hand shape of the players are chosen by the players during the game.

The protocol proceeds as follows. First, both initiator and responder choose their hand shapes and send them to each other using a secure commitment scheme. Next, they both send each other the nonces that are necessary to open the commitments. Each of them then compares the two hand shapes and decides if the initiator wins, the responder wins, or there is a tie. The initiator then sends the responder the outcome. When the responder receives the initiator's verdict, it compares it against its own. It responds with "finished" if it agrees with the initiator and "cheater" if it doesn't. All messages are signed and encrypted, and the initiator's and responder's nonces are included in the messages concerning the outcome of the game. The actual messages sent and choices made are described in more detail below.

1. $A \rightarrow B : pk(B, sign(A, commit(N_A, X_A)))$
2. $B \rightarrow A : pk(A, sign(B, commit(N_B, X_B)))$
3. $A \rightarrow B : pk(B, sign(A, N_A)$
4. $B \rightarrow A : pk(A, sign(B, N_B)$
5. $if\ (X_A\ beats\ X_B)\ then\ R = Win$
   $else\ if\ (X_B\ beats\ X_A)\ then\ R = Lose$
   $else\ if\ (X_B\ =\ X_A)\ then\ R =\ Tie$
6. $A \rightarrow B : pk(B, sign(A, N_A; N_B :; R)))$
7. $if\ (R = Win \& X_A\ beats\ X_B)$
   $or\ (R\ =\ Lose\ \&\ X_B\ beats\ X_A)$
   $or\ (R\ =\ Tie\ \&\ X_A\ =\ X_B)$
   $then\ B \rightarrow A : pk(A, sign(B, N_A; N_B;\ finished\ ))$
   $else\ B \rightarrow A : pk(A, sign(B, N_A; N_B;\ cheater\ ))$

We first tried to see whether the protocol can simulate the game successfully, so we asked for different scenarios in which the player Alice or Bob can win in a round of the game. Maude-NPA was able to generate the expected scenarios, and it did not generate any others. We then gave Maude-NPA a secrecy attack state, in which the intruder, playing the role of initiator against an honest responder, attempts to guess its nonce before the responder receives its commitment. Finally we specified an authentication attack state in which we asked if a responder could complete a session with an honest initiator with the conclusion that the initiator had carried out its rule faithfully, without that actually having happened. For both of these attack states Maude-NPA finished its search without finding any attacks.

One interesting feature of the Rock-Scissors-Paper protocol, is that, in order verify that the commitment has been opened successfully (that is, that the nonce received is the nonce used to create the commitment, one must verify that the result of opening it is well-.; that is, that it is equal to "rock", "scissors", or "paper". This can be done via the evaluation of predicates. First, we create a sort Item and declare the constants "rock", "scissors", and "paper" to be of sort Item. Then we create a variable $X : Item$ of sort Item. We then define a predicate $item?$ such that $item?X : Item$ evaluates to true. Since only terms of sort Item can be unified with $X : Item$, this predicate can be used to check whether or not a term is of sort Item.

8.3 TLS

In Section 1.1 we introduced a simplified version of the handshake protocol in TLS 1.3 [19]. Even this simplified version produced a very large search space, because of the long list of messages and the concurrent interactions of a big amount of choices. We are however able to check the correctness of our specification by producing legal executions in Maude-NPA. Unlike TLS 1.3, we intentionally introduced a "downgrade attack" in our version in which the attacker can trick the principals into using a weaker crypto system. However, we have not yet been able to produce this attack because of the very deep and wide analysis tree (i.e., long reachability sequences with many branches) that is produced. We are currently investigating more efficient ways of managing list processing.

9. Related Work

As we mentioned in the introduction, there is a considerable amount of work on adding choice to the strand space model that involves embedding it into other formal systems, including event-based models for concurrency [4], Petri nets [11], or multiset rewriting [3]. Crazzolara and Winskel model nondeterministic choice as a form of composition, where a conflict relation is defined between possible child strands so that the parent can compose with only one potential child. In [11] Fröschle uses a Petri net model to add branching to strand space *bundles*, which represent the concurrent execution of strand space roles. Note that we have taken the opposite approach of representing bundles as traces of non-branching strands, where a different trace is generated for each choice taken. Although this results in more bundles during forward execution, it makes little difference in backwards execution, and is more straightforward to implement in an already existing analysis tool.

We also note that deterministic choice has been included in the applied pi calculus for cryptographic protocols [2], another widely used formal model, based on Milner's pi calculus [17]. The applied pi calculus includes the rule $if\ M\ =\ N\ then\ P\ else\ Q$, where P and Q are terms. This is similar to our syntax for deterministic choice. However our long-term plan is to add other types or predicates as well (e.g. $M\ subsumes\ N$) ; indeed our approach extends to any type of predicate that can be evaluated on a ground state. Although the applied pi calculus in its original form does not include nondeterministic choice, both nondeterministic and probabilistic choice have been added in subsequent work [12].

In addition, Olarte and Valencia show in [18] how a cryptographic protocol modeling language can be expressed in their universal timed concurrent constraint programming (utcc) model, a framework that extends the timed concurrent constraint programming model to express mobility. The language does not support choice, but utcc does, and it does not appear that it would be diffi-

cult to extend the language to incorporate the utcc choice mechanisms.

The Tamarin protocol analysis tool [14] includes deterministic branching, which was used extensively in the analysis of TLS 1.3 [5]. In particular, it includes an optimization for roles of the form $P.(if\ T\ then\ Q\ else\ R).S$; when backwards search is used, it is sometimes possible to capture such an execution in terms of just one strand until the conditional is encountered, thus reducing the state space. Our approach produces two strands, but since the process algebra semantics makes it easy to tell whether or not R behaves "essentially" the same no matter if P or Q is chosen, we believe we have a pathway for including such a feature if desired.

