Automated Reasoning Workshop 2011

Department of Computing Science
University of Glasgow
April 11th – 12th

Proceedings

Proceedings of ARW 2011

Editors: Alice Miller and Ryan Kirwan

This volume contains the proceedings of the Automated Reasoning Workshop 2011 which was held at the University of Glasgow on 11th-12th April 2011.

Guest Speakers were Colin Stirling (University of Edinburgh) and Patricia Johann (University of Strathclyde) who gave fascinating talks on *Using Game Theory for Higher Order Matching* and *Principled Ways of Deriving Induction Rules for Data Types.*

This workshop series provides an informal forum for the automated reasoning community to discuss recent work, new ideas and current trends. It aims to bring together researchers from all areas of automated reasoning in order to foster links and facilitate cross-fertilisation of ideas among researchers from various disciplines; among researchers from academia, industry and government; and between theoreticians and practitioners.

The series covers the full breadth and diversity of automated reasoning, including topics such as logic and functional programming; equational reasoning; deductive databases; unification and constraint solving; the application of formal methods to specifying, deriving, transforming and verifying computer systems and requirements; deductive and non-deductive reasoning, including abduction, induction, non-monotonic reasoning, and analogical reasoning; commonsense reasoning; and the wide range of topics that fall under the heading of knowledge representation and reasoning.

The workshops in this series are highly interactive, giving all attendees the opportunity to participate. The workshops contain many open discussion sessions organised around specific topics, and often contain sessions for displaying posters and presenting system demonstrations. This is intended to be an inclusive workshop, with participants encouraged from the broad spectrum covered by the field of automated reasoning. We encourage the participation of experienced researchers as well as those new to the field, especially students.

Contents

Abstraction for Model Checking Robot Behaviour

Ryan Kirwan
Department Of Computing Science
University Of Glasgow
kirwanr@dcs.gla.ac.uk

Alice Miller
Department Of Computing Science
University Of Glasgow
alice@dcs.gla.ac.uk

Abstract

We use model checking to verify properties of real systems. These systems consist of robots interacting with obstacles within an environment and learning to avoid collision. We describe an approach to abstract environments so that all feasible scenarios are represented, and properties are still applicable. The abstraction is necessary for the verification (and ultimately, comparison) of learning algorithms and, although currently applied to Promela specifications, is applicable to other specification formalisms.

1 Introduction

Traditionally, testing and simulation have been used to validate robot behaviour and algorithms defining robot learning have been measured by repeated simulation. Model checking [1] robot behaviour allows us to formally check properties and quantify the performance of different learning algorithms. The aim of this paper is to establish model checking as a valid tool within this context.

Our initial models are specified in Promela [3] and checked using SPIN [3]. Ultimately, we aim to use other model checkers such as PRISM [2] and Uppal [5] to allow the checking of qualitative properties. However, our current objective is to define appropriate abstractions of the models to be specified; these abstractions will be relevant for all cases.

Using SPIN we model the behaviour of a robotic agent which learns while interacting with an environment. By modelling existing systems, which involve a robotic agent [4], we hope to be able to recreate results that were obtained by simulation and experimentation of a physical system. The robot we are modelling learns to avoid obstacles in a boundless environment. It has 2 long range "distal" sensors and 2 short range "proximal" sensors; distal sensors are used to avoid collisions, proximal sensors are used to detect collisions.

1.1 Robot Behaviour

The robot behaviour captured in our models is as defined in [4]. This work looks at how learning is affected by an environment and by the perception of the learner. Through simulations, the ability of the robot to successfully navigate its environment is used to assess the learning algorithm used and the robot's sensor configuration. This field of research relies on simulations to obtain system properties, which are generated by averaging results from sets of simulated experiments. By applying model checking we can derive properties by a single search of the state-space (per property).

One of the key factors when creating a model of these systems is the learning of the robot. Learning is greatly simplified in the model in order to reduce the state-space of the system. A 'K' value is used to represent how much the robot has learned and is increased as the robot learns more. This value directly affects the magnitude of response when the robot encounters an obstacle. Whenever the robot collides with an obstacle the 'K' value is incremented; this increment causes a larger response to distal sensor signals.

1.2 Representing the System

There are restrictions on the type of system we can model. An obstacle has a fixed size and shape, as does the robot; the environment is considered a boundless space. The "complexity" of the environment is defined by its concentration of obstacles. The robot's distal sensors send signals of varying strength depending on how close or far away an obstacle is to the robot. The robot moves constantly, changing its orientation to avoid obstacles as it moves. If the robot cannot move forward it will continue to change its orientation until it can proceed.

Obvious problems when representing this system is having a boundless environment to model and that for a given complexity there are many valid environments. To model this set of boundless environments we have to use a suitable abstraction. By limiting what is derived from the model to information on how the robot responds to obstacles we can create an effective abstraction. This abstraction can be visualized in the Polar model in Fig. 1 (left).

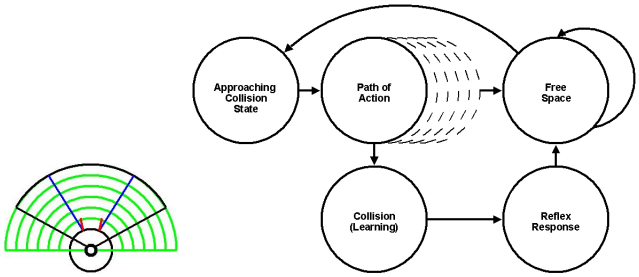

Figure 1: Polar model and State transition diagram.

We refer to the area in front of the robot as a ”cone of influence“, this is where all relevant system interactions take place. There are two main ideas behind this abstraction. First, is that the robot only interacts with obstacles when they contact it's sensors and second, that the robot's coordinates are irrelevant. Storing the robot's coordinates would yield an infinite state-space. Therefore, our model moves the environment around the robot instead. Using this cone of influence representation we can model all permutations of obstacles interacting with the robot. We derive the possible permutations from the complexity of the environment by considering the distance between obstacles. In this way we can represent a set of environments, of a fixed complexity, in one model. This can be visualized in Fig. 2, below.

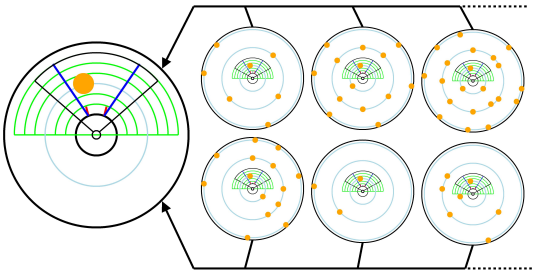

Figure 2: Combining environments

All possible permutations of obstacle collisions are considered in one polar model; this covers all possible encounters in a set of environments of a given complexity. The abstracted state-space is represented in Fig. 1 (right). An approaching collision state is one of the possible permutations of obstacles that can approach the robot and each has a unique path of action based on what the robot has learned. Initial experiments with SPIN have shown the state-space to be tractable.

1.3 Future Work

We plan to use PRISM to create similar system models to verify quantitative properties. Principally we want to develop various models, classifying the benefits and costs of each approach. One further aim is to develop a custom-made tool to model systems that involve agent-based learning.

References

[1] Jr. Edmund M. Clarke, Orna Grumberg, and Doron A. Peled. *Model Checking*. The MIT Press, February 2000.

[2] A. Hinton, M. Kwiatkowska, G. Norman, and D. Parker. Prism: A tool for automatic verification of probabilistic systems. *LNCS*, 3920, 2006.

[3] G. J. Holzmann. *The SPIN Model Checker: Primer and Reference Manual*. Addison-Wesley Pearson Education, 2004.

[4] Tomas Kulvicius, Christoph Kolodziejski, Tomas Minija Tamosiunaite, Bernd Porr, and Florentin Worgotter. Behavioral analysis of differential hebbian learning in closed-loop systems. 2009.

[5] Kim G. Larsen, Paul Patterson, and Wang Yi. Uppaal in a nutshell. *International Journal on Software Tools for Technology Transfer (STTT)*, 1997.

METTEL: A Generic Tableau Prover*

Dmitry Tishkovsky*
*University of Manchester
dmitry@cs.man.ac.uk

Renate A. Schmidt†
†University of Manchester
schmidt@cs.man.ac.uk

Mohammad Khodadadi‡
‡University of Manchester
khodadadi@cs.man.ac.uk

Abstract

METTEL is a generic tableau prover for various modal, description, hybrid, intuitionistic and metric logics. The core component of METTEL is a logic-independent tableau inference engine. A novel feature is that users have the ability to specify the set of tableau rules to be used in derivations. Termination can be achieved via a generalisation of a standard loop checking mechanism, or unrestricted blocking.

1 Introduction

METTEL is a lightweight generic tableau prover implemented in JAVA. Its main purpose is to provide a tool for experimenting with tableau calculi for newly invented non-classical logics for which there are no known sound and complete decision procedures. It enforces termination with generic loop-checking and the unrestricted blocking mechanism introduced in Schmidt and Tishkovsky (2007).

The first versions of METTEL were designed for deciding logics of metric and topology (Hustadt et al., 2006). In 2005, when the implementation started, there were no known tableau calculi for these logics. Much effort was invested in a generic, modular design of METTEL, which led to quick implementations of tableau decision procedures for many other non-classical logics including, for example, intuitionistic propositional logic, the expressive description logic $\mathcal{ALBO}$ (Schmidt and Tishkovsky, 2007), and the extension $\mathcal{ALBO}^{\mathsf{id}}$ with role identity. These tableaux have been implemented and provided as predefined tableau calculi in METTEL.

Most recently, we have extended METTEL with a rule specification language giving users the facility to define the tableau rules to be used as input. Also, Skolem terms have replaced the use of nominals in the rules for expanding existential quantification formulae.

METTEL decides the following logics: classical and intuitionistic propositional logic, hybrid logic HL(@,U) with the universal modality, the logic $\mathcal{MT}$ of metric and topology, and all sublogics of the description logic $\mathcal{ALBO}^{\mathsf{id}}$. At the moment, METTEL is the only tableau prover that can decide logics of metric and topology and description logics with role negation. Using the facility to specify tableau rules, METTEL can be used as a prover for many other more expressive or even undecidable logics.

Most closely related to METTEL are the prover engineering platforms LOTREC (Gasquet et al., 2005) and the Tableau Work Bench (Abate and Gore, 2003). Although METTEL does not give the user the same possibilities for control and programming as these systems, the rule specification language is more expressive. For example, Skolem terms are allowed both in premises and conclusions of rules.

2 METTEL language

The language for specifying formulae is a many-sorted language with five sorts, namely, formulae, relational formulae (or roles), nominal terms including Skolem terms, attributes and rational parameters from metric logic. The available predefined connectives include negation `~`, conjunction `&`, disjunction `|`, logical equivalence `<->`, implication `->`, modal `box` and `dia` (synonyms are description logic `forall` and `exists`), the 'closer' operator `<<` of similarity logic, the `@` operator of hybrid logic, and the relational operators: inverse `-`, union `+`, intersection `&`, negation `~`, composition `;`, and reflexive-transitive closure `*`. The language also includes the propositional constants `FALSE` and `TRUE` and allows arbitrary names for relational constants. For example, although relational identity `id` is not a predefined symbol it can be encoded in the rule specification language. Here are three examples of formulae specifiable in METTEL:

```
box box{~(~R | S)}FALSE
(@i dia{R}j & @j dia{R}k) -> @i dia{R}k
exists{a<x;a<y}P -> exists{a<(x+y)}P
```

The first formula encodes role inclusion $R \sqsubseteq S$ in $\mathcal{ALBO}^{\mathsf{id}}$. The second formula defines R as a transitive relation in hybrid logic. The last formula expresses the triangle property of metric relations in metric logic.

The tableau rule specification language is designed so that it closely mirrors the notation used in the literature. Sample rules are:

```
@i (P|Q) / { @i P } | { @i Q };
@i ~(dia{R} P) @i dia{R} j / { @j ~P };
@i dia{R}P /
   { @f[i,R,P] P, @i dia{R} f[i,R,P] };
@i P, @i ~P / { FALSE };
```

The premises and conclusions are separated by `/` and each rule is terminated by `;`. The first rule is the or rule, defined

*The work is supported by research grant EP/H043748/1 of the UK EPSRC.

as a branching rule which creates a splitting point adding `@i P` to the left branch and `@i Q` to the right branch. The other rules are non-branching rules. The third rule is the standard box rule rewritten in terms of the negated operator `dia`. Note the use of the Skolem term `f[i,R,P]` in the next rule; it represents the witness created, depending on `i,R,P`. Rules, like the last rule, with `FALSE` belonging to the conclusion are closure rules.

3 Implementation

All expressions are internally represented as extensions of the basic MettelExpression interface. For efficiency reasons, each kind of expression is represented as a separate JAVA class, which is not parametrised by connectives. At runtime, the creation of expression objects is managed by means of a factory pattern implemented as MettelObjectFactory and ensures that each expression is represented by a single object.

Each class implementing the MettelExpression interface is required to implement two methods. (i) A method for matching the current object with the expression object supplied as a parameter, returning the matching unifier. (ii) A method to return an instance of the current expression with respect to a given substitution.

Every node of the tableau is represented as a tableau state object comprising of a set of formulae associated with the node and methods for manipulating the formulae and realising rule applications. Each derivation step consists of choosing a rule and applying it to the set of active formulae associated with it by matching its premises. If the rule is branching, the formulae of each branch are added to a corresponding copy of the current tableau state object. If the rule is non-branching, the conclusions of the rule are added to the current state without copying. If a closure rule is applied then the state is immediately discarded and the derivation returns to the parent state, thus, discarding the contradictory branch. The next branch is then explored. The derivation returns to the previous state also in the case if there are no branches to explore. As soon as all the rules have been applied within the branch and there are no more active formulae, the procedure returns the formulae on the branch. This represents a model for the input problem relative to the tableau calculus used and means the problem is satisfiable. If all branches have been found to be contradictory the problem is unsatisfiable.

Since matching is computationally expensive, it is performed only once for any given formula and a premise of a rule we try to associate. This is achieved by maintaining sets of all the substitutions obtained from matching the selected formula with the rule premise. If the tuple of the selected formulae match the premises of the rule, the resulting substitution object is passed to the conclusions of the rule. The result of a rule application is a set of branches which are sets of formulae obtained by applying the substitution to the conclusions of the rule.

Two blocking mechanisms are implemented in METTEL for detecting 'loops' in derivations. The first blocking mechanism is a generalisation of anywhere equality blocking. Unlike other blocking techniques, it does not require all local rules to be applied before rules that introduce new nominal terms. Instead, it keeps track of formulae that depend on a given nominal term for every rule. Once no more local rules are applicable to any pair of nominals, the blocking test is performed for the nominals. If successful, an equality is added to the current tableau state for subsequent rewriting. The second available blocking mechanism is the unrestricted blocking mechanism (Schmidt and Tishkovsky, 2007). This is realised through a branching rule which performs a case distinction of identifying two nominal terms, or not.

4 Concluding remarks

METTEL gives users the option of either using one of the several predefined tableau calculi or define their own set of rules to be used for constructing derivations. In order to provide a more comprehensive tool to users, we intend to incorporate the tableau calculus synthesis framework described in Schmidt and Tishkovsky (2009) into METTEL. Then, the user would be able to construct an automated theorem prover for a specific logic by defining its syntax and semantics.

METTEL is available from `http://www.mettel-prover.org`.

References

P. Abate and R. Gore. The tableaux work bench. In *Automated Reasoning with Analytic Tableaux and Related Methods*, volume 2796 of *Lecture Notes in Computer Science*, pages 230–236. Springer, 2003.

O. Gasquet, A. Herzig, D. Longin, and M. Sahade. LoTREC: Logical tableaux research engineering companion. volume 3702, pages 318–322, 2005.