10. Conclusions

We have provided an extension to the strand space model that allows for both deterministic and nondeterministic choice, together with an operational semantics for choice in strand spaces that not only provides a formal foundation for choice, but allows us to implement it directly in the Maude-NPA cryptographic protocol analysis tool. In particular, we have applied Maude-NPA to several protocols that rely on choice in order to validate our approach.

This work not only provides a choice extension to strand spaces, but extends tem in other ways as well. First of all, it provides a process algebra for strand spaces. This potentially allows us to relate the strand space model to other formal systems (e.g., the applied pi calculus [1]) giving a better understanding of how it compares with other formal models. In addition, the process algebra semantics gives us a basis for creating a new specification language for Maude-NPA, which we believe will be more natural to the user than the current strand-space language.

Another contribution of this work is that it provides a means for evaluating both equality and disequality predicates in the strand space model and in Maude-NPA. This allows us to implement features such as type checking in Maude-NPA, via predicates such as $foocheck(X)$, where $foocheck(0 : Foo) = tt$, that is, $foocheck(X)$ succeeds only if X is of sort Foo. This proved to be very helpful, for example, in our specification of the Rock-Scissors-Paper protocol as we described earlier. We believe the expressiveness of Maude-NPA can be further increased at little cost by extending the types of predicates that can be evaluated, e.g. by including predicates for subsumption and their negation. This is another subject for further investigation.

References

[1] M. Abadi. Leslie lamport's properties and actions. In *Proceedings of the Twentieth Annual ACM Symposium on Principles of Distributed Computing, PODC 2001*, page 15, 2001.

[2] M. Abadi and C. Fournet. Mobile values, new names, and secure communication. In *Conference Record of POPL 2001: The 28th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 104–115, 2001.

[3] I. Cervesato, N. A. Durgin, J. C. Mitchell, P. Lincoln, and A. Scedrov. Relating strands and multiset rewriting for security protocol analysis. In *Proceedings of the 13th IEEE Computer Security Foundations Workshop, CSFW '00*, pages 35–51, 2000.

[4] F. Crazzolara and G. Winskel. Composing strand spaces. In *FST TCS 2002: Foundations of Software Technology and Theoretical Computer Science*, pages 97–108, 2002.

[5] C. Cremers, M. Horval, S. Scott, and T. van der Merwe. Automated analysis and verification of TLS 1.3:0-RTT, resumption and delayed authentication. In *IEEE Security and Privacy, 2016*, to appear: 2016.

[6] S. Escobar, C. Meadows, and J. Meseguer. Maude-NPA: Cryptographic protocol analysis modulo equational properties. In *Foundations of Security Analysis and Design V, FOSAD 2007/2008/2009 Tutorial Lectures*, LNCS vol. 5705, pages 1–50. Springer, 2009.

[7] S. Escobar, C. Meadows, J. Meseguer, and S. Santiago. A rewriting-based forwards semantics for Maude-NPA. In *Proceedings of the 2014 Symposium and Bootcamp on the Science of Security, HotSoS 2014*. ACM, 2014.

[8] S. Escobar, C. Meadows, J. Meseguer, and S. Santiago. State space reduction in the maude-nrl protocol analyzer. *Inf. Comput.*, 238:157–186, 2014.

[9] S. Escobar, C. Meadows, J. Meseguer, and S. Santiago. Symbolic protocol analysis with disequality constraints modulo equational theories. In *Programming Languages with Applications to Biology and Security - Essays Dedicated to Pierpaolo Degano on the Occasion of His 65th Birthday*, pages 238–261, 2015.

[10] F. J. T. Fabrega, J. Herzog, and J. Guttman. Strand Spaces: What Makes a Security Protocol Correct? *Journal of Computer Security*, 7: 191–230, 1999.

[11] S. B. Fröschle. Adding branching to the strand space model. *Electr. Notes Theor. Comput. Sci.*, 242(1):139–159, 2009.

[12] J. Goubault-Larrecq, C. Palamidessi, and A. Troina. A probabilistic applied pi-calculus. In *Programming Languages and Systems, 5th Asian Symposium, APLAS 2007*, pages 175–190, 2007.

[13] J.-P. Jouannaud and H. Kirchner. Completion of a set of rules modulo a set of equations. *SIAM J. Comput.*, 15(4):1155–1194, 1986.

[14] S. Meier, B. Schmidt, C. Cremers, and D. A. Basin. The TAMARIN prover for the symbolic analysis of security protocols. In *Computer Aided Verification - 25th International Conference, CAV 2013*, pages 696–701, 2013.

[15] J. Meseguer. Conditional rewriting logic as a united model of concurrency. *Theor. Comput. Sci.*, 96(1):73–155, 1992.

[16] J. Meseguer. Membership algebra as a logical framework for equational specification. In *WADT 97*, pages 18–61, 1997.

[17] R. Milner. *Communicating and mobile systems - the Pi-calculus*. Cambridge University Press, 1999. ISBN 978-0-521-65869-0.

[18] C. Olarte and F. D. Valencia. The expressivity of universal timed CCP: undecidability of monadic FLTL and closure operators for security. In *Proceedings Principles and Practice of Declarative Programming 2008*, pages 8–19, 2008.

[19] E. Rescorla. The transport layer security (tls) protocol version 1.3. Technical Report draft-ietf-tls-tls13-12, IETF, 2016.

[20] S. Santiago, S. Escobar, C. A. Meadows, and J. Meseguer. Effective sequential protocol composition in maude-npa. *CoRR*, abs/1603.00087, 2016.

[21] TeReSe, editor. *Term Rewriting Systems*. Cambridge University Press, Cambridge, 2003.

A. Bisimulation Proofs

We prove in this section that the relation $\mathcal{H}_{State}$ is a bisimulation. We first show in Lemmas 1 and 2 some properties of states that can be related by the relation $\mathcal{H}_{State}$. Lemma 1 shows that if a PA-State Pst and FW-State Fst are related by $\mathcal{H}_{State}$, then each individual process in Pst and strand in Fst are related by the relation $\mathcal{H}_{LP\_Str}$. Processes and strands related by $\mathcal{H}_{LP\_Str}$ can have the same future behaviors under same intruder knowledge. In Lemma 2 we then show that Pst and Fst have the same intruder knowledge. Following these two lemmas, we prove by case analysis on the transition labels that the relation $\mathcal{H}_{State}$ is a bisimulation

Let us first define the notation of label sequence that we will use in the proofs.

Definition 10 (Label Sequence)**.** *An ordered sequence α of transition labels is defined by using $\_.\_$ as an associative concatenation operator with nil as an identity. The length of a label sequence α is denoted by $|\alpha|$. Given a label sequence α, we denote by $\alpha|_{(ro,i)}$ the sub-sequence of labels in α that have ro as role name, and i as identifier, i.e. labels of the form $(ro, i, \_, \_, \_)$ ($\_$ is a shorthand for denoting any term).*

We then define the relation $\mathcal{H}_{LP\_Str}$, which relates a possibly partially executed labeled process and a constrained strand. Recall that a process can be "deconstructed" by the mapping *toCstrSS* into a set of constrained protocol strands, each representing a possible execution path. If a labeled process LP is related to a constrained strand Str by the relation $\mathcal{H}_{LP\_Str}$, then: (i) LP and Str denote the behavior of the same role with the same identity in the same protocol, and (ii) for any strand Str_{LP}, Str_{LP} denotes a possible execution path of LP iff Str followed by Str_{LP} forms a valid possible execution path of the protocol.

Definition 11 (Relation $\mathcal{H}_{LP\_Str}$)**.** *Given a protocol $\mathcal{P}$, and a possibly partially executed labeled process LP of $\mathcal{P}$, a possibly partially executed constrained strand Str of $\mathcal{P}$, then $(LP, Str) \in \mathcal{H}_{LP\_Str}$ iff toCstrSS$(LP) = \&\{(ro, i)[u_{j+1}, \ldots, u_n]\rho_{ro,i}\theta \mid$ $\exists$ ground substitution θ $\exists (ro)[u_1, \ldots u_j, u_{j+1}, \ldots, u_n] \in \mathcal{P}_{Cstr}$ s.t. $Str = (ro, i)[u_1, \ldots u_j]\rho_{ro,i}\theta\}$*

Where $\&\{S_1, S_2, \ldots, S_n\}$ is a shorthand for a term $S_1 \& S_2 \& \ldots \& S_n$ denoting a set of strands. $\rho_{ro,i} = \{r_1 \mapsto r_1.ro.i, \ldots, r_m \mapsto r_m.ro.i\}$ for fresh variables $r_1, \ldots, r_m$ in $[u_1, \ldots u_j, u_{j+1}, \ldots, u_n]$.

Example A.1. *Following Examples 4.2 and 5.1, we show a process LP and a strand Str that are related by relation $\mathcal{H}_{LP\_Str}$. LP (resp. Str) is the labeled process (resp. constrained strand) of the Server role after making the first explicit nondeterministic choice.*

$$
\begin{aligned}
LP =& (Server, 1, 2)\ \sigma(+(hs; retry) \cdot -(hs; N'; G'; gen(G'); E') \cdot \\
&+ (hs; n(S_?, r1); G'; gen(G'); keyG(G', S_?, r_2); \\
&Z(AReq_?, G', E', S, r_2, S_?, HM)))\\
Str =& (Server, 1)\ \sigma[\ \{?, 1\}, -(hs; N; G; gen(G); E)]
\end{aligned}
$$

where σ is a ground substitution to the pattern variables N, G, and E.

Notice the *duality* between the process description and that of its constrained strand: the process describes its *continuation* in the future, whereas its strand describes the part of the process that has *already been executed*.

Lemma 1. *Let $Pst = \{LP_1 \& \ldots \& LP_n \mid \{IK\}\}$ be a PA-State and $Fst = \{Str_1 \& \ldots \& Str_m \& \{IK'\}\}$ be a FW-State, if $(Pst, Fst) \in \mathcal{H}_{State}$, i.e., exists a label sequence α such that $P_{init} \rightarrow_\alpha Pst$, and $F_{init} \rightarrow_\alpha Fst$, then:*

(i) For each labeled process $LP_k \in Pst$, $1 \leqslant k \leqslant n$, there exists a strand $Str_{k'} \in Fst$, $1 \leqslant k' \leqslant m$, such that $(LP_k, Str_{k'}) \in \mathcal{H}_{LP\_Str}$.

(ii) For each strand $Str_{k'} \in Fst$, $1 \leqslant k' \leqslant m$, there exists a labeled process $LP_k \in Pst$, $1 \leqslant k \leqslant n$, such that $(LP_k, Str_{k'}) \in \mathcal{H}_{LP\_Str}$.

Proof. We first prove property (i). If $|\alpha| = 0$, since both the strand set and the process configuration are empty, the statement is vacuously true.

Now suppose that $|\alpha| > 0$. Then without lose of generality, assume there exists a labeled process $LP_k = ((ro, i, j)\ P_k)$ in Pst, with $i, j \geqslant 1$. Then there is at least one label in α of the form $(ro, i, \_, \_, \_)$ ($\_$ is a short hand for any content), therefore, there is a strand $St_{k'}$ in Fst of the form $(ro, i)[v_1, \ldots, v_{j'}]$.

We then show that the above mentioned LP_k and $Str_{k'}$ are related by $\mathcal{H}_{LP\_Str}$, i.e., $(LP_k, Str_{k'}) \in \mathcal{H}_{LP\_Str}$. Since the state Fst is reachable from the initial state by the label sequence α, and $Str_{k'} \in Fst$, $[v_1, \ldots, v_{j'}]$ denotes exactly the sequence of messages in the unique sequence of labels $\alpha|_{(ro,i)}$. Moreover, $j' = j - 1$.