U. Hustadt, D. Tishkovsky, F. Wolter, and M. Zakharyaschev. Automated reasoning about metric and topology. In *Logics in Artificial Intelligence*, volume 4160 of *Lecture Notes in Computer Science*, pages 490–493. Springer, 2006.

R. Schmidt and D. Tishkovsky. Using tableau to decide expressive description logics with role negation. volume 4825, pages 438–451. 2007. ISBN 978-3-540-76297-3.

R. Schmidt and D. Tishkovsky. Automated synthesis of tableau calculi. In *Automated Reasoning with Analytic Tableaux and Related Methods*, volume 5607 of *Lecture Notes in Computer Science*, pages 310–324. Springer, 2009.

Smart Semantic Tree Theorem Prover

Nourah A. S. Shenaiber

The University of Birmingham
School of Computer Science
N.A.Shenaiber@cs.bham.ac.uk

Abstract

Semantic trees have long been used as a theoretical tool for verifying the unsatisfiability of sets of clauses. However, on a class of problems they can be used to efficiently prove the unsatisfiability of clauses in first order predicate logic. The exact proof depends on the Herbrand Base set used to build the semantic tree. The Herbrand Base (HB) is generated with the help of the Herbrand Universe (HU), which in turn comes from the base set of clauses in a given theorem. In order to effectively build a closed semantic tree, it is necessary to make a smart selection of atoms from the Herbrand Base (i.e. that is, it is necessary to make an educated guess). Proving unsatisfiability using semantic trees (Herbrand's procedure) suffers from a major drawback because this method requires successively generating sets of ground instances of the clauses in S and then successively testing these sets for unsatisfiability. In order to keep these tests as small as possible, we propose a new approach. Instead of generating the ground instances of the clauses in S in some arbitrarily defined order, the algorithm computes ground instantiations in an intelligent way. This way it is possible to show the unsatisfiability in a more efficient way.

1 Introduction

Herbrand gave a very important approach to mechanical theorem proving in 1930. By definition, a valid formula is a formula that is TRUE under all interpretations. Herbrand developed an algorithm to find an interpretation that can falsify a given formula. This method is the basis for most modern automatic proof procedures.

In this abstract, an effective technique for building a semantic tree theorem prover named SSTTP (Smart Semantic Tree Theorem Prover) is presented. SSTTP uses different ways to produce atoms of the Herbrand base. We present a new technique for producing Herbrand base atoms. It is better than listing Herbrand base atoms in canonical order as described in (2). This technique gave rise to the Smart Herbrand Base.

2 SSTTP idea

The problem is how to make a smart selection of atoms from the Herbrand Base to build a closed semantic tree in a more efficient way. The researcher in this field tries to select the proper atoms from the Herbrand Base by providing some heuristic techniques during the construction of the tree. So it is interesting if someone can prove that there is a subset of the Herbrand Base for any problem to be solved by constructing a short closed semantic tree.

The algorithm that generates the Smart Herbrand Base (SHB) is motivated by the Set-of-Support strategy. It starts like the original SoS with the base set of clauses in $S = A_1 \wedge A_2 \wedge \ldots \wedge A_n \wedge \neg B$. But then, it has a different way of using the resolution technique to generate a SHB in the aim to build closed semantic trees in an effective way. The algorithm uses the resolution only to check if the given two predicates resolve with each

other or not. The steps of the algorithm are shown next:

1. Put the clause(s) of the negated conclusion (denoted by $\neg B$) in front of axioms, $\neg B \wedge A_1 \wedge A_2 \wedge \cdots \wedge A_n$.
2. Then try to resolve the first literal in $\neg B$ denoted by L_1 with all literals in axioms after it. If a resolvent occurs then put the literal L_1 (positive predicate) in the updated HB after checking that this literal L_1 is a ground predicate (i.e. it does not have any variables). If it has variables then put them in the set after giving them place holder variables temporarily until they will be substituted by an element from the Herbrand Universe when the program uses them in the semantic tree.
3. If no resolvent occurs then jump to the next literal L_2 in $\neg B$. If there are no more literals in $\neg B$, then jump to next clause A_i after $\neg B$. If there is no clause to jump to (i.e. end of file), then exit (stop algorithm) else repeat step 2 with considering A_i as $\neg B$ now.
4. Delete the literal L_j on which the resolution step was based from $\neg B$ and literals that resolved with it from the axioms.
5. Keep repeating the above four steps as long as the Base Set is not empty.

The outcome of this algorithm is a smart ordering (noncanonical ordering) of atoms in the Herbrand base (SHB) which leads to a faster construction of closed semantic trees for some tested problems. Let us consider the following simple example, let S be the following set of clauses:
$\{\neg M(f(a)), \neg H(X) | M(X), \neg H(X) | H(f(X)), H(a)\}$ SHB = $\{H(a), M(f(a)), H(f(a))\}$

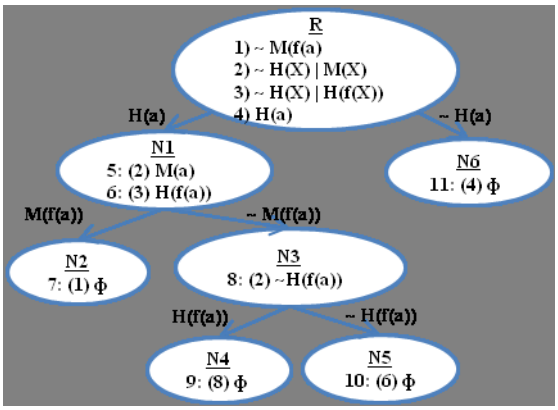

Figure 1: Closed semantic tree of example

References

[1] Chang, C., and Lee, R. (1973). Symbolic Logic and Mechanical Theorem Proving, Academic Press.

[2] Newborn, M. (2001). Automated Theorem Proving: Theory and Practice, Springer-Verlag, New York.

Proof-Theoretic Soundness and Completeness

Robert Rothenberg*
*School of Computer Science, University of St Andrews, St Andrews, Fife KY16 9SX, Scotland, UK
rr@cs.st-andrews.ac.uk

Abstract

We give a calculus for reasoning about the first-order fragment of classical logic that is adequate for giving the truth conditions of intuitionistic Kripke frames, and outline a proof-theoretic soundness and completeness proof, which we believe is conducive to automation.

1 A Semantic Calculus for Intuitionistic Kripke Models

In Rothenberg (2010), we use correspondence theory (Blackburn et al., 2001) to give a cut-free calculus for reasoning about intuitionistic Kripke models (Kripke, 1965) using a fragment of first-order classical logic.

Definition 1 (Partially-Shielded Formulae)**.** We define the partially-shielded fragment (PSF) of first-order formulae: (1) $\bot$; (2) $P\{x\}$ iff P is an atomic propositional variable, or an atomic first-order formula with a free variable x; (3) $A\{x\} \wedge B\{x\}$ and $A\{x\} \vee B\{x\}$, iff $A\{x\}, B\{x\}$ are in PSF; (4) $\mathcal{R}xy$, where $\mathcal{R}$ is a *fixed* atomic binary relation (5) $\forall y.(\mathcal{R}xy \wedge A\{y\}) \rightarrow B\{y\}$, iff $A\{x\}$ and $B\{x\}$ are in PSF.

We give the calculus $\mathbf{G3c/PSF}$ in Figure 1, which is useful for reasoning about sequents of formulae in PSF. A variant of it was introduced in Rothenberg (2010), based on ideas from a calculus for the guarded fragment (GF) of first-order formulae given in Dyckhoff and Simpson (2006).

$$\frac{}{\Gamma, P \Rightarrow P, \Delta}\ \mathsf{Ax} \qquad \frac{}{\Gamma, \bot \Rightarrow \Delta}\ \mathsf{L}\bot$$

$$\frac{\Gamma, A, B \Rightarrow \Delta}{\Gamma, A \wedge B \Rightarrow \Delta}\ \mathsf{L}\wedge \qquad \frac{\Gamma \Rightarrow A, \Delta \quad \Gamma \Rightarrow B, \Delta}{\Gamma \Rightarrow A \wedge B, \Delta}\ \mathsf{R}\wedge \qquad \frac{\Gamma, A \Rightarrow \Delta \quad \Gamma, B \Rightarrow \Delta}{\Gamma, A \vee B \Rightarrow \Delta}\ \mathsf{L}\vee \qquad \frac{\Gamma \Rightarrow A, B, \Delta}{\Gamma \Rightarrow A \vee B, \Delta}\ \mathsf{R}\vee$$

$$\frac{\Gamma, \mathcal{R}xz, \forall y. \ldots \Rightarrow A\{z\}, \Delta \quad \Gamma, \mathcal{R}xz, \forall y. \ldots, B\{z\} \Rightarrow \Delta}{\Gamma, \mathcal{R}xz, \forall y.(\mathcal{R}xy \wedge A\{y\}) \rightarrow B\{y\} \Rightarrow \Delta}\ \mathsf{L}\forall \rightarrow \qquad \frac{\Gamma, \mathcal{R}xz, A\{z\} \Rightarrow B\{z\}, \Delta}{\Gamma \Rightarrow \forall y.(\mathcal{R}xy \wedge A\{y\}) \rightarrow B\{y\}, \Delta}\ \mathsf{R}\forall \rightarrow$$

Figure 1: The calculus $\mathbf{G3c/PSF}$ for sequents of partially shielded formulae.

In Figure 1, that the variable y is fresh for the conclusion of the $\mathsf{R}\forall \rightarrow$ rule, and that $\forall y. \ldots$ in the premisses of the $\mathsf{L}\forall \rightarrow$ and $\mathsf{R}\forall \rightarrow$ rules is an abbreviation of "$\forall y.(\mathcal{R}xy \wedge A\{y\}) \rightarrow B\{y\}$".

Proposition 1 (Standard Structural Rules, (Rothenberg, 2010))**.** *The following rules are admissible in* $\mathbf{G3c/PSF}$*:*

$$\frac{\Gamma \Rightarrow \Delta}{\Gamma, \Gamma' \Rightarrow \Delta', \Delta}\ \mathsf{W} \qquad \frac{\Gamma, \Gamma', \Gamma' \Rightarrow \Delta', \Delta', \Delta}{\Gamma, \Gamma' \Rightarrow \Delta', \Delta}\ \mathsf{C} \qquad \frac{\Gamma \Rightarrow \Delta, A \quad A, \Gamma' \Rightarrow \Delta'}{\Gamma, \Gamma' \Rightarrow \Delta', \Delta}\ \mathsf{Cut}$$

Proposition 2 ((Negri, 2007))**.** *Let* $\mathbf{G3c/PSF}^*$ *be* $\mathbf{G3c/PSF}$ *plus the following (geometric) rules:*

$$\frac{\mathcal{R}xx, \Gamma \Rightarrow \Delta}{\Gamma \Rightarrow \Delta}\ \mathsf{refl} \qquad \frac{\mathcal{R}xz, \mathcal{R}xy, \mathcal{R}yz, \Gamma \Rightarrow \Delta}{\mathcal{R}xy, \mathcal{R}yz, \Gamma \Rightarrow \Delta}\ \mathsf{tran} \qquad \frac{\mathcal{R}xy, Px, Py, \Gamma \Rightarrow \Delta}{\mathcal{R}xy, Px, \Gamma \Rightarrow \Delta}\ \mathsf{mono}$$

where Px, Py *in the* mono *rule are atomic.*

Corollary 3 ((Negri, 2007))**.** *The standard structural rules (Proposition 1) are admissible in* $\mathbf{G3c/PSF}^*$.

Remark 1. The labelled sequent calculus **G3I** Negri (2007) can be thought of as an alternative form of $\mathbf{G3c/PSF}^*$ which hides the quantifiers.

Definition 2 (Translation of Propositional Formulae into PSF)**.**

$$\bot^\dagger =_{def} \bot \qquad (A \wedge B)^\dagger =_{def} A^\dagger \wedge B^\dagger \qquad (A \to B)^\dagger =_{def} \forall y.(\mathcal{R}xy \wedge A^\dagger) \to B^\dagger$$
$$P^\dagger =_{def} \hat{P}x \qquad (A \vee B)^\dagger =_{def} A^\dagger \vee B^\dagger$$

where the translation of $A \to B$ requires that the free variable of $A^\dagger, B^\dagger$ is x, and $y \neq x$, and $\hat{P}x$ uniquely corresponds to P. Recall that $\mathcal{R}$-formulae occur only as strict subformulae in the translation. The extension is adapted to sequents naturally, where all formulae have the same free variable.

Theorem 4 (Soundness and Completeness, (Rothenberg, 2010))**.** *Let* $\mathfrak{M} = \langle W, R, \Vdash \rangle$ *be a Kripke model for* **Int**. *Then* $\mathfrak{M} \vDash \Gamma \Rightarrow \Delta$ *iff* $\mathbf{G3c/PSF}^* \vdash \Gamma^\dagger \Rightarrow \Delta^\dagger$.

Theorem 5. *Let* $\mathbf{G}$ *be a multisuccedent sequent calculus for* **Int***, e.g.* $\mathbf{m\text{-}G3ip}$ *(Troelstra and Schwichtenberg, 2000). Then* $\mathbf{G} \vdash \Gamma \Rightarrow \Delta$ *iff* $\mathbf{G3c/PSF}^* \vdash \Gamma^\dagger \Rightarrow \Delta^\dagger$.

Proof. By induction on the derivation height. An outline of the proof is as follows: (1) Hyperextend (Avron, 1991) $\mathbf{G}$ to a hypersequent calculus $\mathbf{HG}$; (2) Show $\mathbf{G} \vdash \Gamma \Rightarrow \Delta$ iff $\mathbf{HG} \vdash \Gamma \Rightarrow \Delta$ (straightforward). (3) Extend Definition 2 so that components in hypersequents are translated with unique free variables; (4) Show $\mathbf{HG} \vdash \Gamma \Rightarrow \Delta$ iff $\mathbf{G3c/PSF}^* \vdash (\Gamma \Rightarrow \Delta)^\dagger$. (Note that instances of mono can be eliminated from $\mathbf{G3c/PSF}^*$ proofs of sequents with a single free variable.) □

2 Future Work

We expect that adapting this work to single-succedent calculi, e.g. $\mathbf{G3ip}$ (Troelstra and Schwichtenberg, 2000), should be straightforward. An obvious extension is to adapt this work to hypersequent calculi for superintuitionistic logics, e.g. in (Avron, 1991). For logics with geometric Kripke semantics (Rothenberg, 2010), this should be straightforward. This work can be adapted to cut-free sequent calculi for modal logics in a straightforward manner, using similar calculi for guarded formulae, such as Dyckhoff and Simpson (2006).

Adapting this work to non-Gentzen calculi, such as nested sequents (Brünnler, 2009) should be possible, by applying translations of their data structures into sequents of partially-shielded formulae, and using a limited form of proof search on schematic rules. (Such work may be easier, if the data structure allows relations between points in a Kripke frame to be explicit.)

Acknowledgements

We'd like to thank Roy Dyckhoff for providing a copy of Dyckhoff and Simpson (2006), and for helpful suggestions regarding $\mathbf{G3c/PSF}$.

References

A. Avron. Hypersequents, logical consequence and intermediate logics for concurrency. *Ann. Math. Artif. Intell.*, 4(3-4): 225–248, September 1991. ⟨doi:10.1007/BF01531058⟩.

P. Blackburn, M. de Rijke, and Y. Yenema. *Modal Logic*. Cambridge U.P., 2001.

K. Brünnler. *Nested Sequents*. Habilitation, Institut für Informatik und angewandte Mathematik, Universität Bern, 2009.

R. Dyckhoff and A. Simpson. Proof theory of guarded logics, 2006. Unpublished Ms.

S. Kripke. Semantical analysis of intuitionistic logic I. In J. Cossley and M. Dummett, editors, *Formal Systems and Recursive Functions*, pages 92–129. North Holland, Amsterdam, 1965.