Since the process state Pst is reachable from the initial state P_{init} by label sequence α, there exists a unique process $(ro)P_{spec}$ in the specification P_{PA}, and LP_k represents all possible behaviors of $(ro)P_{spec}$ after the sequence of transitions $\alpha|_{(ro,i)}$. Therefore, *toCstrSS*$(LP_k) =$

$$
\begin{aligned}
&\&\{(ro, i)[u_j, \ldots, u_n]\rho_{ro,i}\theta \mid \\
&\exists \textit{ ground substitution } \theta \\
&\exists (ro)[u_1, \ldots, u_{j-1}, u_j, \ldots, u_n] \in \textit{toCstrSS}((ro)P_{spec}) \\
&\textit{s.t. } (ro, i)[u_1, \ldots, u_{j-1}]\rho_{ro,i}\theta = (ro, i)[v_1, \ldots, v_{j-1}]\}
\end{aligned}
$$

By the correspondence between protocol specifications defined in definition 3 , $\mathcal{P}_{CstrF} =$ *toCstrSS*(P_{PA}). Also note that $(ro)P_{spec}$ is the only process in P_{PA} that has ro as its role name, therefore, *toCstrSS*$((ro)P_{spec}) = \{(ro)[u_1, \ldots, u_n] \mid (ro)[u_1, \ldots, u_n] \in \mathcal{P}_{CstrF}\}$. Therefore, *toCstrSS*$(LP_k) =$

$$
\begin{aligned}
&\&\{(ro, i)[u_j, \ldots u_n]\rho_{ro,i}\theta \mid \\
&\exists \textit{ ground substitution } \theta, \\
&\exists (ro)[u_1, \ldots, u_{j-1}, u_j, \ldots, u_n] \in \mathcal{P}_{CstrF} \\
&\textit{s.t. } (ro, i)[u_1, \ldots, u_{j-1}]\rho_{ro,i}\theta = (ro, i)[v_1, \ldots, v_{j-1}]\}.
\end{aligned}
$$

Therefore, $(LP_k, Str_{k'}) \in \mathcal{H}_{LP\_Str}$.

The proof for property (ii) above is similar to the one for property (i). □

Since the intruder knowledge in a PA-State or FW-State can be extracted from the historical message exchange sequences, which are kept track of in the transition labels, the equivalence of label sequence implies the same intruder knowledge. This property is shown in the lemma below.

Lemma 2. *Given a PA-State Pst and a FW-State Fst such that $(Pst, Fst) \in \mathcal{H}_{State}$, i.e., there exists a label sequence α such that $P_{init} \rightarrow_\alpha Pst$ and $F_{init} \rightarrow_\alpha Fst$, then the contents of intruder knowledge in Pst and in Fst are syntactically equal.*

Proof. In both semantics the only transition rules that add new elements to the intruder's knowledge are the ones whose label is of the form $(ro, i, j, +m, n)$. Therefore, given the two states Pst and Fst as described above, their intruder's knowledge can be computed from the sequence of labeled transitions α as $IK(Pst) = IK(Fst) = \{m \in \mathcal{I} \mid (\_, \_, \_, +m, \_) \in \alpha\}$. □

A.1 Proof of Theorem 1.

Based on the lemmas above, we then show that the relation $\mathcal{H}_{State}$ is a bisimulation, i.e., Theorem 1. More specifically,

i) $(P_{init}, F_{init}) \in \mathcal{H}_{State}$.

ii) For all PA-State Pst_n, and FW-State Fst_n, if $(Pst_n, Fst_n) \in \mathcal{H}_{State}$, and there exists a PA-State Pst_{n+1} such that $Pst_n \rightarrow_a Pst_{n+1}$, then there exists a FW-State Fst_{n+1} such that $Fst_n \rightarrow_a Fst_{n+1}$ and $(Pst_{n+1}, Fst_{n+1}) \in \mathcal{H}_{State}$.

iii) For all PA-State Pst_n, and FW-State Fst_n, if $(Pst_n, Fst_n) \in \mathcal{H}_{State}$, and there exists a FW-State Fst_{n+1} such that $Fst_n \rightarrow_a Fst_{n+1}$, then there exists a PA-State Pst_{n+1} such that $Pst_n \rightarrow_a Pst_{n+1}$ and $(Pst_{n+1}, Fst_{n+1}) \in \mathcal{H}_{State}$

Proof. (i) Holds for the label sequence nil, since $P_{init} \rightarrow_{nil} P_{init}$ and $F_{init} \rightarrow_{nil} F_{init}$, therefore, $(P_{init}, F_{init}) \in \mathcal{H}_{State}$.

We now prove (ii). If $(Pst_n, Fst_n) \in \mathcal{H}_{State}$, by definition of the relation $\mathcal{H}_{State}$, there exists a label sequence α s.t. $P_{init} \rightarrow_\alpha Pst_n$ and $F_{init} \rightarrow_\alpha Fst_n$. Suppose there exists state Pst_{n+1} such that $Pst_n \rightarrow_a Pst_{n+1}$. We prove by case analysis on label a that there exists Fst_{n+1} such that $Fst_n \rightarrow_a Fst_{n+1}$. The fact that $(Pst_{n+1}, Fst_{n+1}) \in \mathcal{H}_{State}$ then follows this by the definition of relation $\mathcal{H}_{State}$.

In the rest of this proof, $\overrightarrow{L}$, $\overrightarrow{L_1}$ and $\overrightarrow{L_2}$ denote lists of messages, M, M' and m denote messages, P, Q and R denote processes, PS denotes a process configuration, SS denotes a set of constrained protocol strands, IK and IK' denote the set of messages in the intruder's knowledge.

1) $a = (ro, i, j, +m, 0)$: if $j > 1$, according to the semantics, $Pst_n \rightarrow_a Pst_{n+1}$ by applying rule (PA++), Pst_n is of the form $\{(ro, i, j)\ (+M \cdot P)\ \&\ PS \mid \{IK\}\}$ s.t. there exists a ground substitution σ binding the choice variables in M s.t. $m = M\sigma$, $Pst_{n+1} = \{(ro, i, j{+}1)\ P\sigma\ \&\ PS \mid \{m \in \mathcal{I}, IK\}\}$ and $m \in \mathcal{I} \notin IK$. Since $Pst_n\ \mathcal{H}_{State}\ Fst_n$, by Lemmas 1 and 2, Fst_n is of the form $\{(ro, i)\ [\overrightarrow{L}]\ \&\ SS\ \&\ \{IK\}\}$ s.t. $(ro, i, j)\ (+M \cdot P)\ \mathcal{H}_{LP\_Str}\ (ro, i)\ [\overrightarrow{L}]$. Let $(ro)[\overrightarrow{L_1}, \overrightarrow{L_2}]$ be a constrained strand in P_{CstrSS} s.t. there exists a ground substituion θ s.t. $\overrightarrow{L_1}\rho_{ro,i}\theta = \overrightarrow{L}$. By the definition of relation $\mathcal{H}_{LP\_Str}$ and mapping *toCstrSS*, the first message of $\overrightarrow{L_2}$ is $+M'$, s.t. $M'\rho_{ro,i}\theta = M$. Then since $M\sigma = m$ and $m \in \mathcal{I} \notin IK$, the rule (F++) can be applied for the rewrite $Fst_n \rightarrow_a Fst_{n+1}$, where $Fst_{n+1} = \{(ro, i)\ [\overrightarrow{L}, +m]\ \&\ SS\ \&\ \{m \in \mathcal{I}, IK\}\}$.

   If $j = 1$, $Pst_n \rightarrow_a Pst_{n+1}$ by applying rule (PA&), there exists a process $(ro)\ (+M \cdot P)$ in P_{PA} and a ground substitution σ s.t. $M\rho_{ro,i}\sigma = m$. Since *toCstrSS*$(P_{PA}) = P_{CstrSS}$, by the definition of *toCstrSS*, for all strands of role ro in P_{CstrSS}, the first message is $+M$. Without loss of generality, let Pst_n be $\{PS \mid \{IK\}\}$, and Fst_n be $\{SS\ \&\ \{IK'\}\}$. Since the rule (PA&) can be applied, $m \in \mathcal{I} \notin IK$. By Lemma 2, $IK = IK'$. Moreover, by Lemma 1, *MaxStrId*$(SS, ro) =$ *MaxProcId*(PS, ro), and since *MaxProcId*$(PS, ro) + 1 = i$, by applying the rule (F++&) we get $Fst_n \rightarrow_a Fst_{n+1}$.
2) $a = (ro, i, j, M\sigma, 0)$: similar to case 1.
3) $a = (ro, i, j, -m, 0)$: if $j > 1$, according to the semantics, $Pst_n \rightarrow_a Pst_{n+1}$ by applying rule (PA-), Pst_n is of the form $\{(ro, i, j)\ (-M \cdot P)\ \&\ PS \mid \{m \in \mathcal{I}, IK\}\}$ s.t. $m =_{E_\mathcal{P}} M\sigma$ for some ground substitution σ and $Pst_{n+1} = \{(ro, i, j + 1)\ P\sigma\ \&\ PS \mid \{m \in \mathcal{I}, IK\}\}$. Since $Pst_n\ \mathcal{H}_{State}\ Fst_n$, by Lemmas 1 and 2, $Fst_n = \{(ro, i)\ [\overrightarrow{L}]\ \&\ SS\ \&\ \{m \in \mathcal{I}, IK\}\}$ s.t. $(ro, i, j)\ (-M \cdot P)\ \mathcal{H}_{LP\_Str}\ (ro)\ [\overrightarrow{L}]$. Let $(ro)[\overrightarrow{L_1}, \overrightarrow{L_2}] \in P_{CstrSS}$ s.t. there exists a ground substitution θ s.t. $\overrightarrow{L_1}\rho_{ro,i}\theta = \overrightarrow{L}$, then by definition of $\mathcal{H}_{LP\_Str}$ and *toCstrSS*, the first message of $\overrightarrow{L_2}$ is $-M'$ s.t. $M'\rho_{ro,i}\theta = M$. Since $m =_{E_\mathcal{P}} M\sigma$, rule (F-) can be applied to get the transition $Fst_n \rightarrow_a Fst_{n+1}$, where $Fst_{n+1} = \{(ro, i)\ [\overrightarrow{L}, -m]\ \&\ SS\ \&\ \{m \in \mathcal{I}, IK\}\}$.