S. Negri. Proof analysis in non-classical logics. In C. Dimitracopoulos, L. Newelski, D. Normann, and J. Steel, editors, *Logic Colloquium 2005*, volume 28 of *Lecture Notes in Logic*, pages 107–128. Cambridge U.P., 2007.

R. Rothenberg. *On the relationship between hypersequent calculi and labelled sequent calculi for intermediate logics with geometric Kripke semantics*. PhD thesis, University of St Andrews, St Andrews, Scotland, June 2010.

A.S. Troelstra and H. Schwichtenberg. *Basic Proof Theory*. Cambridge U.P., 2000.

Metamathematics based on nominal terms and nominal logic: foundations of a nominal theorem-prover

Murdoch J. Gabbay*
*Heriot-Watt University

Abstract

We propose nominal terms as the syntax of a theorem-prover, instead of first- or higher-order terms.

Then term-formers can bind, so we can handle binding, but we do not take syntax up to β-equivalence or put functions directly into our syntax.

This gives us expressivity to axiomatise and reason about higher-order data, the π-calculus, locality, substitution, specification languages with binding—whose common attribute is the use of binding—but our theorem-prover remains based on a first-order framework.

1 Nominal terms

Binding naturally appears for instance in axiomatisations involving locality, name-restriction, functions, substitution, and instantiation. Here are examples:

$$\begin{array}{ll} \int_x f(x) = \int_y f(y) & (\forall x.\phi)[x::=t] = \forall x.(\phi[x::=t]) \quad \text{provided that } x \text{ is not free in } t \\ \lambda x.f(x) = \lambda y.f(y) & P \mid \nu a.Q = \nu a.(P \mid Q) \quad \text{provided that } a \text{ is not free in } P \end{array}$$

In addition, we often want to reason intensionally about inequality of names (e.g. when specifying α-inequality) or about renaming names (e.g. when α-converting).

If we use higher-order terms then binding is identified with functional abstraction via λ-abstraction. Thus $\lambda x.f(x)$ is a term denoting 'input x, then output $f(x)$', and $\int$, $\forall$, and ν are considered as higher-order constants. This is the approach taken e.g. by Isabelle, HOL, and mCRL2 (and also by Coq). But this introduces functions directly as primitive in the meta-language. Furthermore, if names are identified with functional abstraction then we have no intensional access to the name of a variable; only to the functional argument it represents. We invited binding into our syntax; we did not necessarily want to invite its friends, simple type theory and even set theory, to the party too.

I argue that nominal terms give us the power to reason intuitively about binding, and intensionally on names, while remaining first-order. Recent advances have extended nominal terms with first-order quantification. Thus, a theorem-prover based on nominal terms would be a practical basis for mechanised mathematics.

2 An example: axiomatise functional abstraction

Here, for instance, are axioms for the λ-calculus as they might be specified in a nominal terms theorem-prover e.g. based on nominal algebra Gabbay and Mathijssen (2009):

$$\begin{array}{lll} (\beta\mathbf{var}) & & (\lambda a.a)X = X \\ (\beta\#) & a\#Z \vdash & (\lambda a.Z)X = Z \\ (\beta\mathbf{app}) & & (\lambda a.(Z'Z))X = ((\lambda a.Z')X)((\lambda a.Z)X) \\ (\beta\mathbf{abs}) & b\#X \vdash & (\lambda a.(\lambda b.Z))X = \lambda b.((\lambda a.Z)X) \\ (\beta\mathbf{id}) & & (\lambda a.Z)a = Z \end{array}$$

Points to note here are:

- λ is just a term-former, like application, and $\lambda a.X$ decomposes as λ applied to a *nominal atoms-abstraction* $a.x$ which is a mathematically strictly smaller and simpler structure than a function.
- We have just axiomatised functions. From a 'nominal' point of view, this is just an axiomatic theory. Axioms can express arbitrary complexity, if we want it.
- $a\#Z$ and $b\#X$ are freshness side-conditions corresponding to conditions like 'provided x is not free in t'. Nominal terms axioms tend to closely parallel informal axioms; reasoning in a theorem-prover based on nominal terms is likely to be intuitive.

For completeness, here are the axioms above written out again as they might be written informally:

$$\begin{array}{lll}
(\lambda x.x)r & = r & \\
(\lambda x.t)r & = t & \text{provided that } x \text{ is not free in } t \\
(\lambda x.(t't))r & = ((\lambda x.t')r)((\lambda x.t)r) & \\
(\lambda x.(\lambda y.t))r & = \lambda y.((\lambda x.t)r) & \text{provided that } y \text{ is not free in } r \\
(\lambda x.t)x & = t &
\end{array}$$

This kind of example invites us to think afresh about what it is we are doing when we axiomatise a system with binding, and about the relationship between syntax and semantics. Axiomatisations in nominal terms are different from those using higher-order terms, but in what I would argue is a good way: unfamiliar perhaps, but reasonable.

3 Two more examples: inequality of names and on-the-fly renaming

I mentioned that nominal terms are more expressive than higher-order terms. Here are two simple examples of how:

$$a \neq b \qquad\qquad \phi \Rightarrow \pi{\cdot}\phi$$

The first example observes that the name a is not equal to the name b. Nominal terms permit direct intensional reasoning on names. Higher-order terms cannot not permit this, since variables can always be bound by an outer λ-abstraction (in other words: 'variables vary; names do not').

The second example is a form of α-equivalence, or *equivariance*. Whenever we write 'choose some variables x and y, and prove ϕ', we are using equivariance implicitly to say 'where it does not matter what the names of x and y are'. Higher-order terms cannot permit direct reasoning on renaming variables—once the user has chosen names for variables those names are fixed and they cannot be accessed from within the system.[1]

4 The theorems so far

So far, we have rewriting, equality, and first-order frameworks, called *nominal rewriting*, *nominal algebra*, and *permissive-nominal logic* (Fernández and Gabbay, 2007; Gabbay and Mathijssen, 2009; Dowek and Gabbay, 2010).

We have used them to axiomatise substitution, the λ-calculus, first-order logic, and arithmetic. But that is not all: we have also proved these theories *correct* (Gabbay and Mathijssen, 2008a, 2010, 2008b; Dowek and Gabbay, 2010). That is, the axioms do not just *look* like they axiomatise what they look like they axiomatise, but it has also been proved by arguments on models that they *do* axiomatise what they look like they axiomatise.

Since substitution, the λ-calculus, first-order logic, and arithmetic are a basis for computer science—at least in theory—this can be read as a proof-of-concept for nominal mechanised mathematics. Put another way: a nominal theorem-prover is feasible.

References

Gilles Dowek and Murdoch J. Gabbay. Permissive Nominal Logic. In *Proceedings of the 12th International ACM SIGPLAN Symposium on Principles and Practice of Declarative Programming (PPDP 2010)*, pages 165–176, 2010. doi: http://dx.doi.org/10.1145/1836089.1836111.

Maribel Fernández and Murdoch J. Gabbay. Nominal rewriting (journal version). *Information and Computation*, 205(6): 917–965, June 2007. doi: http://dx.doi.org/10.1016/j.ic.2006.12.002.

Murdoch J. Gabbay and Aad Mathijssen. Capture-Avoiding Substitution as a Nominal Algebra. *Formal Aspects of Computing*, 20(4-5):451–479, June 2008a. doi: http://dx.doi.org/10.1007/11921240_14.

Murdoch J. Gabbay and Aad Mathijssen. One-and-a-halfth-order Logic. *Journal of Logic and Computation*, 18(4):521–562, August 2008b. doi: http://dx.doi.org/10.1093/logcom/exm064.

Murdoch J. Gabbay and Aad Mathijssen. Nominal universal algebra: equational logic with names and binding. *Journal of Logic and Computation*, 19(6):1455–1508, December 2009. doi: http://dx.doi.org/10.1093/logcom/exp033.

Murdoch J. Gabbay and Aad Mathijssen. A nominal axiomatisation of the lambda-calculus. *Journal of Logic and Computation*, 20(2):501–531, April 2010. doi: http://dx.doi.org/10.1093/logcom/exp049.

[1] A solution to this is *raising*. Raising identifies a name with an explicitly numbered functional abstraction. From that point of view, the advantage of nominal techniques is that its naming is 'lazy' and 'global', whereas the naming in raised higher-order terms is 'eager' and 'local'.

A Modal Tableau Approach for Minimal Model Generation

Fabio Papacchini*
* University of Manchester, School of Computer Science
papacchf@cs.man.ac.uk

Renate A. Schmidt†
† University of Manchester, School of Computer Science
schmidt@cs.man.ac.uk

Abstract

Several studies have been performed for minimal model reasoning for classical logics. In contrast, for non-classical logics like modal logic just few studies have been carried out. We present a modal tableau calculus for minimal model generation which performs on modal clauses. This calculus returns all and only minimal modal Herbrand models, and each model exactly once.

1 Introduction

Previous works (e.g. Bry and Yahya (2000); Niemelä (1996)) have focused on the creation of tableaux calculi for minimal model generation for classical logics. Similar studies have also been carried out for non-classical logics, but most of them work on the first-order translation to which are applied already existent calculi (Georgieva et al., 2001). Our aim is to present a modal tableau calculus generating all and only minimal modal Herbrand models of a set of modal clauses, and each model no more than once. This calculus works directly on modal clauses and. Specifically, it works on disjunctions of labelled formulae and labelled relations which are represented, respectively, by $u : \phi$ and $(u, v) : R$.

A *modal Herbrand universe* ($W_{\mathcal{U}}$) is the set of all ground terms over a signature composed of a constant w and unary functional symbols $f_{\Diamond\phi_i}$ uniquely associated to diamond formulae. $W_{\mathcal{U}}$ represents all possible worlds that are part of any model of the modal input.

A *modal Herbrand interpretation* I is a set composed of labelled formulae and labelled relations. Specifically, labelled formulae have the form $u : p$ where $u \in W_{\mathcal{U}}$ and $p \in \Sigma$, and labelled relations have the form $(u, v) : R$ where $u, v \in W_{\mathcal{U}}$. There exists a close relation between the modal Herbrand interpretation and the standard definition of model for modal formulae.

If a labelled formula $u : \phi$ is true in a modal Herbrand interpretation I, then I is a *modal Herbrand model* H of $u : \phi$. The definition of minimal modal Herbrand model is based on the subset relationship. Let H be a modal Herbrand model of a labeled formula $u : \phi$, then H is a *minimal modal Herbrand model* iff for every other modal Herbrand model H' of $u : \phi$, if $H' \subseteq H$ then $H = H'$.

2 Minimal Modal Model Generation Calculus

As the tableau works on a set of clauses, the modal formula has to be transformed into clausal normal form. We apply all classical steps of this transformation with two exceptions. First, we apply the box miniscoping during the conjunctive normal form transformation. Second, we add the label w during the clause extraction. The *box miniscoping* is the transformation $\Box(\phi_1 \wedge \phi_2) \Rightarrow \Box\phi_1 \wedge \Box\phi_2$. In this way, a conjunction appears only in the scope of a $\Diamond$ operator.

Our tableau calculus is composed of four expansion rules and the model constraint (Table 1).

The $(\Diamond)$ rule is the union of the standard α and $\Diamond$ rule in modal tableau calculi. There is an important difference between our rule and the $\Diamond$ rule, in fact $f_{\Diamond\phi}(u)$ is not *any* created term, but it is uniquely associated to that specific diamond formula and the term attached to the diamond formula.

The $(\vee)_E$ rule is the only rule which does not contribute to the model, and it prepares the input for the other rules. The symbol Φ which appears in this rule represents a generic disjunction of tableau literals.

The (CS) rule takes as input a disjunction of positive tableau literals. A tableau literal is either a positive labelled literal of the form $u : p$ or $u : \Diamond\phi$ (in Table 1 this kind of modal formulae is represented by the symbol $\mathcal{P}$), or a positive labelled relation $(u, v) : R$. The negation of positive labelled literals is obtained by applying the function neg. neg is defined as follows:

$$neg(u, \mathcal{P}) = \begin{cases} u : \neg p & \text{if } \mathcal{P} = p \\ (u, f_{\Diamond\phi}(u)) : \neg\mathcal{R} & \text{if } \mathcal{P} = \Diamond\phi \end{cases}.$$

The (CS) rule selects one of the positive tableau literals and the negation of all the others that appear on the right of the selected one. This rule is the only branching rule of the calculus. It avoids the creation of a model more than once, and the model associated to the left-most open, fully expanded and finite branch is minimal.

Table 1: Expansion rules and model contraint for the minimal modal model generation calculus

Expansion Rules

$$(\Diamond)\frac{u : \Diamond(\phi_1 \wedge \ldots \wedge \phi_n)}{\begin{array}{c}(u, f_{\Diamond\phi}(u)) : R \\ f_{\Diamond\phi}(u) : \phi_1 \\ \vdots \\ f_{\Diamond\phi}(u) : \phi_n\end{array}}$$

Where $\phi = \phi_1 \wedge \ldots \wedge \phi_n$ and $f_{\Diamond\phi}$ is the uniquely associated functional symbol

$$(CS)\frac{u_1 : \mathcal{P}_1 \vee \ldots \vee u_n : \mathcal{P}_n \vee (v_1, w_1) : R \vee \ldots \vee (v_m, w_m) : R}{\begin{array}{c|c|c|c} (u_1 : \mathcal{P}_1) & (u_2 : \mathcal{P}_2) & & (v_m, w_m) : R \\ neg(u_2, \mathcal{P}_2) & \vdots & \therefore & \\ \vdots & (v_m, w_m) : \neg R & & \\ (v_m, w_m) : \neg R & & & \end{array}}$$

$$(\vee)_E\frac{u : \phi_1 \vee \ldots \vee \phi_n \vee \Phi}{u : \phi_1 \vee \ldots \vee u : \phi_n \vee \Phi}$$

$$(PUHR)\frac{\begin{array}{c}u_1 : p_1, \ldots, u_n : p_n \\ (v_1, w_1) : R, \ldots, (v_m, w_m) : R \\ (s_1, t_1) : R, \ldots, (s_j, t_j) : R \\ u_1 : \neg p_1 \vee \ldots \vee u_n : \neg p_n \vee v_1 : \Box\phi_1 \vee \ldots \\ \vee v_m : \Box\phi_m \vee (s_1, t_1) : \neg R \vee \ldots \vee (s_j, t_j) : \neg R \vee \Psi\end{array}}{(w_1 : \phi_1) \vee \ldots \vee (w_m : \phi_m) \vee \Psi}$$

Model Constraint

$$(u_1 : \neg p_1) \vee \ldots \vee (u_n : \neg p_n) \vee (v_1, w_1) : \neg R \vee \ldots \vee (v_m, w_m) : \neg R$$

This clause is generated from a finite, open and fully expanded tableau branch $\mathcal{B}$, and it is propagated to all the other branches in the tableau derivation on the right of $\mathcal{B}$.

The $(PUHR)$ rule is the most complex rule and the only one that can close a branch. The idea behind this rule is that the elements of a model are only positive, and a clause with negative tableau literals is expanded only if it is necessary. A negative tableau literal has one of three forms: $u : \neg p$, $u : \Box\phi$ or $(u, v) : \neg R$. The symbol Ψ which appears in the main premise represents a disjunction of positive tableau literals.

In addition to the expansion rules we use minimal model constraints during backtracking to avoid the generation of non-minimal models. Given a minimal modal Herbrand model H, the *model constraint* extracted from H is defined as follows:

$$u_1 : \neg p_1 \vee \ldots \vee u_n : \neg p_n \vee (v_1, w_1) : \neg R \vee \ldots \vee (v_m, w_m) : \neg R$$

where $H = \{u_1 : p_1, \ldots, u_n : p_n, (v_1, w_1) : R, \ldots, (v_m, w_m) : R\}$.

Once a branch is fully expanded, open and finite the model associated to that branch is extracted, and its model constraint is propagated to all the other branches on the right of the one from which the model has been extracted.