   If $j = 1$, $Pst_n \rightarrow_a Pst_{n+1}$ by applying rule (PA&), therefore, there exists a process $(ro)\ (-M \cdot P)$ in P_{PA} and a ground substitution σ s.t. $M\rho_{ro,i}\sigma = m$. Without loss of generality, let Pst_n be $\{PS \mid \{IK\}\}$. Then $m \in \mathcal{I} \in IK$. Since *toCstrSS*$(P_{PA}) = P_{CstrSS}$, by the definition of *toCstrSS*, for all strands of role ro in P_{CstrSS}, the first message is $-M$. By Lemma 2, $m \in \mathcal{I}$ is in the intruder knowledge of Fst_n. Moreover, by Lemma 1, *MaxStrId*$(SS, ro) =$ *MaxProcId*(PS, ro), and since *MaxProcId*$(PS, ro) + 1 = i$, by applying the rule (F-&) we get $Fst_n \rightarrow_a Fst_{n+1}$.
4) $a = (ro, i, j, T, 1)$: according to the transition rules, $Pst_n \rightarrow_a Pst_{n+1}$ by applying rule (PAif1). Therefore Pst_n is of the form $\{(ro, i, j)((\mathit{if}\ \ c\ \mathit{then}\ P\ \mathit{else}\ Q)\ \cdot R)\ \&\ PS \mid \{IK\}\}$, $Pst_{n+1} = \{(ro, i, j + 1)\ (P \cdot R)\ \&\ PS \mid \{IK\}\}$ and $c =_{E_\mathcal{P}} true$. Since $Fst_n\ \mathcal{H}_{State}\ Pst_n$, by Lemma 1, $Fst_n = \{(ro)\ [\overrightarrow{L}]\ \&\ SS\ \&\ \{IK'\}\}$ s.t. $(ro, i, j)\ ((\mathit{if}\ c\ \mathit{then}\ P\ \mathit{else}\ Q) \cdot R)\ \mathcal{H}_{LP\_Str}\ (ro, i)\ [\overrightarrow{L}]$. By the definition of the relation $\mathcal{H}_{LP\_Str}$ and the mapping *toCstrSS*, there exists $(ro)[\overrightarrow{L_1}, \{C, 1\}, \overrightarrow{L_2}] \in P_{CstrSS}$ and a ground substitution θ s.t. $\overrightarrow{L} = \overrightarrow{L_1}\rho_{ro,i}\theta$, and $C\rho_{ro,i}\theta = c$. Since $c =_{E_\mathcal{P}} true$, the rule (Fif) can be applied for the rewrite $Fst_n \rightarrow_a Fst_{n+1}$, where $Fst_{n+1} = \{\{(ro)\ [\overrightarrow{L}, \{t, 1\}]\ \&\ SS\ \&\ \{IK'\}\}$
5) $a = (ro, i, j, T, 2)$: similar to case 4.
6) $a = (ro, i, j, ?, 1)$: if $j > 1$, $Pst_n \rightarrow_a Pst_{n+1}$ by applying rule (PA?1). Therefore Pst_n is of the form $\{(ro, i, j)\ ((P\ ?\ Q) \cdot R)\ \&\ PS \mid \{IK\}\}$ and $Pst_{n+1} = \{(ro, i, j + 1)\ (P \cdot R)\ \&\ PS \mid \{IK\}\}$. Since $Fst_n\ \mathcal{H}_{State}\ Pst_n$, by Lemma 1, $Fst_n = \{(ro, i)\ [\overrightarrow{L}]\ \&\ SS\ \&\ \{IK'\}\}$ s.t. $(ro, i, j)\ ((P\ ?\ Q) \cdot R)\ \mathcal{H}_{LP\_Str} (ro, i)\ [\overrightarrow{L}]$. By the definition of $\mathcal{H}_{LP\_Str}$ and *toCstrSS*, there is a strand $(ro, i)[\overrightarrow{L_1}, \{?, 1\}, \overrightarrow{L_2}] \in P_{CstrSS}$ s.t. $\overrightarrow{L} = \overrightarrow{L_1}\theta$. Therefore, rule (F?) can be applied for the rewrite $Fst_n \rightarrow_a Fst_{n+1}$, and $Fst_{n+1} = \{(ro, i)\ [\overrightarrow{L}, \{?, 1\}]\ \&\ SS\ \&\ \{IK'\}\}$.

   If $j = 1$, $Pst_n \rightarrow_a Pst_{n+1}$ by applying rule (PA&). Therefore, there exists a process $(ro)\ ((P\ ?\ Q) \cdot R)$ in P_{PA}. Since *toCstrSS*$(P_{PA}) = P_{CstrSS}$, by the definition of *toCstrSS*, there is a strand of role ro whose first message is $(?, 1)$ in P_{CstrSS}. Moreover, by Lemma 1, *MaxStrId*$(SS, ro) =$ *MaxProcId*(PS, ro), and since *MaxProcId*$(PS, ro) + 1 = i$, by applying the rule (F?&) we get $Fst_n \rightarrow_a Fst_{n+1}$.
7) $a = (ro, i, j, ?, 2)$ similar to case 6.

The proof for (iii) is similar. □

B. Soundness and Completeness Proofs

In this section we show the soundness and completeness of transitions in constrained backwards strand semantics w.r.t. the constrained forwards strand semantics by proving two lemmas stating the completeness and soundness of one-step transition in the constrained backwards strand semantics w.r.t. the constrained forwards strand semantics. The soundness and completeness result directly follows these two lemmas.

In the proofs of the lemmas we consider only transition rules added in both semantics to deal with explicit choices, that is, rules (Fif) $\cup$ (F?) $\cup$ (F?&) in the constrained forwards strand semantics and rules $\{$(B?), (Bif=), (Bif$\neq$)$\}$ in the constrained backwards strand semantics. The proof of the soundness and completeness of one-step transitions performed in the constrained backwards strand semantics using rules $\{$(B-), (B+), (B++)$\} \cup$ (B&) w.r.t to one-step transitions performed in the constrained forwards strand semantics using rules (F++) $\cup$ (F+) $\cup$ (F++&) $\cup$ (F+&) $\cup$ (F-) $\cup$ (F-&) is same as [7], since in these transitions no constraint is involved. Note

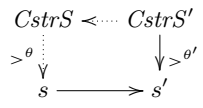

Figure 1. Lemma 3

that although in [7], *Choice Variables* were not defined explicitly, the proof extends to strands with choice variables naturally, since the lifting relation between a ground state and a symbolic state does not need to be changed to cover choice variables. Since the strand labels are irrelevant for the result of this section, we will omit the strand labels to simplify the notation from now on. Also, we include the fresh substitution in the substitutions and do not separate the fresh substitutions explicitly.

First, we recall the definition of a symbolic state and a ground state.

Definition 12 (Symbolic Strand State)**.** *Given a protocol $\mathcal{P}$, a* symbolic *strand state S of $\mathcal{P}$ is a term of the form:*

$$S = \{ :: r_{1_1}, \ldots, r_{m_1} :: [u_{1_1}, \ldots u_{i_1-1} \mid u_{i_1}, \ldots, u_{n_1}] \,\&$$
$$\vdots$$
$$:: r_{1_k}, \ldots, r_{m_k} :: [u_{1_k}, \ldots, u_{i_k-1} \mid u_{i_k}, \ldots, u_{n_k}] \,\&\, SS$$
$$\{w_1 \in \mathcal{I}, \ldots, w_m \in \mathcal{I},\ w'_1 \notin \mathcal{I}, \ldots, w'_{m'} \notin \mathcal{I}, IK\}\}$$

where for each $1 \leqslant j \leqslant k$, there exists a strand $[m_{1_j}, \ldots m_{i_j-1}, m_{i_j}, \ldots, m_{n_j}] \in P_{CstrSS}$ and a substitution $\rho_j : \mathcal{X} \to \mathcal{T}_{\Sigma_\mathcal{P}}(\mathcal{X})$ such that $m_{1_j}\rho_j =_{E_\mathcal{P}} u_{1_j}$, ..., $m_{n_j}\rho_j =_{E_\mathcal{P}} u_{n_j}$, SS is a variable denoting a (possibly empty) set of strands, and IK is a variable denoting a (possibly empty) set of intruder's knowledge facts.

Definition 13 (Ground Strand State)**.** *Given a protocol $\mathcal{P}$, a* ground *strand state s of $\mathcal{P}$ is a term without variables of the form:*

$$s = \{[u_{1_1}, \ldots u_{i_1-1}] \,\&\, \cdots \,\&\, [u_{1_k}, \ldots, u_{i_k-1}] \,\&$$
$$\{w_1 \in \mathcal{I}, \ldots, w_m \in \mathcal{I}\}\ \}$$

where for each $1 \leqslant j \leqslant k$, there exists a strand $[m_{1_j}, \ldots m_{i_j-1}, m_{i_j}, \ldots, m_{n_j}] \in P_{CstrSS}$ and a substitution $\rho_j : \mathcal{X} \to \mathcal{T}_{\Sigma_\mathcal{P}}$ such that $m_{1_j}\rho_j =_{E_\mathcal{P}} u_{1_j}$, ..., $m_{i_j}\rho_j =_{E_\mathcal{P}} u_{i_j}$.

B.1 Proof of Theorem 2

The lemma below extends the lifting lemma in [7] to strands with constrained messages. The lifting lemma shows how the lifting of a ground state to a symbolic state induces a lifting of a forwards rewriting step in the forwards semantics to a backwards narrowing step in the backwards semantics. The proof of Theorem 2 in Section 7.1 is a straightforward corollary of Lemma 3 below. The Lifting Lemma is illustrated by Figure 1.

Lemma 3 (Lifting Lemma)**.** *Given a protocol $\mathcal{P}$, two ground strand states s and s', a constrained symbolic strand state $CstrS' = \langle S', \Psi' \rangle$ and a substitution θ' s.t. $s \to s'$ and $CstrS' >^{\theta'} s'$, then there exists a constrained symbolic strand state $CstrS = \langle S, \Psi \rangle$ and a substitution θ s.t. $CstrS >^{\theta} s$ and either $CstrS \overset{\mu}{\leftsquigarrow} CstrS'$ or $CstrS = CstrS'$.*

Proof. As has been explained before, we only need to consider the new rules: (Fif), (F?), (F?&). The proof in [7] is divided in cases, some of which has specific requirement on intruder knowledge, or involve changes made to the intruder knowledge. Since all the new rules we are considering do not have specific requirements on the intruder knowledge, and do not change the intruder knowledge as well, so the cases that we need to consider are the following (cases e) and f) in the proof in [7]), which involve the appearance or non-appearance of certain strand(s):

e: There is a strand $[u_1, \ldots, u_{j-1}, u_j, \ldots, u_n]$ in P_{CstrSS}, $n \geqslant 1$, $1 \leqslant j \leqslant n$, and a substitution ρ such that $[u_1, \ldots, u_{j-1}, u_j]\rho$ is a strand in s' and $[u_1, \ldots, u_{j-1}, u_j \mid u_{j+1}, \ldots, u_n]\rho$ is a strand in $S'\theta'$.

f: There is a strand $[u_1, \ldots, u_{j-1}, u_j, \ldots, u_n]$ in P_{CstrSS}, $n \geqslant 1$, $1 \leqslant j \leqslant n$, and a substitution ρ such that $[u_1, \ldots, u_{j-1}, u_j]\rho$ is a strand in s' but $[u_1, \ldots, u_{j-1}, u_j \mid u_{j+1}, \ldots, u_n]\rho$ is not a strand in $S'\theta'$.

Now we consider for the forward rewrite rule application in the step $s \to s'$.