This calculus terminates, generates all and only minimal models (i.e. it is minimal model complete and sound) and does not create any model more than once. These properties can be proved showing that there exists a bijection between this calculus and the depth-first minimal model generation procedure presented in Bry and Yahya (2000) applied to the semantic translation of the modal clauses.

References

François Bry and Adnan Yahya. Positive unit hyperresolution tableaux and their application to minimal model generation. *J. Autom. Reason.*, 25(1):35–82, 2000.

L. Georgieva, U. Hustadt, and R. A. Schmidt. Computational space efficiency and minimal model generation for guarded formulae. In *Proceedings of LPAR'01*, Lecture Notes in Artificial Intelligence. Springer, 2001.

Ilkka Niemelä. A tableau calculus for minimal model reasoning. In *Proceedings of the Fifth Workshop on Theorem Proving with Analytic Tableaux and Related Methods*, pages 278–294. Springer-Verlag, 1996.

Productive use of failure in top-down formal methods

Alan Bundy
[⋆]School of Informatics
University of Edinburgh
bundy@staffmail.ed.ac.uk

Gudmund Grov
[†]School of Informatics
University of Edinburgh
ggrov@staffmail.ed.ac.uk

Yuhui Lin
[‡]School of Informatics
University of Edinburgh
Y.H.Lin-2@sms.ed.ac.uk

Abstract

Proof obligations (POs) arise when applying formal methods (FMs). The undischarged POs can become a bottleneck in the use of FMs in industry. The work we proposed here, as a part of AI4FM project, aims at increasing the proportion of discharged POs by analysing failed proofs and related patches to classify POs as families and construct proof strategies, which can be used to guide proof search for the POs in the same families.

1 Introduction

Commercial use of formal methods (FM) has had great success in many safety-critical domains such as railway and aviation. There is also increased use in other sectors, for example Microsoft has used FMs to verify device drivers. A recent paper by (Woodcock et al., 2009) provides an up-to-date analysis of a significant number of industrial applications of FMs.

In a top-down approach, FMs are applied early in the development to capture the requirements. The specification is stepwise refined into the final product, typically a piece of software. Many such methods are posit and prove, where a designer posits a step of development and then seeks to justify it. This justification is by proof of generated proof obligations. Examples of such top-down FMs are VDM, B, Event-B and some use of Z.

2 Our approach

As the use of FMs has spread beyond small groups of experts out to far larger groups of industrial engineers, proof automation has become a bottleneck. Although a large proportion of the POs can be discharged by automatic theorem provers, $5 - 10\%$ still requires user interaction, which often requires (expensive) theorem proving experts. For commercial application this can be thousands of proof obligations (POs), requiring many man months or years of work.

We observe that the POs from FMs tend to exhibit a "similarity" in the sense that they can be grouped into "families" and the same (high-level) proof approach can be successfully applied to all members of the family.To illustrate, an industrial case-study using Event-B generated 300 POs, 100 of these required user-interaction, however they could be handled by 5 strategies.

Therefore, we hope to explore this notion of proof families. In particular, we believe that the notion of failures by automatic theorem provers and heuristics can be explored to guide the proof further. Our hypothesis is:

> *we believe that it is possible to classify families of failed proofs within FMs by analysing the context of failures. In each family there exist patterns of proof strategies which can be used to guide the proof search of the blocked proofs.*

Our work is inspired by previous work in *rippling* (Bundy et al., 2005), which is a rewriting technique which works when the goal is embedded in one of the given. Rippling then forces rewriting towards the given, thus guaranteeing termination. It was originally developed for inductive proofs, but has been successfully applied to many other domains. *Proof critics* have been developed for rippling, where the nature of the failure is explored to propose a patch. For example, certain failures suggest a generalisation, others suggests a missing lemma or a different induction rule. Rippling is applicable to certain types of proof obligations, those where a system invariant has to be preserved over the operations. In Grov et al. (2010) we report on a small experiment where use rippling for such POs arising from an Event-B model. A few observations can be made there to illustrate our hypothesis:

- each rippling step has to reduce some measure (to ensure termination) and to preserve what is known as the skeleton, which is the given we are rippling towards. Some steps were not measure reducing while preserving the skeleton. One patch which would have solved this was to introduce a secondary measure on the symbols. In this case the function updates operator was rewritten to set union.
- most rewrite rules were conditional, hence when rippling terminated; the conditions (which were not rippling goals) had to be discharged. Separate patches had to be developed for these.

Our approach will be empirical and iterative, and will include analysis of proof failures and related patches to them. Ideally we would like to use existing (preferable industrial) examples. However, these are the finished articles, which makes it hard to see the productive use of failure that the expert user (implicitly) used to discharge it. Thus we need to see the whole search space including places were failed proof attempts were patched, so to explore these search spaces ourselves.

Through these proofs, we are able to classify POs as families, and extract proof strategies from successful proof attempts. These proof strategies can be used to guide proof search for POs, which are not discharged, in the same families. We will most likely implement and evaluate our patches in Isaplanner (Dixon and Fleuriot, 2003), which is an Isabelle based proof planner, implementing rippling among other methods.

3 Conclusion

We have outlined an approach where we plan to explore the type failures of proofs in top-down formal methods to discover new patches which can be applied to family of proofs. This work is part to the AI4FM project, where the overall goal is to learn high-level proof strategies (and patches from failures) from exemplar proofs provided by expert users. This work is a first step of this project, since we need to understand the typical strategies and patches before we can use machine learning techniques to automatically derive them.

Acknowledgements

This work is supported by EPSRC grant EP/H024204/1: *AI4FM: using AI to aid automation of proof search in Formal Methods*. For more details see `http://www.ai4fm.org`.

References

A. Bundy, D. Basin, D. Hutter, and A. Ireland. *Rippling: Meta-level Guidance for Mathematical Reasoning*, volume 56 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 2005.

Lucas Dixon and Jaques Fleuriot. IsaPlanner: A Prototype Proof Planner in Isabelle. In *Proceedings of CADE'03*, volume 2741 of *LNCS*, pages 279–283, 2003.

Gudmund Grov, Alan Bundy, and Lucas Dixon. A small experiement in event-b rippling. In proceedings of AVoCS 2010 and Rodin User and Developer Workshop 2010, April 2010.

J. Woodcock, P.G. Larsen, J. Bicarregui, and J. Fitzgerald. Formal methods: Practice and experience. *ACM Computing Surveys (CSUR)*, 41(4):1–36, 2009.

Coalgebraic Derivations in Logic Programming

Ekaterina Komendantskaya*
*School of Computing
University of Dundee
katya@computing.dundee.ac.uk

John Power†
†Department of Computer Science
University of Bath
A.J.Power@bath.ac.uk

Abstract

Coalgebra may be used to provide semantics for SLD-derivations, both finite and infinite. We first give such semantics to classical SLD-derivations, proving results such as adequacy, soundness and completeness. Then, based upon coalgebraic semantics, we propose a new sound and complete algorithm for parallel derivations. We analyse this new algorithm in terms of the Theory of Observables, and we prove soundness, completeness, correctness and full abstraction results.

1 Motivation

There are two trends in logic programming that are both desirable and problematic: coinductive definitions and concurrent computations. We illustrate both using the following example.

Example 1 *The program* `Stream` *defines the infinite stream of binary bits:*

$$\begin{aligned} \texttt{bit}(0) &\leftarrow \\ \texttt{bit}(1) &\leftarrow \\ \texttt{stream(scons (x,y))} &\leftarrow \texttt{bit(x)}, \texttt{stream(y)} \end{aligned}$$

`Stream` is a coinductive definition, with proof search for the goal `stream(x)` resulting in an infinite derivation. Programs like `Stream` can be given declarative semantics via the *greatest* fixed point of the semantic operator T_P. But fixed point semantics is incomplete (Lloyd (1987)) as it fails for some infinite derivations.

Example 2 *The program $R(x) \leftarrow R(f(x))$ is characterised by the greatest fixed point of the T_P operator, which contains $R(f^\omega(a))$, although no infinite term is computed by SLD-resolution.*

There have been numerous attempts to resolve the mismatch between infinite derivations and greatest fixed point semantics, e.g., Gupta et al. (2007); Jaume (2002); Lloyd (1987); Paulson and Smith (1989); Simon et al. (2007). But infinite SLD derivations of both finite and infinite objects have not yet received a uniform semantics.

Another distinguishing feature of logic programming languages is that they allow implicit parallel execution of programs. The three main types of parallelism used in implementations are *and-parallelism*, *or-parallelism*, and their combination: see Gupta and Costa (1994); Pontelli and Gupta (1995).

Or-parallelism arises when more than one clause unifies with the goal: the corresponding bodies can be executed in or-parallel fashion. *And-parallelism* arises when more than one atom is present in the goal. That is, given a goal $G = \leftarrow B_1, \ldots B_n$, an *and-parallel algorithm* of resolution looks for derivations for each B_i simultaneously. *And-or parallelism* features both kinds of parallelism. However, many first-order algorithms are P-complete and hence inherently sequential (Dwork et al. (1984); Kanellakis (1988)). This especially concerns first-order unification and variable substitution in the presence of variable dependencies.

Example 3 *The goal* `stream(cons(x, cons(y,x)))`*, if processed sequentially, will lead to a failed derivation in three derivation steps. But, if the goal is processed in and-or parallel fashion, the derivation algorithm will not be able to determine inconsistency between substitutions for x in parallel branches of the derivation tree.*

2 Summary of results

We apply the coalgebraic semantics proposed in Komendantskaya et al. (2010); Komendantskaya and Power (2011) to address the above two problems. Coalgebraic semantics can be used instead of greatest fixed point semantics. Unlike

greatest fixed point semantics, coalgebraic semantics yields soundness and completeness results for both finite and infinite SLD-derivations.

We test our coalgebraic semantics using the *theory of observables*, Comini et al. (2001). According to this theory, the traditional characterisation of logic programs in terms of their input/output behavior, successful derivations and their corresponding fixed point semantics, is not sufficient for program analysis and optimisation. One requires more complete information about SLD-derivations, such as sequences of goals, most general unifiers and variants of clauses.

The idea of observational semantics for logic programs is to observe equal behaviour of logic programs and to distinguish logic programs with different computational behaviour. So the choice of observables and semantic models is closely related to the choice of equivalence relation defined on logic programs. In the theory of observables, programs P_1 and P_2 are said to be observationally equivalent if, for any goal G, they yield the same call patterns in the SLD derivations. Given a suitable semantics, one can prove that if P_1 and P_2 have equal semantic models, then P_1 and P_2 are observationally equivalent (*correctness result*). The converse is a *full abstraction* result.

Our coalgebraic semantics yields the correctness result relative to the sequential algorithm of SLD resolution but not the full abstraction result. The failure of the latter does not mean that coalgebraic semantics is not suitable for characterizing observational behavior of logic programs. Rather, it indicates the mismatch between the sequential algorithm of SLD resolution and the concurrent nature of coalgebraic semantics.

We therefore propose a new *coinductive derivation algorithm* inspired by coalgebraic semantics. The algorithm provides an elegant solution to the problem of implementing both corecursion and concurrency in logic programming. We prove soundness, completeness, correctness and full abstraction results for the new coinductive derivations relative to the coalgebraic semantics of Komendantskaya and Power (2011).

References

M. Comini, G. Levi, and M. C. Meo. A theory of observables for logic programs. *Inf. Comput.*, 169(1):23–80, 2001.

C. Dwork, P.C. Kanellakis, and J.C. Mitchell. On the sequential nature of unification. *Journal of Logic Programming*, 1: 35–50, 1984.

G. Gupta and V.S. Costa. Optimal implementation of and-or parallel prolog. In *Conference proceedings on PARLE'92*, pages 71–92, New York, NY, USA, 1994. Elsevier North-Holland, Inc.

G. Gupta, A. Bansal, R. Min, L. Simon, and A. Mallya. Coinductive logic programming and its applications. In *ICLP 2007*, volume 4670 of *LNCS*, pages 27–44. Springer, 2007.

M. Jaume. On greatest fixpoint semantics of logic programming. *J. Log. Comput.*, 12(2):321–342, 2002.

P. C. Kanellakis. Logic programming and parallel complexity. In *Foundations of Deductive Databases and Logic Programming.*, pages 547–585. M. Kaufmann, 1988. ISBN 0-934613-40-0.

E. Komendantskaya and J. Power. Coalgebraic semantics for derivations in logic programming. In *Submitted to CALCO'11*, 2011.

E. Komendantskaya, G. McCusker, and J. Power. Coalgebraic semantics for parallel derivation strategies in logic programming. In *Proc. of AMAST'2010 - 13th Int. Conf. on Algebraic Methodology and Software Technology*, volume 6486 of *LNCS*, 2010.

J.W. Lloyd. *Foundations of Logic Programming*. Springer-Verlag, 2nd edition, 1987.

L.C. Paulson and A.W. Smith. Logic programming, functional programming, and inductive definitions. In *ELP*, pages 283–309, 1989.

E. Pontelli and G. Gupta. On the duality between or-parallelism and and-parallelism in logic programming. In *Euro-Par*, pages 43–54, 1995.

L. Simon, A. Bansal, A. Mallya, and G. Gupta. Co-logic programming: Extending logic programming with coinduction. In *ICALP*, volume 4596 of *LNCS*, pages 472–483. Springer, 2007.

The Application of AI Techniques to Deformation in Metal Manufacturing

Clare Dixon, Frans Coenen and Muhammad Khan,
* Department of Computer Science,
University of Liverpool,
Liverpool, L69 3BX, UK
{cldixon,coenen mskhan}@liverpool.ac.uk

Abstract

We describe the INMA project, part of which aims to use AI techniques to predict and correct shape deformation in the manufacture of titanium parts using in the aircraft industry.

1 Introduction

The INMA project[1] is a multi-partner project funded by the EU[2] which brings together engineers, materials scientists, computer scientists and industrialists and aims to develop and improve manufacturing techniques for the production of titanium parts to be used in the aircraft industry. Currently the aircraft industry uses complicated and expensive technologies to shape titanium sheet components. The project proposes technology based on the relatively new technique of Asymmetric Incremental Sheet Forming (AISF) aiming to obtain increased flexibility, cost reduction, minimised energy consumption and a speed up in the industrialisation phase.

2 Background

When manufacturing parts using AISF a metal sheet is clamped into a holder and the desired shaped is produced using the continuous movement of a simple round-headed forming tool. The forming tool is provided with a tool path generated by a CAD model and the part is pressed out according to the co-ordinates of the tool path. However, due to the nature of the metal used and the manufacturing process, *spring back* occurs which means that the geometry of the shaped part is different from the geometry of the desired part, i.e. some deformation is introduced. Work package 4 of the INMA project (Liverpool's part) involves analysing the AISF process parameters, the required shape and the formed shape to predict and correct the deformation.

Currently we are looking at the background to this problem and solutions adopted elsewhere. For example Dearden et al. (2006) present an iterative laser forming solution to the problem of correcting such distortion. In Allwood et al. (2005) the authors consider a number of products that could potentially be formed using AISF and show that the accuracy of the formed part needs to be improved before this process could be used in a large number of these products. In Hirt et al. (2004) the authors consider two drawbacks to AISF, namely that of the metal thickness of the resulting shape, and the geometric accuracy, the latter relating to the problem we study. Regarding the geometric accuracy, a part is produced and measured. Then the deformation is calculated and the tool path corrected using the inverse of the deformation at each point. This process is applied iteratively until the deformation is small enough. In Bambach et al. (2009) the authors propose a multi-stage forming technique, i.e. rather than a single pass by the machine tool several are made. Further, they consider the effect of different initial geometries with respect to the deformation produced on the desired shape. As a case study they consider a flat topped pyramid shape. The initial geometry with a corner radius larger than the desired shape and a number of forming stages produced the least deformation.