- Given ground states s and s' s.t. $s \to s'$ using a rule in set (Fif), then there exists a ground substitution τ, variables SS' and IK', and strand $[u_1, \ldots, u_{j-1}, \{T, Num\}, u_{j+1}, \ldots, u_n]$ in P_{CstrSS}, such that $s = \{SS'\tau \& \{IK'\tau\} \& (ro)[u_1\tau, \ldots, u_{j-1}\tau]\}$, and $s' = \{SS'\tau \& \{IK'\tau\} \& [u_1\tau, \ldots, u_{j-1}\tau, \{T\tau, Num\}]\}$ and $T\tau =_{E_\mathcal{P}} true$. Since there exists a substitution θ' s. t. $CstrS' >^{\theta'} s'$, we consider the following two cases:
  - Case e) The strand appears in $S'\theta'$. More specifically, $[u_1\sigma, \ldots, u_{j-1}\sigma, \{T\sigma, Num\} \mid u_{j+1}\sigma, \ldots, u_n\sigma]$ is a strand in S' s.t. $\sigma\theta' =_{E_\mathcal{P}} \tau$. If the constraint T is an equality constraint, since $T\tau =_{E_\mathcal{P}} T\sigma\theta' =_{E_\mathcal{P}} true$, and by the lifting relation, $E_\mathcal{P} \models \Psi'\theta'$, rule (Bif=) can be applied for the backwards narrowing $CstrS' \overset{\mu}{\leftsquigarrow} CstrS$, and $CstrS >^{\theta} s$ such that $\mu\theta =_{E_\mathcal{P}} \theta'$. If the constraint T is a disequality constraint, since $T\tau =_{E_\mathcal{P}} T\sigma\theta' =_{E_\mathcal{P}} true$, and by the lifting relation, $E_\mathcal{P} \models \Psi'\theta'$, we have $E_\mathcal{P} \models T\sigma\theta' \wedge \Psi'\theta'$. Therefore, rule (Bif≠) can be applied for the backwards narrowing, and $CstrS >^{\theta} s$.
  - Case f) The strand does not appear in $S'\theta'$. Then θ' makes S' as a valid symbolic strand state of s, i.e., $S = S'$ and $CstrS' >^{\theta'} s$.
- Given ground strand states s and s' s.t. $s \to s'$ using a rule in set (F?), then we consider the following two applicable cases:
  - Case e) The strand appears in $S'\theta'$ and thus we can perform a backwards narrowing step from $CstrS'$ with rule (B?), i.e., $CstrS' \rightsquigarrow CstrS$, and $CstrS >^{\theta'} s$.
  - Case f) The strand does not appear in $S'\theta'$. Then θ' makes $CstrS'$ as a valid constraint symbolic state of s, i.e., $CstrS = CstrS'$ and $CstrS >^{\theta'} s$.
- Given states s and s' s.t. $s \to s'$ using a rule in set (F?&), the proof is similar with using a rule in the set (F?).

□

B.2 Proof of Theorem 3

We show that Lemma 2 in [7] still holds after extending to constrained states, and therefore, the soundness of the symbolic reachability analysis with respect to the forwards rewriting-based semantics still holds, since the proof of Theorem 3 in Section 7.1 is a straightforward corollary of the lemma below.

Lemma 4. *Given a protocol $\mathcal{P}$, two constrained symbolic states $CstrS = \langle S, \Psi \rangle$ and $CstrS' = \langle S', \Psi' \rangle$, a ground strand state s and a ground substitution θ, if $CstrS \overset{\mu}{\leftsquigarrow} CstrS'$ and $CstrS >^{\theta} s$, then there exists a ground strand state s' and a ground substitution θ' such that $s \to s'$, and $CstrS' >^{\theta'} s'$.*

Lemma 4 is illustrated by the Figure 2.

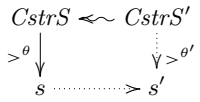

Figure 2. Lemma 4

Proof. We only need to consider the new rules: rule (Bif=), (Bif$\neq$) and (B?).

1) If $CstrS \overset{\mu}{\leftsquigarrow} CstrS'$ using rule (B?), then there are associated rules in the sets (F?) and (F?&).

2) If $CstrS \overset{\mu}{\leftsquigarrow} CstrS'$ using rule (Bif=), there is a strand $[u_1\sigma, \ldots, u_{j-1}\sigma \mid \{(u = v)\sigma, Num\}, u_{j+1}\sigma, \ldots, u_n\sigma]$ in S, $[u_1\sigma', \ldots, u_{j-1}\sigma', \{(u = v)\sigma', Num\} \mid u_{j+1}\sigma', \ldots, u_n\sigma']$ in S' s.t. $\sigma =_{E_\mathcal{P}} \sigma'\mu$, $\Psi =_{E_\mathcal{P}} \Psi'\mu$ and $u\sigma =_{E_\mathcal{P}} v\sigma$, where $[u_1, \ldots, u_{j-1}, \{u = v, Num\}, u_{j+1}, \ldots, u_n]$ is a strand in P_{CstrSS}. Since $CstrS >^{\theta} s$, there is a ground strand $[u_1\sigma\theta, \ldots, u_{j-1}\sigma\theta]$ in s, and $E_\mathcal{P} \models \Psi\theta$. Therefore, $E_\mathcal{P} \models \Psi'\mu\theta$ and $u\sigma\theta =_{E_\mathcal{P}} v\sigma\theta$. By rule (Fif), $s \to s'$, and $CstrS' >^{\mu\theta} s'$.

If $CstrS \overset{\mu}{\leftsquigarrow} CstrS'$ using rule (Bif$\neq$), there is a strand $[u_1\sigma, \ldots, u_{j-1}\sigma \mid \{(u \neq v)\sigma, Num\}, u_{j+1}\sigma, \ldots, u_n\sigma]$ in S , $[u_1\sigma', \ldots, u_{j-1}\sigma', \{(u \neq v)\sigma', Num\} \mid u_{j+1}\sigma', \ldots, u_n\sigma']$ in S' s.t. $\sigma =_{E_\mathcal{P}} \sigma'\mu$ and $\Psi =_{E_\mathcal{P}} \Psi'\mu \wedge (u \neq v)\sigma'\mu$, where $[u_1, \ldots, u_{j-1}, \{u \neq v, Num\}, u_{j+1}, \ldots, u_n]$ is a strand in P_{CstrSS}. Since $CstrS >^{\theta} s$, there is a ground strand $[u_1\sigma\theta, \ldots, u_{j-1}\sigma\theta]$ in s, and $E_\mathcal{P} \models \Psi\theta$. Therefore, $E_\mathcal{P} \models \Psi'\mu\theta \wedge (u \neq v)\sigma'\mu\theta$. By rule (Fif), $s \to s'$, and $CstrS' >^{\mu\theta} s'$. □