3 Proposed Solutions

The main tasks are as follows:

- investigating or developing good representations for the data;

[1] www.inmaproject.eu
[2] http://cordis.europa.eu/fetch?CALLER=FP7_PROJ_EN&ACTION=D&DOC=1&CAT=PROJ&QUERY=012eb403bac5:d891:5cf7dbdc&RCN=96396

- the application and development of AI techniques to the data to propose tool paths that account for the effects of spring back to minimise deformation;
- experimentation with these techniques and analysis of the best or combination of techniques.

3.1 Representation

Co-ordinate Points Currently the data (both the desired shape and the actual shape) is represented as sets of 3-D (x, y, z) co-ordinates. Input to the machine tools is a standard CAD/CAM file in APT (Automatically programmed Tool) code. Due to the volume of data involved we may have to reduce the number of co-ordinate points by considering points at particular co-ordinates in a mesh rather than the full co-ordinate set.

Registration We need to ensure that the 3-D co-ordinates are relative to the same origin otherwise we won't be comparing related parts of the geometry and the deformation may appear larger than expected.

Deformation The deformation of the resultant part with respect to the required part can be calculated a number of ways. One example is to calculate the difference in z co-ordinates for given x, y co-ordinates. We will need to be able to provide a measure of overall deformation, for example, the average per point for a grid like-representation to allow analysis of our results.

Parameters It is likely that a number of parameters will affect the deformation, for example, the type of metal, the composition of the metal, the processing of the sheets prior to part forming etc. We also need to represent data about these parameters to investigate whether they do have some affect on the deformation. Additionally we anticipate having to utilise derived data, for example, to differentiate parts of the shape representing peaks, troughs, ridges, flat portions etc as we expect the deformation will be affected by such factors.

3.2 Predicting Deformation

Given the data and a representation of this for a specific shape we need a way of predicting the deformation to generate a new tool path such that the resultant shape, once manufactured, is close enough to the required shape. An obvious first attempt would be to invert the deformation at each point. That is, if for some x, y point the z co-ordinate for the obtained shape is h units greater than expected, then to correct this in the new tool path we would subtract h from the z value of the desired shape. This is the approach taken in Hirt et al. (2004) on which we hope to improve by applying AI techniques. We will consider a number of techniques but are considering data mining, neural networks, statistical methods. Obviously the collection of a large amount of data on which to develop and test our ideas is expensive when dealing with titanium. We hope to be able to initially use data from parts formed in aluminum before moving on to testing with titanium.

3.3 Experimentation and Analysis

Here we would like to compare a number of techniques developed above at predicting and correcting the tool path for a particular shape and, if possible, combine the best of these.

Acknowledgements This work was supported by the EU project "Innovative Manufacturing of complex Ti Sheet Components" (INMA), grant agreement number 266208.

References

J.M. Allwood, G.P.F. King, and J. Duflou. A structured search for applications of the incremental sheet-forming process by product segmentation. *Proceedings of the Institution of Mechanical Engineers, Part B: Journal of Engineering Manufacture*, 219(2):239–244, 2005.

M. Bambach, B. Taleb Araghi, and G. Hirt. Strategies to improve the geometric accuracy in asymmetric single point incremental forming. *Production Engineering Research and Development*, 3(2):145–156, 2009.

G. Dearden, S.P. Edwardson, E. Abed, K. Bartkowiak, and K.G. Watkins. Correction of distortion and design shape in aluminium structures using laser forming. In *25th International Congress on Applications of Lasers and Electro Optics(ICALEO 2006)*, pages 813–817, 2006.

G. Hirt, J. Ames, M. Bambach, R. Kopp, and R. Kopp. Forming strategies and Process Modelling for CNC Incremental Sheet Forming. *CIRP Annals - Manufacturing Technology*, 53(1):203–206, 2004.

Heterogeneous Proofs: Spider Diagrams meet Higher-Order Provers

Matej Urbas*
*Computer Laboratory
University of Cambridge
Matej.Urbas@cl.cam.ac.uk

Mateja Jamnik†
†Computer Laboratory
University of Cambridge
Mateja.Jamnik@cl.cam.ac.uk

Abstract

We present an interactive heterogeneous theorem proving framework, which performs formal reasoning by arbitrarily mixing diagrammatic and sentential proof steps.

We use Isabelle to enable formal reasoning with either traditional sentences or spider diagrams. We provide a mechanisation of the theory of abstract spider diagrams and establish a formal link between diagrammatic concepts and the existing theories in Isabelle.

1 Introduction

Diagrams are often employed as illustrations in "pen and paper" reasoning. In the past, they frequently formed essential parts of proofs. Eventually, with advent of proof theory, their role became almost exclusively that of a visual help. Still, the intuitive nature of diagrams motivated the design of formal diagrammatic reasoning systems – for example, spider diagrams (Howse et al., 2001) and constraint diagrams (Gil et al., 2001). Consequently, some purely diagrammatic theorem provers have been developed, DIAMOND (Jamnik et al., 1999), Edith (Stapleton et al., 2007) and Dr.Doodle (Winterstein et al., 2004) are some examples.

Heterogeneous reasoning was the next step in the development of diagrammatic reasoning systems. It merged the diagrammatic and sentential modes of reasoning and allowed proof steps to be applied to either diagrammatic, sentential or mixed formulae. In the paper *Reasoning with Sentences and Diagrams* (Hammer, 1994), Hammer laid the formal foundations for such heterogeneous reasoning systems.

Later, Barwise and Etchemendy (1998), Barker-Plummer et al. (2008) and Shin (2004) investigated heterogeneous reasoning software. The result was a framework called Openproof (Barker-Plummer et al., 2008), which uses diagrammatic representation as an input method. The diagrammatic part of the framework is not formalised within the logic of the sentential reasoner. Therefore, the diagrammatic and sentential components remain logically separated.

Our goal is to enable formal interactive heterogeneous reasoning in a general purpose theorem prover. We investigate three aspects of interactive heterogeneous reasoning: *a)* the direction of proofs (e.g., from a diagrammatic assumption to a sentential conclusion and vice versa), *b)* expression of statements that contain mixed sentential and diagrammatic terms, and *c)* mixed application of diagrammatic, sentential, and heterogeneous inference steps.

Our key motivation is to provide different representations of formulae and to enable reasoning about diagrams sententially or vice versa. We believe this can improve intuitiveness and readability of formulae and proofs in specific domains of verification.

In contrast to the approach of Openproof our aim is not to keep the two reasoning systems separated, but to integrate them using heterogeneous representation and reasoning. In addition, we want to formalise diagrams and some of their inference rules in the logic of an LCF-style (Gordon et al., 1979) higher-order theorem prover. With this we hope to enable certified proof reconstruction of heterogeneous proofs.

In order to build a heterogeneous reasoning framework, we first chose an existing diagrammatic reasoning language called *spider diagrams*. The second part is the sentential reasoner, for which we chose Isabelle (Wenzel et al., 2008).

We formalised the theory of spider diagrams in Isabelle. This enabled sentential reasoning about diagrams and also simplified translation from spider diagrams to sentences. Translation from sentences to diagrams, proof automation, and proof reconstruction, however, is still work in progress. Diagrammatic reasoning on spider diagrams will be performed by our external diagrammatic reasoner Speedith (currently also in development).

2 Heterogeneous Integration

Ultimately, we want to enable formal graphical reasoning as is depicted in Figure 1. Sentential expressions are drawn as diagrams if the translation is feasible. More importantly, the external diagrammatic reasoner can be invoked in an interactive mode within the proof body like any other tactic in Isabelle.

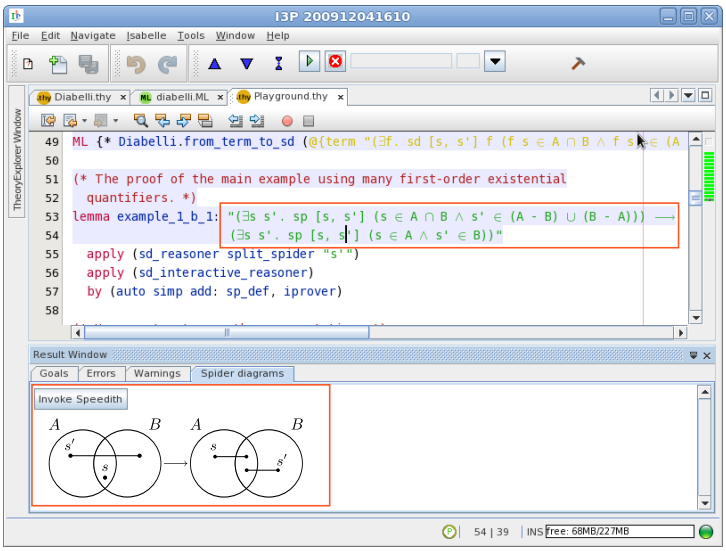

Figure 1: A mockup of the resulting heterogeneous framework.

To achieve this we will integrate the diagrammatic reasoner on both the level of reasoning tactics as well as within the graphical user interface of Isabelle. For this purpose we aim to use Speedith.

The user will be able to invoke the diagrammatic reasoner at any time during the proof, with an option to do so in an interactive or fully automated mode. Additionally, the currently active statement (lemma or proof obligation) will be automatically visualised as a diagram in an embedded window of the GUI.

Discussion. We outline a work-in-progress of an integration of a diagrammatic language with diagrammatic inference rules into a sentential theorem prover. This enables formal heterogeneous reasoning with mixed diagrams and sentences.

In summary, we believe that heterogeneous reasoning can improve the ease of working with specific domains of verification in general purpose theorem provers.

References

Dave Barker-Plummer, John Etchemendy, Albert Liu, Michael Murray, and Nik Swoboda. Openproof - A Flexible Framework for Heterogeneous Reasoning. *LNCS*, 5223:347, 2008. doi: {10.1007/978-3-540-87730-1_32}.

Jon Barwise and John Etchemendy. A Computational Architecture for Heterogeneous Reasoning. In *TARK*, pages 1–11. Morgan Kaufmann, 1998. doi: {10.1.1.24.5942}.

Joseph Gil, John Howse, and Stuart Kent. Towards a Formalization of Constraint Diagrams. In *IEEE Symposia on Human-Centric Computing Languages and Environments*, page 72, 2001. doi: {10.1109/HCC.2001.995227}.

Michael J. Gordon, Arthur J. Milner, and Christopher P. Wadsworth. *Edinburgh LCF: A Mechanised Logic of Computation*, volume 78 of *LNCS*. Springer Berlin / Heidelberg, 1979. doi: {10.1007/3-540-09724-4}.

Eric Hammer. Reasoning with Sentences and Diagrams. *NDJFL*, 35(1):73–87, 1994.

John Howse, Fernando Molina, John Taylor, Stuart Kent, and Joseph Gil. Spider Diagrams: A Diagrammatic Reasoning System. *JVLC*, 12(3):299–324, 2001. doi: {10.1006/jvlc.2000.0210}.

Mateja Jamnik, Alan Bundy, and Ian Green. On Automating Diagrammatic Proofs of Arithmetic Arguments. *JOLLI*, 8 (3):297–321, 1999.

Sun-Joo Shin. Heterogeneous Reasoning and its Logic. *BSL*, 10(1):86–106, 2004.

Gem Stapleton, Judith Masthoff, Jean Flower, Andrew Fish, and Jane Southern. Automated Theorem Proving in Euler Diagram Systems. *JAR*, 39(4):431–470, 2007. doi: {10.1007/s10817-007-9069-y}.

Makarius Wenzel, Lawrence C. Paulson, and Tobias Nipkow. The Isabelle Framework. In *TPHOLs*, pages 33–38, 2008.

Daniel Winterstein, Alan Bundy, and Corin A. Gurr. Dr.Doodle: A Diagrammatic Theorem Prover. In *IJCAR*, pages 331–335, 2004.

Towards the Formal Verification of Human-Agent Teamwork *

Richard Stocker, Louise Dennis, Clare Dixon and Michael Fisher
*Department of Computer Science, University of Liverpool, UK

Abstract

The formal analysis of computational processes is by now a well-established field. Yet, the problem of dealing with the interactions with humans still remains. In this work we are concerned with addressing this problem. Our overall goal is to provide formal verification techniques for human-agent teamwork.

1 Introduction

Computational devices naturally interact with humans. These devices can range from mobile phones or domestic appliances, all the way to fully autonomous robots. In many cases all that the users care about is that the device works well most of the time. However, in mission critical scenarios we clearly require a more formal, and consequently much deeper analysis. Specifically, as various space agencies plan missions to the Moon and Mars which involve robots and astronauts collaborating, then we surely need some form of *formal verification* for astronaut-robot teamwork. This is needed at least for astronaut safety (e.g. "the astronaut will never be out of contact with the base") but also for mission targets (e.g. "three robots and two astronauts can together build the shelter within one hour"). However, this presents three questions: (1) How are we to go about this; (2) How can we possibly verify human behaviour; and (3) How can we analyse teamwork?

2 Background

An agent (14) essentially captures the idea of an autonomous entity, being able to make its own choices and carry out its own actions. Beyond simple autonomy, *rational agents* are increasingly used as a high-level abstraction/metaphor for building complex/autonomous space systems (6). Rational agents can be seen as agents that make their decisions in a *rational* and *explainable* way (rather than, for example, purely randomly). The central aspects of the rational agent metaphor is that such agents are autonomous, but can react to changes in circumstance and can choose what to do based on their own agenda.

By formal verification we mean analyzing the correctness of a system using mathematical proofs. One technique for doing so is model-checking (1). Model-checking is the process of checking a specification of the system against a model representing that system. This model takes the form of a finite state machine i.e. a directed graph consisting of vertices and edges. The specification is checked on every path within that model. If the specification fails on any path (representing a potential execution), then this is identified to the designer.

In (3) a formal approach to the problem of human-agent (and so astronaut-robot) analysis was proposed and suggested the model-checking of *Brahms* models (8; 13; 11) but provided no details. *Brahms* is a simulation/modelling language in which complex human-agent work patterns can be described. Importantly for our purposes, *Brahms* is based on the concept of *rational agents* and the system continues to be successfully used within NASA for the sophisticated modelling of astronaut-robot planetary exploration teams (5; 10; 9). Thus, it seems natural to want to formally verify the *Brahms* language designed to model human activity using *rational agents*.

3 Proposed Solution

The main tasks are as follows:

- produce a formal semantics of the *Brahms* language;
- create a tool capable of performing formal verification on *Brahms* code based on our formal semantics;
- investigating or developing adequately sized scenarios in the *Brahms* language on which to perform formal verification.

*Work partially funded in the UK through EPSRC grants EP/F033567 and EP/F037201.

3.1 Semantics

Until now, the *Brahms* language had no *formal* semantics (unless you count the implementation code as this). So, in (12) we describe the first formal semantics for the *Brahms* language. These formal semantics provide a route towards formal verification of *Brahms*. In particular, they open up the opportunity of using agent verificaiton tools such as AJPF (2). They also provide a means to aid the creation of a translation tools, so that *Brahms* may be translated into the input languages of model-checkers such as Spin (7) and NuSMV (4). Any translation created could be checked against these semantics to prove its correctness.

3.2 Verification Tool

The ultimate goal of this project is to create a tool which will formally verify human-agent teamwork. By creating a tool to formally verify *Brahms* we hope to make a step towards achieving this goal. Currently we are looking at creating a *Brahms* parser which will parse *Brahms* code into Java data structures that reflect the structure of *Brahms* in our semantics. Using the transition rules in our semantics we will then be able to create tools which will generate code in the form of input languages for model-checkers such as Spin (7). Using our semantics in this way will enable us to prove the code our tool creates is an accurate representation of the *Brahms* code it was generated from.

3.3 Scenarios

The scenarios for testing and demonstrating our tool need to be chosen very carefully. They need to be big enough to have a critical nature and show further possible applications, yet small enough so the accuracy of the verification is easily visible in the results and during demonstrations.

References

[1] C. Baier, and J. P Katoen. Principles of Model Checking. The MIT Press, May 2008.

[2] R. H. Bordini, L. A. Dennis, B. Farwer, and M. Fisher. Automated Verification of Multi-Agent Programs. In *Proc. 23rd IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 69–78, 2008.

[3] R. H. Bordini, M. Fisher, and M. Sierhuis. Formal Verification of Human-Robot Teamwork. In *Proc. 4th ACM/IEEE International Conference on Human Robot Interaction (HRI)*, pages 267–268. ACM Press, 2009.

[4] A. Cimatti, E. M. Clarke, F. Giunchiglia, M. Roveri. *NUSMV: A New Symbolic Model Verifier*, 1999 Computer Aided Verification, 11th International Conference, CAV '99, Trento, Italy, July 6-10, Springer, 1999.

[5] W. Clancey, M. Sierhuis, C. Kaskiris, and R. van Hoof. Advantages of Brahms for Specifying and Implementing a Multiagent Human-Robotic Exploration System. In *Proc. 16th Florida Artificial Intelligence Research Society (FLAIRS)*, pages 7–11. AAAI Press, 2003.

[6] L. A. Dennis, M. Fisher, A. Lisitsa, N. Lincoln, and S. M. Veres. Satellite Control Using Rational Agent Programming. *IEEE Intelligent Systems*, 25(3):92–97, 2010.

[7] G.J. Holzmann *Software model checking with SPIN*. Advances in Computers, 2005.

[8] M. Sierhuis. *Modeling and Simulating Work Practice. BRAHMS: a multiagent modeling and simulation language for work system analysis and design*. PhD thesis, Social Science and Informatics (SWI), University of Amsterdam, The Netherlands, 2001.

[9] M. Sierhuis. Multiagent Modeling and Simulation in Human-Robot Mission Operations. (See `http://ic.arc.nasa.gov/ic/publications`), 2006.

[10] M. Sierhuis, J. M. Bradshaw, A. Acquisti, R. V. Hoof, R. Jeffers, and A. Uszok. Human-Agent Teamwork and Adjustable Autonomy in Practice. In *Proc. 7th International Symposium on Artificial Intelligence, Robotics and Automation in Space (i-SAIRAS)*, 2003.

[11] M. Sierhuis and W. J. Clancey. Modeling and Simulating Work Practice: A Human-Centered Method for Work Systems Design. *IEEE Intelligent Systems*, 17(5), 2002.

[12] R. Stocker, M. Fisher, C. Dixon and L. Dennis. A Formal Semantics for Brahms. (See http://www.csc.liv.ac.uk/∼rss/publications), 2011.

[13] R. van Hoof. Brahms website: http://www.agentisolutions.com, 2000.

[14] M. Wooldridge. *An Introduction to Multiagent Systems*. John Wiley & Sons, 2002.

Modelling of Wireless Local Area Network Security Attacks in Alloy

Aliaa Alabdali, Lilia Georgieva*, Greg Michaelson
*MACS, Heriot-Watt University, Edinburgh, EH14 4AS
aa989@macs.hw.ac.uk, lilia@macs.hw.ac.uk, greg@macs.hw.ac.uk

Abstract

We model communication in a wireless local area network (WLAN) in first-order logic. We analyse the model for security vulnerabilities using the model analyser Alloy. We develop a methodology for representing secure message exchange on a WLAN and detecting passive and active *man in the middle attacks*.

1 Introduction

Recently wireless networks have become pervasive. They have millions of users and enable convenient and flexible information exchange among devices such as notebook computers, PDAs and Tablet PCs, but also among mobile phones, digital cameras, and hand-held games.

Growing number of users and intensive use revealed vulnerabilities of wireless networks that can be exploited. Security of wireless networks can be compromised in a number of ways: the wireless channel can be eavesdropped; the data transmitted over the network can be altered, identities of the communicated parties can be impersonated, the communication channel can be overused or jammed (Butty and Hubaux, 2007). Ensuring information security on a wireless network has been identified as paramount (Pathan et al., 2006; Yang et al., 2005).

Security solutions devised for wired networks cannot be used to protect the wireless ones (Arbaugh, 2004). The rapid commercialization of new products and services which use wireless networks does not allow enough time for design of a robust security architecture (Latré et al., 2011). Consequently, the risk of losing and misusing data transmitted over a wireless network is high (Gritzalis and Lopez, 2009; Clark and Poovendran, 2010).

In the context of wireless networks, preserving information security has three main goals: (i) confidentiality, i.e. ensuring that nobody but the receiver can see the information; (ii) integrity, i.e. ensuring that the information has not been modified, and (iii) availability, i.e. ensuring that anytime the information is needed, it would be available (Williams, 2007).

2 Our approach

We study the application of existing tools and mechanisms for preserving the goals of information security over a wireless network. We model communication exchange in a subset of typed first-order logic and analyse the model using the lightweight model analyser Alloy (Jackson, 2006).

Lightweight modelling is a design technique in which small, abstract formal models are explored and verified with a pushbutton tool. We build small, abstract models of a transmission of data over a wireless network, explore the properties of candidate models using a push-button analysis tool, and we verify aspects of the final model using the same tool.

The modelling language of Alloy is characterised as textual and declarative (Jackson, 2006). In an Alloy model basic entity types and relations on them are declared, constrains on entities and relations are specified, predicates are defined and assertions made. An instance of a model is a collection of entities and relations that satisfies all the model constraints. A scope assigns to each basic type a small integer. This is the maximum number of members that the type can have.

Alloy models contain signature *(sig)*, facts *(fact)*, functions *(fun)*, predicates *(pred)*, and assertions *(assert)*. The Alloy Analyzer works by enumerating all possible instances of a model within a scope. In this way it can find examples (instances that satisfy a predicate), counterexamples (instances that falsify an assertion), and verify assertions (within the given scope, all model instances satisfy the assertion). If there are no instances of a model, then the model is inconsistent. The concept of time is not built into Alloy, but there is a conventional way to model time, states, and se- quences of events as done in (Zave, 2009).

Our approach is inspired by network analysis using Alloy which has been shown to be a valuable design technique for protocols (Zave, 2009). Alloy models allow us to specify communication flow and characterise the participants in the communication.

*Corresponding author

2.1 Man in the middle attack

We study one of the most common network attacks, the man in the middle (MITM). MITM is an attacker who intercepts the message exchange by impersonating each end-point. He then controls the communication by making the participants believe that they are talking to each other directly in secure connection.

Approaches to thwart MITM attacks have been developed, for example cryptographic protocols put a form of endpoint authentication. SSL uses a mutually trusted certification authority to authenticate the server (Meyer and Wetzel, 2004).

In our network model we model three cases of communication between client and server. The complexity of the models is incremental.

The first case describes *ideal* communication between one client and one server in secure scope by sending and receiving an unencrypted message. The second case, describes the *ideal encrypted* communication between one client and one server in secure scope for sending and receiving a message which is encrypted and decrypted using the respective public and private keys. The third case describes the communication between one client and one server in unsecure scope which is intercepted by MITM.

We model both active and passive interception. Active interception results in altered communication by impersonation or changing the data; passive MITM attack is eavesdropping.

Assertions in this model relate to the time a request was sent, to the identities of both server and client, and to the message data. We can specify the behaviour of all participants and the sequence of events leading to compromised (unsecure) communication. We can, for example, specify that a participant in the communication is impersonated by MITM at a given time, or that MITM has intercepted a packet which includes an encrypted message, is able to decrypt the message using the server's private key, eavesdrops it and then encrypts it again using client's public key that he stored at a certain time in the communication.

When an assertion is violated, Alloy produces a counterexample which is useful for analysing the security of the network. For example, we can specify message exchange between one client and one server which use public and private keys. We use the following concepts: *time* defined a sequence of discrete events, all *participants* identifiable by their respective *keys*, and specific to the message attributes such as *header* and *data*. A signatures can inherit fields and constrains from another signature and can be defined as a subset of another signature. For example, the signature *time* extends the signature *packet* to inherit its fields and constrains. This allows us to specify that the packet was sent at a given time and to analyse whether the packet was received at the expected time by the expected recipient and without modification; the signature *IP* is defined as a subset of signature *header* and allows us to specify the source and the destination of a transmitted packet; the signature *key* includes three kinds of keys, i.e. the client's public and private keys, server's public and private keys and MITM's public and private keys which allows us to write assertions about the identity of the participants in the communication.

References

W. A. Arbaugh. Wired on wireless. *IEEE Security & Privacy*, 2(3), 2004.

L. Butty and J. Hubaux. Security and cooperation in wireless networks – thwarting malicious and selfish behavior in the age of ubiquitous computing, 2007.

A. Clark and R. Poovendran. A metric for quantifying key exposure vulnerability in wireless sensor networks. In *WCNC*, 2010.

D. Gritzalis and J. Lopez, editors. *Emerging Challenges for Security, Privacy and Trust*, volume 297 of *IFIP*, 2009.

D. Jackson. *Software Abstractions: Logic, Language, and Analysis*. The MIT Press, 2006.

B. Latré, B. Braem, I. Moerman, Ch. Blondia, and P. Demeester. A survey on wireless body area networks. *Wireless Networks*, 17(1), 2011.

U. Meyer and S. Wetzel. On the impact of gsm encryption and man-in-the-middle attacks on the security of interoperating gsm/umts networks. In *PIMRC*, 2004.

A. Pathan, T. Dai, and C. Hong. An efficient LU decomposition-based key pre-distribution scheme for ensuring security in wireless sensor networks. In *CIT*, 2006.

R. Williams. Introduction to information security concepts, 2007.

H. Yang, F. Ye, Y. Yuan, S. Lu, and W. Arbaugh. Toward resilient security in wireless sensor networks. In *MobiHoc*, 2005.

P. Zave. Lightweight modeling of network protocols in Alloy, 2009.

Compositional Reasoning for Processes and Data

Liam O'Reilly*
*Swansea University
Singleton Park, Swansea, SA2 8PP
csliam@swansea.ac.uk

Till Mossakowski†
†DFKI Lab Bremen
Enrique-Schmidt-Str. 5, D-28359 Bremen
Till.Mossakowski@dfki.de

Markus Roggenbach‡
‡Swansea University
Singleton Park, Swansea, SA2 8PP
csmarkus@swansea.ac.uk

Abstract

CSP-CASL is a specification language tailored to the description of distributed systems depending on proper treatment of data. Here, we discuss how to exploit its specification structure to ease verification.

1 Introduction

Specification in the *large* allows to model systems using components. In this context, algebraic specification has provided a rich variety of operators that allow to build complex specifications out of simpler ones (Bergstra et al., 1990). These operators include the union of specifications, the renaming and hiding of symbols within specifications, as well as parametrized specifications. In the context of data, these structuring mechanisms with their relations to proof calculi have been studied extensively (Bidoit et al., 1998). To the best of our knowledge, we are the first to apply these ideas to state based systems. Here, we develop proof calculi for the language CSP-CASL (Roggenbach, 2006) that exploit the specification structure to ease verification.

2 CSP-CASL

Distributed computer applications like web services, flight booking systems, and electronic payment systems involve the parallel processing of data. Consequently, these systems exhibit concurrent aspects (e.g. deadlock-freedom) as well as data aspects (e.g. functional correctness). Often, these aspects depend on each other. The algebraic specification language CASL (Mosses, 2004) alone can deal with data aspects, while the process algebra CSP (Roscoe, 2010) is quite capable of modelling concurrent systems. The mutual dependencies between processes and data, however, require an integrated language such as CSP-CASL.

A CSP-CASL specification consists of a *data part* and a *process part*. The data part establishes the data of the system, and declares the sorts, operations and predicates available. The process part then establishes the desired behaviour of the system components. Each component is specified as a CSP process, where CASL terms are used as communications, CASL sorts denote sets of communications, relational renaming is described by a binary CASL predicate, and the CSP conditional construct uses CASL formulae as conditions.

3 Structured CSP-CASL

Recently (O'Reilly et al., to appear), we extended CSP-CASL by the structuring mechanisms as known from algebraic specification. To this end, we formulated the language as a so-called institution. In order to cater for parametrization, we introduced the notion of "loose processes". As an example of structured as well as generic CSP-CASL consider the following specification of an online shopping system:

spec SHOP [*RefCl*(CUSTOMER)] [*RefCl*(WAREHOUSE)] [*RefCl*(PAYMENTSYSTEM)] [*RefCl*(COORDINATOR)] =
  process *System* : *C_C*, *C_W*, *C_PS* ;
    System = *Coordinator* ⟦ *C_C*, *C_W*, *C_PS* || *C_C*, *C_W*, *C_PS* ⟧
      (*Customer* ⟦ *C_C* || *C_W*, *C_PS* ⟧
        (*Warehouse* ⟦ *C_W* || *C_PS* ⟧ *PaymentSystem*))
end

The above describes how the overall *System* is composed out of four components, i.e. processes, namely *Customer*, *Warehouse*, *PaymentSystem*, and *Coordinator*. These are specified separately in specifications CUSTOMER, WAREHOUSE, PAYMENTSYSTEM, and COORDINATOR. The *RefCl* operator ensures that any refinement of these component specifications provides a valid actual parameter for SHOP. Concerning proof support, the question arises if such a parametrized specification allows for compositional reasoning.

4 Compositional reasoning

In (O'Reilly et al., to appear), we developed a set of proof rules that allow to decompose complex proof obligations on structured specifications into simpler ones. These rules exploit in a systematic way the specification building operators such as union, renaming, and hiding. Astonishingly enough, even proof rules out of the CSP context are "well-behaved" under these specification building operators. Here, we study especially the notions of *responsiveness* and *deadlock-freedom of networks* (Reed et al., 2004). A typical example of one such rule in our calculi is:

$$\frac{Server :: A\ ResToLive^{\checkmark}\ Client :: B\ on\ J\ in\ SP_1}{Server :: A\ ResToLive^{\checkmark}\ Client :: B\ on\ J\ in\ (SP_1\ \mathbf{and}\ SP_2)}$$

This proof rule (for the union operator, which syntactically is written as **and**) states that in order to show that a *Server* is responsive to a *Client* in the context of the specification $(SP_1\ \mathbf{and}\ SP_2)$, it is enough to establish responsiveness in the context of SP_1 alone. Here, A, B and J denote sets of events. This rule illustrates how to reduce the specification text involved in a proof. Others rules work on the processes involved. For example, in order to prove deadlock freedom of a network, we isolate a single process, establish that it is responsive, and finally prove deadlock freedom for the smaller network with the isolated process removed. Finally, we have rules that relate refinement and structuring.

5 Proof rules in practice

An extensive specification and verification exercise on the online shop example introduced above demonstrates that our techniques work in practice. Tool support for the calculi developed is well under way.

References

J.A. Bergstra, J. Heering, and P. Klint. Module algebra. *Journal of the ACM*, 37(2):335–372, 1990.

M. Bidoit, V.V. Cengarle, and R. Hennicker. Proof systems for structured specifications and their refinements. In *Algebraic Fondations of System Specification*. Springer, 1998.

P. D. Mosses, editor. CASL *Reference Manual*. LNCS 2960. Springer, 2004.

L. O'Reilly, T. Mossakowski, and M. Roggenbach. Compositional modelling and reasoning in an institution for processes and data. LNCS. Springer, to appear.

J.N. Reed, J.E. Sinclair, and A.W. Roscoe. Responsiveness of interoperating components. *Formal Aspect of Computing*, 16(4):394–411, 2004.

M. Roggenbach. CSP-CASL: A new integration of process algebra and algebraic specification. *Theoretical Computer Science*, 354(1):42–71, 2006.

A.W. Roscoe. *Understanding Concurrent Systems*. Springer, 2010.

Natural Deduction in the setting of Paraconsistent Logic

Alexander Bolotov[⋆]
[⋆]University of Westminster
A.Bolotov@wmin.ac.uk

Vasilyi Shangin[† *]
[†]Moscow State University
shangin@philos.msu.ru

1 Introduction

In this paper we continue our analysis of natural deduction calculi and corresponding proof procedures for non-classical logics. Our attention is now paid to the framework of paraconsistent logics Priest (2002). These logics are used, in particular, for reasoning about systems where paradoxes do not lead to the 'deductive explosion', i.e. where formulae of the type **false** $\Rightarrow A$, for any A, are not valid. Paraconsistent reasoning has rather old tradition in philosophy, but is quite young in automated reasoning, perhaps with first works related to 60th-70th Belnap (1977a), Belnap (1977b). Recently paraconsintent logics have been studied by a wider community with such application areas as inconsistency-tolerant and uncertainty reasoning than other types of logical systems Chen et al. (2007).

2 A sound and complete natural deduction system NPCont

We fix a standard propositional language L over an alphabet $p, q, r, p_1, q_1, r_1, \ldots$.

We adopt the standard definition of an L-formula and the following notation - letters $A, B, C, D, A_1 \ldots$ denote arbitrary L-formulae while letters $\Gamma, \Delta, \Gamma_1, \ldots$ denote arbitrary finite sets of L-formulae. For brevity we say 'formula' instead of 'L-formula'.

Logic PCont is studied in Rozonoer (1983), see, also, Popov (1989). Natural deduction system NPCont was defined in Shangin (2010).

In Figure 1 we define the sets of elimination and introduction rules, where prefixes '*el*' and '*in*' denote an elimination and an introduction rule, respectively.

Elimination Rules :

$\wedge el_1 \quad \dfrac{A \wedge B}{A} \qquad \wedge el_2 \quad \dfrac{A \wedge B}{B}$

$\neg \wedge \ el \quad \dfrac{\neg(A \wedge B)}{\neg A \vee \neg B} \qquad \vee el \quad \dfrac{A \vee B, \quad [A]C, [B]C,}{C}$

$\neg \vee \ el_1 \quad \dfrac{\neg(A \vee B)}{\neg A} \qquad \neg \vee \ el_2 \quad \dfrac{\neg(A \vee B)}{\neg B}$

$\Rightarrow el \quad \dfrac{A \Rightarrow B, \quad A}{B}$

$\neg \Rightarrow el_1 \quad \dfrac{\neg(A \Rightarrow B)}{A} \qquad \neg \Rightarrow el_2 \quad \dfrac{\neg(A \Rightarrow B)}{\neg B}$

$\neg \, el \quad \dfrac{\neg\neg A}{A}$

$PCont - \vee el \quad \dfrac{[A]\, C, [\neg A]\, C}{C}$

Introduction Rules :

$\wedge in \quad \dfrac{A, \quad B}{A \wedge B}$

$\neg \wedge \ in_1 \quad \dfrac{\neg A}{\neg(A \wedge B)} \qquad \neg \wedge \ in_2 \dfrac{\neg B}{\neg(A \wedge B)}$

$\vee in_1 \quad \dfrac{A}{A \vee B} \qquad \vee in_2 \quad \dfrac{B}{A \vee B}$

$\neg \vee \ in \quad \dfrac{\neg A, \neg B}{\neg(A \vee B)}$

$\Rightarrow \ in \quad \dfrac{[C] \quad B}{C \Rightarrow B}$

$\neg \Rightarrow in \quad \dfrac{A, \neg B}{\neg(A \Rightarrow B)}$

$\neg \, in \quad \dfrac{B}{\neg\neg B}$

Figure 1: NPCont-rules

[*†] The second author is supported by Russian Foundation for Humanities, project 10-03-00570a.

An inference in the system NPCont is a finite non-empty sequence of formulae with the following conditions: each formula is an assumption or is derived from the previous ones via a NPCont-rule; by applying $\Rightarrow_{in}$ in each formula starting from the last alive assumption until the result of the application of this rule, inclusively, is discarded from the inference; by applying $\vee_{el}$ each formula starting from assumption A until formula C, inclusively, as well as each formula starting from assumption B until formula C, inclusively, is discarded from the inference; by applying $PCont - \vee_{el}$ each formula starting from assumption A until formula C, inclusively, as well as each formula starting from assumption A until formula C, inclusively, is discarded from the inference. A proof in the system NPCont is an inference from the empty set of assumptions.

3 A proof searching algorithm for NPCont

We use heuristics and proof search strategies proposed earlier for classical and non-classical logics Bolotov et al. (June, 2007). We consider this paper as an adaptation of our previous results to paraconsistent logic PCont, i.e. our strategies defined for the classical setting work in the new setting in the goal-oriented fashion - any application of an introduction NPCont-rule is determined by the current goal from $list\text{-}proof$. The procedure is shown to be finite, and terminates either with finding a proof of the desired formula (the initial formula in $list\text{-}goals$) or with providing a counterexample, i.e. a valuation under which the desired formula is false is extractable from $listproof$. Therefore, both soundness and completeness of the proof search procedure are secured.

4 Example

With respect to the formulae which are not classical tautologies, the proof search procedure for NPCont works in the same way it works for the classical logic which reflects the semantic fact that the characteristic matrixes for PCont invalidate each formula which is not a classical tautology. Therefore, the distinguished characteristics of the procedure is how it deals with the formulae which are classical tautologies and are not PCont-valid. An example of such a formula is $(p \Rightarrow \neg q) \Rightarrow (q \Rightarrow \neg p)$.

list_proof	*list_goals*
1. $p \Rightarrow \neg q$ assumption	$q \Rightarrow \neg p$
2. q assumption	$\neg p$
3. p assumption	$[p]\ \neg p, [\neg p]\ \neg p$
4. $\neg q$ $\Rightarrow el,\ 1, 3$	

Goal $[p]\ \neg p$ cannot be reached and we can extract a counter example from $listproof$. However, has this goal been reached then the PCont- el rule is applicable which would have given us the desired goal $\neg p$.

References

N. Belnap. A useful four-valued logic: How a computer should think. In J.M. Dunn and G. Epstein, editors, *Modern Use of Multiple-valued Logic*. Dordrecht: D. Reidel, 1977a.

N. Belnap. *How a Computer Should Think.* Oriel Press, comtemporary aspects of philosophy edition, 1977b.

A. Bolotov, O. Grigoriev, and V. Shangin. Automated natural deduction for propositional linear-time temporal logic. In *Proceedings of the Time-2007, International Symposium on Temporal Representation and Reasoning*, June, 2007.

Donghuo Chen, Guangquan Zhang, and Jinzhao Wu. First joint ieee/ifip symposium on theoretical aspects of software engineering, tase 2007, june 5-8, 2007, shanghai, china. In *TASE*. IEEE Computer Society, 2007.

V. Popov. Sequential formulations for paraconsistent logics. *Syntactical and semantic investigations of nonextensional logics*, 1989. (In Russian).

G. Priest. Paraconsistent logic. In Gabbay and F. Guenthner, editors, *Handbook of Philosophical Logic, Second Edition*, volume 6. Dordrecht: Kluwer Academic Publishers, 2002.

L. Rozonoer. On finding contradictions in formal theories. *Automatica and telemekhanica*, (6), 1983. (in Russian).

V. Shangin. Natural deduction systems for logics Par, PCont, PComp and PContPComp. In *Logic, methodology: actual problems and perspectives*, 2010. (In Russian).

Solving rule admissibility problem for S4 by a tableau method

Renate Schmidt*
*School of Computer Science,
The University of Manchester, UK
renate.schmidt@manchester.ac.uk

Dmitry Tishkovsky†
†School of Computer Science,
The University of Manchester, UK
dmitry.tishkovsky@manchester.ac.uk

Abstract

Rules that are admissible can be used in any derivations in any axiomatic system of a logic. In this paper we introduce a method for checking the admissibility of rules in the modal logic S4. Our method is based on a standard semantic ground tableau approach. In particular, we reduce rule admissibility in S4 to satisfiability of a formula in a logic that extends S4. The extended logic is characterised by a class of models that satisfy a variant of the co-cover property. The class of models can be formalised by a well-defined first-order specification. Using a recently introduced framework for synthesising tableau decision procedures this can be turned into a sound, complete and terminating tableau calculus for the extended logic, and gives a tableau-based method for determining the admissibility of rules.

1 Introduction

Logical admissible rules were first considered by Lorenzen (1955). Initial investigations were limited to observations on the existence of interesting examples of admissible rules that are not derivable. The area gained new momentum when Friedman (1975) posed the question whether algorithms exist for recognising whether rules in intuitionistic propositional logic IPC are admissible. This problem and the corresponding problem for modal logic S4 are solved affirmatively in Rybakov (1984).

In this paper we focus on S4 and introduce a new method for recognising the admissibility of rules. Our method is based on the reduction of the problem of rule admissibility in S4 to the satisfiability of a certain formula in an extension of S4. We refer to the extended logic as $\mathsf{S4}^{\mathsf{u},+}$. $\mathsf{S4}^{\mathsf{u},+}$ can be characterised by a class of models, which satisfy a variant of the co-cover property that is definable by modal formulae. This property is expressible in first-order logic and means that the semantics of the logic can be formalised in first-order logic. We exploit this property in order to devise a tableau calculus for $\mathsf{S4}^{\mathsf{u},+}$ in a recently introduced framework for automatically synthesising tableau calculi and decision procedures (Schmidt and Tishkovsky, 2008, 2009).

The tableau synthesis method of Schmidt and Tishkovsky (2009) consists of two steps: (i) the user defines the formal semantics of the given logic in a many-sorted first-order language so that certain well-definedness conditions hold, (ii) the method automatically reduces the semantic specification of the logic to Skolemised implicational forms which are then rewritten as tableau inference rules. These are combined with some default closure and equality rules.

The set of rules obtained in this way provides a sound and constructively complete calculus. Furthermore, this set of rules automatically has a subformula property with respect to a finite subformula operator. If the logic can be shown to admit finite filtration with respect to a well-defined first-order semantics then adding a general blocking mechanism produces a terminating tableau calculus (Schmidt and Tishkovsky, 2008).

2 A Tableau Method for Testing Rule Admissibility

The algorithm for testing rule admissibility is based on two tableau calculi: $T_{\mathsf{S4}^{\mathsf{u}}}$ for deciding the logic $\mathsf{S4}^{\mathsf{u}}$ which extends S4 with universal modality $[\mathsf{u}]$ and $T_{\mathsf{S4}^{\mathsf{u},+}}$ for deciding the extended logic $\mathsf{S4}^{\mathsf{u},+}$. The calculus $T_{\mathsf{S4}^{\mathsf{u},+}}$ extends the calculus $T_{\mathsf{S4}^{\mathsf{u}}}$ with the infinite set of rules which ensure the co-cover property. The rules are listed in Figure 1 and contain a number of Skolem functions h_n and g_n for $n > 0$ and a Skolem constant g_0. Notice that these rules are non-standard since Skolem symbols $g_0, \ldots, g_n$ occur in their premises. Both calculi are synthesised and refined with use of the method of Schmidt and Tishkovsky (2009) and, hence, sound and complete for corresponding logics. By a filtration argument, it can be proved that $\mathsf{S4}^{\mathsf{u}}$ and $\mathsf{S4}^{\mathsf{u},+}$ have the bounded finite model property. Based on these results, the termination for these calculi is achieved by using the unrestricted blocking mechanism (Schmidt and Tishkovsky, 2008).

The algorithm for testing rule admissibility in S4 is the following:

Step 1. Given an S4-rule α/β, rewrite it to $[\mathsf{u}]\alpha \wedge \neg\beta$.

Step 2. Use the tableau calculus $T_{\mathsf{S4}^{\mathsf{u}}}$ to test the satisfiability of $[\mathsf{u}]\alpha \wedge \neg\beta$ in $\mathsf{S4}^{\mathsf{u}}$.

Step 3. If $T_{\mathsf{S4}^{\mathsf{u}}}([\mathsf{u}]\alpha \wedge \neg\beta)$ is closed, i.e., $[\mathsf{u}]\alpha \wedge \neg\beta$ is unsatisfiable in $\mathsf{S4}^{\mathsf{u}}$, then stop and return 'derivable'.

Step 4. Otherwise (i.e., $T_{\mathsf{S4}^{\mathsf{u}}}([\mathsf{u}]\alpha \wedge \neg\beta)$ is open), *continue* the tableau derivation with the rules in $T_{\mathsf{S4}^{\mathsf{u}}}$ plus the rules (cc'^{0}_{0}), (cc'^{0}_{1}), (cc'^{n}_{0}), and (cc'^{n}_{1}). In particular, continue the derivation with a finite open branch of $T_{\mathsf{S4}^{\mathsf{u}}}([\mathsf{u}]\alpha \wedge \neg\beta)$ using the rules of $T_{\mathsf{S4}^{\mathsf{u},+}}$ until the derivation stops.

$$(\mathsf{cc'}^0_0)\colon \frac{}{@_{g_0} g_0} \qquad (\mathsf{cc'}^0_1)\colon \frac{@_{g_0}\Diamond i,\ @_i p}{@_{g_0} p} \qquad (\mathsf{cc'}^n_0)\colon \frac{@_{i_1} i_1,\ \ldots,\ @_{i_n} i_n}{@_{g_n(\bar{i})}\Diamond i_1,\ \ldots,\ @_{g_n(\bar{i})}\Diamond i_n}$$

$$(\mathsf{cc'}^n_1)\colon \frac{@_{g_n(\bar{i})}\Diamond j,\ @_j p}{@_{g_n(\bar{i})} p \mid @_{i_1}\Diamond h_n(p,\bar{i},j),\ @_{h_n(p,\bar{i},j)} p \mid \cdots \mid @_{i_n}\Diamond h_n(p,\bar{i},j),\ @_{h_n(p,\bar{i},j)} p}$$

Figure 1: Infinite set of rules to capture the co-cover property.

Step 5. If $T_{\mathsf{S4^{u,+}}}([\mathsf{u}]\alpha \wedge \neg\beta)$ is closed then return 'not derivable and admissible'. Otherwise, i.e., if $T_{\mathsf{S4^{u,+}}}([\mathsf{u}]\alpha \wedge \neg\beta)$ is open, return 'not admissible'.

The answers returned by the method are either 'derivable', 'not derivable and admissible', or 'not admissible'. If a rule is derivable it is also admissible but not conversely.

3 Concluding Remarks

A major difficulty in dealing with S4-admissibility, is that the theory of S4-admissible rules does not have the finite approximation property. Therefore the tableau algorithm that we have introduced in this paper builds open branches that represent a (possibly) infinite S4-model for the S4-admissible rules and is a counter-model to a non-admissible rule. The subsequent filtration provides a finite model (not necessarily a model for all admissible rules) witnessing the refutation.

Algorithms deciding admissibility for some transitive modal logics and IPC, based on projective formulae and unification, are described in (Ghilardi, 2002). They combine resolution and tableau approaches for finding projective approximations of a formula and rely on the existence of an algorithm for theorem proving. A practically feasible realisation for S4 built on the algorithm for IPC in (Ghilardi, 2002) is described in (Zucchelli, 2004). These algorithms were specifically designed for finding general solutions for matching and unification problems. In contrast, the original algorithm of Rybakov (1984) can be used to find only *some* solution of such problems in S4. In particular, this can be done through the relation: an equation $\alpha \equiv \beta$ is solvable iff a rule $\alpha \equiv \beta/\bot$ is not admissible. We expect that our method can be modified for finding the general solutions of logical equations as well.

Rybakov (2007) showed that the logic $\mathsf{S4^u}$ is decidable with respect to admissibility. We strongly believe that our tableau method can also be extended to rule admissibility in $\mathsf{S4^u}$.

In fact, our method of replacing the first-order co-cover condition with its *formulaic* variant (which is also first-order but in the extended language) is not restricted to S4. We expect that it can be modified to deal with a number of other modal logics, especially those covered by the general theory of (Rybakov, 1997). Similarly, it can be applied to their superintuitionistic counterparts (either through Gödel's translation or directly) and to transitive modal logics augmented with the universal modality.

Full details of the presented results can be found in (Babenyshev et al., 2010).

References

S. Babenyshev, V. Rybakov, R. A. Schmidt, and D. Tishkovsky. A tableau method for checking rule admissibility in S4. *Electron. Notes Theor. Comput. Sci.*, 262:17–32, 2010.

H. Friedman. One hundred and two problems in mathematical logic. *J. Symb. Log.*, 40(3):113–130, 1975.

S. Ghilardi. A resolution/tableaux algorithm for projective approximations in IPC. *Log. J. IGPL*, 10(3):229–243, 2002.

P. Lorenzen. *Einführung in die operative Logik und Mathematik*. Springer, Berlin-Göttingen, Heidelberg, 1955.

V. V. Rybakov. A criterion for admissibility of rules in modal system S4 and the intuitionistic logic. *Algebra Logic*, 23(5):369–384, 1984.

V. V. Rybakov. *Admissibility of logical inference rules*, volume 136 of *Studies in Logic and the Foundations of Mathematics*. Elsevier, New-York–Amsterdam, 1997.

V. V. Rybakov. Logics with universal modality and admissible consecutions. *J. Appl. Non-Classical Log.*, 17 (3):381–394, 2007.

R. A. Schmidt and D. Tishkovsky. Automated synthesis of tableau calculi. In *Proc. TABLEAUX'09*, volume 5607 of *LNAI*, pages 310–324. Springer, 2009.

R. A. Schmidt and D. Tishkovsky. A general tableau method for deciding description logics, modal logics and related first-order fragments. In *Proc. IJCAR'08*, volume 5195 of *LNCS*, pages 194–209. Springer, 2008.

D. Zucchelli. Studio e realizzazione di algoritmi per l'unificazione nelle logiche modali. Laurea specialistica in informatica (Masters Thesis), Università degli Studi di Milano, July 2004.

Typestate modelling and verification with Hanoi

Iain McGinniss*
*School of Computing Science
University of Glasgow
iainmcg@dcs.gla.ac.uk

Simon Gay*
simon@dcs.gla.ac.uk

Abstract

The interface of an object can be viewed as a form of contract between the user of the interface and the implementation of the interface. Often the details of such contracts in a language like Java are a mixture of formal specification (the type information of the methods) and informal specification, embedded in javadoc comments. Informal specification is dangerous for two reasons: ambiguity and opaqueness to automated reasoning tools. We present Hanoi, a specification language for typestate on Java interfaces. Hanoi models are designed to be readable by both humans and machines, with a formal semantics to aid verification of interface usage and implementation.

1 Introduction

Object oriented languages, such as Java, provide many complex APIs which are fundamentally stateful — the behaviour and capabilities of an object representing access to some external entity change as the object is used, and as external conditions change. This impacts the usage of such objects, in that the set of methods which may legally be called varies over time. This behaviour can typically be captured by a finite state machine — this is known as "typestate", the combination of types and state machines which describe their restrictions (Strom and Yemini, 1986).

It has been recently discovered that Java APIs with typestate restrictions are more frequent than those with generic parameters (Beckman et al., 2011). Yet, there is no support for expressing typestate restrictions formally in Java and as such the state transitions and restrictions are documented informally. This prevents any kind of verification, whether static or dynamic, than an interface is being used as intended by client code. Diligent programmers will write implementations that defend against invalid usage by manually checking invariants and throwing runtime exceptions if the restrictions are violated, however writing such code is tedious, error prone and costly.

2 Hanoi

Hanoi is a domain specific language for the specification of finite state machines that control the set of legal methods on an object, and expressing state transitions in response to method calls. An example of the Hanoi specification for the Java Iterator interface is shown in Listing 1. Each state defines a set of legal methods and conditional transitions to other states based upon the value returned when that method is called. The state machine is also hierarchical, in that a child state inherits the legal set of method calls defined in the parent and can extend and override this sucject to some restrictions.

The Iterator specification in Listing 1 defines four states: `CHECK_NEXT`, `NEXT_AVAILABLE`, `CAN_REMOVE` and `CAN_REMOVE_MIDDLE`. `CHECK_NEXT` is the root state, from which `NEXT_AVAILABLE` and `CAN_REMOVE` inherit which is indicated by their definition within the body of `CHECK_NEXT`. The definition of `CHECK_NEXT` indicates that the `hasNext()` method may be called, and that if it returns `true` then the object will move to state `NEXT_AVAILABLE`, otherwise it will remain in the present state. The `NEXT_AVAILABLE` state, adds one extra legal method, `next()`, while the `CAN_REMOVE` state allow for the `remove()` method to be called.

`CAN_REMOVE` provides an example of transition overriding — the successor state after a call to `hasNext()` when `true` is returned is changed from `NEXT_AVAILABLE` to `CAN_REMOVE_MIDDLE`. This is legal in Hanoi as the new successor is a substate of the original state. Essentially, Hanoi requires that the successor state be covariant, in order to ensure safe substitutability of child states for parent states.

The intention behind Hanoi is that such specifications provide the rules governing the usage of an object in a form that is unambiguous, and readable by both humans and programs. The syntax and semantics are simple yet allow for the expression of the typestate restrictions found on most interfaces in Java which have state-based preconditions.

Hanoi differs from existing work through it's use of a DSL for specification, rather than embedded code annotations that describe method pre- and post-conditions as in Plural (Bierhoff et al., 2009), or regular expressions that capture illegal sequences of method calls as in AspectJ trace matches (Allan et al., 2005). We intend to perform a user trial of Hanoi in the near future to determine whether the use of a DSL makes Hanoi models easier to understand than code annotations.

Listing 1: Hanoi model for java.util.Iterator

```
1  CHECK_NEXT {
2    NEXT_AVAILABLE {
3      CAN_REMOVE_MIDDLE { remove() -> NEXT_AVAILABLE }
4      next() -> CAN_REMOVE
5    }
6    CAN_REMOVE {
7      remove() -> CHECK\_NEXT
8      hasNext() :: true -> CAN_REMOVE_MIDDLE
9    }
10   hasNext() :: true -> NEXT_AVAILABLE
11   hasNext() :: <other> -> <self>
12 }
```

Listing 2: Iterator usage

```
1 boolean r = shouldRemove();
2 Iterator i = list.iterator(); // state = CHECK_NEXT
3 if(i.hasNext()) {
4     i.next();
5     if(r) {
6         i.remove();
7     } // state = CHECK_NEXT or CAN_REMOVE
8 }
```

3 Static Analysis of client code

The hierarchical structure and safe substitutability of child states for parent states in Hanoi makes the implementation of a data flow analysis for Hanoi straightforward - the tree structure of the inheritance hierarchy of the states provides a natural merge operation. For instance, on Line 7 of Listing 2, after the `if` statement the iterator will either be in state `CHECK_NEXT` or `CAN_REMOVE` depending on the value of the boolean `r`. We can merge these to `CHECK_NEXT`, which is the least upper bound of both states.

Such static analysis is sound if we can be sure that there are no aliases to the stateful object in question, otherwise unobserved state changes may occur in another context. Alias control is not part of the Hanoi model, as such the current implementation is limited to dynamic checking through byte-code weaving. Aliasing guarantees could in principle be provided by some other static analysis which a Hanoi static analysis could leverage, and we are presently pursuing this idea as part of our future work.

References

Chris Allan, Pavel Avgustinov, Aske Simon Christensen, Laurie Hendren, Sascha Kuzins, Ondej Lhoták, Oege de Moor, Damien Sereni, Ganesh Sittampalam, and Julian Tibble. Adding trace matching with free variables to AspectJ. In *Proceedings of the 20th annual ACM SIGPLAN conference on Object oriented programming systems languages and applications - OOPSLA '05*, page 345, New York, New York, USA, 2005. ACM Press. ISBN 1595930310. doi: 10.1145/1094811.1094839. URL `http://portal.acm.org/citation.cfm?doid=1094811.1094839`.

Nels E. Beckman, Duri Kim, and Jonathan Aldrich. An Empirical Study of Object Protocols in the Wild. In *Proceedings of the European Conference on Object-Oriented Programming (ECOOP '11)*, 2011.

K. Bierhoff, N.E. Beckman, and J. Aldrich. Practical API Protocol Checking with Access Permissions. In *23rd European Conference on Object-Oriented Programming*, volume 5653 of *Lecture Notes in Computer Science*, pages 195–219, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg. ISBN 978-3-642-03012-3. doi: 10.1007/978-3-642-03013-0. URL `http://www.springerlink.com/index/10.1007/978-3-642-03013-0`.

Robert E Strom and Shaula Yemini. Typestate: A programming language concept for enhancing software reliability. *IEEE Transactions on Software Engineering*, 12(1):157–171, 1986.

The Logical Verification of Security Protocols

David Berry, Michael Fisher and Clare Dixon

Department of Computer Science,
University of Liverpool,
Liverpool, L69 3BZ, UK
[D.R.Berry, mfisher, cldixon]@liverpool.ac.uk

Abstract

Here we describe the specification and verification of security protocols using temporal logics of knowledge.

1 Introduction

Security protocols are distributed programs which aim to provide secure communication between two or more parties. This aim is typically achieved through encryption techniques. Techniques such as RSA (Rivest et al., 1978) are based on computationally hard problems such as the factorisation of large prime numbers. Essentially nothing better than a brute force attack can break such encryption schemes. It has long been known, however, that one does not have to necessarily break the encryption to unlock the secret communication. Security protocols are famously difficult to design, subtle flaws can render a protocol completely vulnerable to one of many attacks. One of the most studied of these vulnerabilities was discovered in the Needham-Schroeder protocol by Lowe (Lowe, 1996). The Needham-Schroeder protocol is as follows, message 1. A $\rightarrow$ B : $\{$A, na$\}_{pkb}$ (A sends his id and fresh nonce to B, encrypted with B's public key), message 2. B $\rightarrow$ A : $\{$na, nb$\}_{pka}$ (B sends na and fresh nonce back to A, encrypted with A's public key), message 3. A $\rightarrow$ $\{$nb$\}_{pkb}$ (A sends nb back to B, encrypted with B's public key). At first glance this looks secure, Lowe's attack uses two runs of the protocol, leading to an intruder finding out all secret messages (the nonces).

Much work has been done on the verification of security protocol, among the most influential is formal methods, in particular model checking and theorem proving. We will focus on the logical verification of security protocols. In particular we are interested in specifying protocols in Temporal Logics of knowledge, and using theorem proving techniques to prove specific properties of these protocols hold. Work in this area is currently very active, some very public examples of flawed protocols have been highlighted (Murdoch et al., 2010). There is a necessity to formally prove that attacks can not happen on a protocol.

2 Logical Verification (for security)

The benefit of using logical techniques for the formal verification of security protocols is well established. BAN logic (Burrows et al., 1990) being the first of many. BAN allows one to reason about simple beliefs and knowledge of the principals communicating. Subsequently other logics have been applied to the problem of security analysis. Most recently temporal epistemic logics have been applied. In (Dixon et al., 2004) the authors show how temporal epistemic logic (LTL + S5) can be used to model a protocol, this is then verified using the TeMP theorem prover. Others have also used epistemic properties to model the knowledge transferred during the execution of a cryptographic protocol. The MCMAS tool (Lomuscio et al., 2009), is a model checker which has been used to prove the temporal epistemic properties of security protocols.

It is believed that having the knowledge axioms of the S5 logic is beneficial in the representation of security protocols. One aim of this research is to assess the benefits of using this logic in the specification of protocols. Particular attention will be given to the types of protocol which can be represented in this logic, and the properties which can be checked using theorem proving techniques. A lot of our initial focus has been specifying security properties in a logical unambiguous way, eventually we will have a hierarchy of related security properties which can be compared with one another in a similar way to Lowe's authentication hierarchy (Lowe, 1997).

A problem encountered in (Dixon et al., 2004), was the specification of a relatively simple protocol (Needham-Schroeder with public keys) was an arduous, error-prone task leading to a large and complex formalism. This is true of most formal representations of protocols I have encountered (for example using CSP). In order to counter this problem several leading tools in security protocol verification, such as CASPER (Lowe, 1998), allow users to specify protocols in

a high level abstract representation which is automatically transformed. This is a logical step to take for the verification technique we are using, many frustrated hours will be saved if a high level language is utilised.

3 High Level Protocol Specification

A number of high level languages have been suggested for the specification of security protocol, notably CAPSL (Millen and Denker). The syntax of CAPSL is very similar to the Alice-Bob notation many people are familiar with. For example A $\rightarrow$ B : $\{A, na\}_{pkb}$ represents A sending a message "m" to B, encrypted with a key "pkb". This representation is much easier to read and write, leading to more concise, shorter and understandable specifications. Currently we are working with temporal logics of knowledge which leads to a verbose protocol representation. The same message is represented as follows in our specification language.

```
send(A, B, encrypt(M1, pkb)) => sometime(receive(B, M1))
receive(B, M1) => K(b, hasValueOf(M1))
(K(b, hasValueOf(M1)) & encrypt(M1, pkb) & decrypt(M1, pkb, m1) & K(b, hasValueOf(skb)))
  | K(b, hasValueOf(m1)) <=> next(K(b, hasValueOf(m1)))
```

This is the core of the message sending, obviously the former representation is much simpler and less error prone.

We are working on a translation tool, which given a protocol in a CAPSL-like language will transform the input into a temporal epistemic specification. This work is still in preliminary stages with much work still to be done on specifying and proving a protocol correct in the temporal epistemic logic.

4 Future Direction

Currently the direction of our project is focused on using theorem proving techniques. The logics we are using could be extended. Incorporating belief into the system would have some obvious advantages in the specification of cryptographic protocols. We might want to reason about what a principal may believe, a less strict requirement that knowing something to be true. The aim of the project is to verify that no attacks exists on protocols, if an attack is possible a user friendly diagram and/or description will be made available. A major part of this research will be tracing the output from the theorem prover (which is large, confusing and messy) into a form which is intuitive and easier to understand.

References

M. Burrows, M. Abadi, and R. Needham. A Logic of Authentication. *ACM TRANSACTIONS ON COMPUTER SYSTEMS*, 8:18–36, 1990.

C. Dixon, M. F. Gago, M. Fisher, and W. van der Hoek. Using temporal logics of knowledge in the formal verification of security protocols. *Temporal Representation and Reasoning, International Syposium on*, 0:148–151, 2004. ISSN 1530-1311.

A. Lomuscio, H. Qu, and F. Raimondi. MCMAS: A Model Checker for the Verification of Multi-Agent Systems. In *Computer Aided Verification*, Lecture Notes in Computer Science, pages 682–688. Springer, 2009.

G. Lowe. Breaking and Fixing the Needham-Schroeder Public-Key Protocol Using FDR. In *TACAs '96: Proceedings of the Second International Workshop on Tools and Algorithms for Construction and Analysis of Systems*, pages 147–166, London, UK, 1996. Springer-Verlag. ISBN 3-540-61042-1.

G. Lowe. A hierarchy of authentication specifications. *Computer Security Foundations Workshop, IEEE*, 0:31, 1997. ISSN 1063-6900. doi: http://doi.ieeecomputersociety.org/10.1109/CSFW.1997.596782.

G. Lowe. Casper: A Compiler for the Analysis of Security Protocols. In *Journal of Computer Security*, pages 53–84. Society Press, 1998. `http://www.comlab.ox.ac.uk/gavin.lowe/Security/Casper/`.

J. Millen and G. Denker. CAPSL and MuCAPSL. `www.csl.sri.com/~millen/capsl/papers/jtitpub.ps`.

S. J. Murdoch, S. Drimer, R. J. Anderson, and M. Bond. Chip and pin is broken. In *IEEE Symposium on Security and Privacy*, pages 433–446, 2010.

R. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public key cryptosystems. *Communications of the ACM*, pages 120–126, 1978.