Proceedings of the Automated Reasoning Workshop 2009 Bridging the Gap between Theory and Practice ARW 2009

21st–22nd April 2009
University of Liverpool
Liverpool, United Kingdom

Editor:
Ullrich Hustadt

Technical Report ULCS-09-007
Department of Computer Science
University of Liverpool

UNIVERSITY OF LIVERPOOL

Organising Committee

Clare Dixon	University of Liverpool
Jacques Fleuriot	University of Edinburgh
Alexander Bolotov	University of Westminster
Simon Colton	Imperial College London
David Crocker	Escher Technologies
Louise Dennis	University of Liverpool
Roy Dyckhoff	University of St Andrews
Ullrich Hustadt	University of Liverpool
Mateja Jamnik	University of Cambridge
Tom Melham	University of Oxford
Alice Miller	University of Glasgow
Renate Schmidt	University of Manchester
Volker Sorge	University of Birmingham

Workshop Website

`http://www.csc.liv.ac.uk/~arw09/`

Preface

This volume contains the proceedings of ARW 2009, the sixteenth Workshop on Automated Reasoning, held 21st–22nd April 2009, in Liverpool, England (UK). As for the previous events in this series, this workshop 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.

These proceedings contain the abstracts of two invited talks, by Konstantin Korovin (University of Manchester), on “Instantiation-Based Automated Reasoning for First-Order Logic”, and Stefan Szeider (Durham University), on “Finding Hidden Structure in Reasoning Problems”, and twentyone extended abstracts contributed by participants of the workshop.

The abstracts cover a wide range of topics including the translation of Grid systems into state-based formal specifications natural deduction calculi for temporal logics, piecewise fertilisation for inductive proofs, techniques for instantiation based reasoning, explanations for entailments in description logics, ‘exactly one’ epistemic logic, verification of pervasive systems, reasoning about real-time system specifications, resolution calculi and resolution-based theorem provers for monodic first-order temporal logic and CTL, modal logic correspondence theory and second-order reduction, the embedding of Promela-Lite into the general purpose theorem prover PVS, fully automated techniques for symmetry reduction of partially symmetric systems, the classification of quasigroup-structures with respect to their cryptographic properties, diagrammatic reasoning and its use for software verification, first-order logic concept symmetry for theory formation, model checking auctions, coalitions and trust, synthesising tableau decision procedures, the application of combinations of machine learners, theorem provers, and constraint solvers in order to solve real world scenarios with incomplete background knowledge.

I would like to thank the members of the ARW Organising Committee for their advice. I would also like to thank all the colleagues who have helped with the local organisation, namely, Helen Bradley, Kenneth Chan, Patrick Colleran, Clare Dixon, Louise Dennis, Judith Lewa, Dave Shield, Lisa Smith, and Thelma Williams.

Liverpool
April 2009

Ullrich Hustadt

Invited Talks

Contributed Abstracts

Instantiation-Based Automated Reasoning For First-Order Logic

Konstantin Korovin[⋆]

[⋆]The University of Manchester
korovin@liverpool.ac.uk

Instantiation-based automated reasoning aims to utilise industrial-strength SAT and SMT technologies in the more general context of first-order logic. The basic idea behind instantiation-based reasoning is to combine clever generation of instances with satisfiability checking of ground formulae.

There are a number of challenges arising in this area ranging from theoretical issues such completeness, integration of theories and decidable fragments to efficient implementation: inference strategies, indexing and redundancy elimination. In this talk I will overview the recent advances in instantiation-based reasoning, focusing on a modular approach which allows us to use off-the-shelf SAT and SMT solvers for ground formulae as part of general first-order reasoning.

Finding Hidden Structure in Reasoning Problems (Invited Talk)

Stefan Szeider[⋆]
[⋆]Department of Computer Science
Durham University
Durham DH1 3LE, UK
stefan.szeider@durham.ac.uk

Abstract

Heuristic methods work often quite well on real-world instances of computational reasoning problems, contrasting the theoretical intractability of the problems. For example, state-of-the art satisfiability solvers solve routinely real-world instances with hundreds of thousands of variables; however no algorithm is known to solve instances with n variables in time $2^{o(n)}$ (and it is believed that such algorithm does not exist). This discrepancy between theoretical performance guarantees and empirically observed problem hardness causes a huge gap between theory and practise. It is a widely accepted view that the good practical performance of heuristic methods is due to a "hidden structure" present in real-world problem instances. However, classical complexity theory focuses on one dimension only, the input size, and does not provide a suitable framework for studying how a hidden structure influences the hardness of a reasoning problem.

In this talk I will advocate the framework of *Parameterized Complexity* (Downey and Fellows, 1999; Flum and Grohe, 2006; Niedermeier, 2006) for the theoretical study of computational reasoning problems. Parameterized complexity considers, in addition to the input size, a secondary measure, the parameter. The parameter can represent, among others, the degree of hidden structure present in a reasoning problem. This two-dimensional setting allows a more fine grained complexity analysis with the potential of being closer to problems as they appear in the real world, and thus to narrow the gap between theory and practise. The central notion of Parameterized Complexity is *fixed-parameter tractability* which refers to solvability in time $f(k)n^c$, where n denotes the input size, f is some function (usually exponential) of the parameter k and c is a constant. The subject splits into two complementary questions, each has its own mathematical toolkit and methods: how to design and improve fixed-parameter algorithms, and how to gather evidence that a problem is not fixed-parameter tractable for a certain parameterization.

The reasoning problems considered in this talk will include problems that arise in satisfiability solving, model counting, quantified Boolean formula evaluation, nonmonotonic reasoning and constraint satisfaction. I will mainly focus on two groups of parameters: parameters that represent *decomposability* of problems, and parameters that represent the existence of reasoning shortcuts through the search space in terms of *backdoors*. Most topics of the talk are covered by recent surveys (Gottlob and Szeider, 2006; Samer and Szeider, 2009).

References

Rodney G. Downey and Michael R. Fellows. *Parameterized Complexity*. Monographs in Computer Science. Springer Verlag, 1999.

Jörg Flum and Martin Grohe. *Parameterized Complexity Theory*, volume XIV of *Texts in Theoretical Computer Science. An EATCS Series*. Springer Verlag, 2006.

Georg Gottlob and Stefan Szeider. Fixed-parameter algorithms for artificial intelligence, constraint satisfaction, and database problems. *The Computer Journal*, 51(3):303–325, 2006.

Rolf Niedermeier. *Invitation to Fixed-Parameter Algorithms*. Oxford Lecture Series in Mathematics and its Applications. Oxford University Press, 2006.

Marko Samer and Stefan Szeider. Fixed-parameter tractability. In Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors, *Handbook of Satisfiability*, chapter 13, pages 425–454. IOS Press, 2009.

State-Based Behavior Specification for GCM Systems[*]

Alessandro Basso, Alexander Bolotov, Vladimir Getov

[*]School of Computer Science, University of Westminster, Watford Road, Harrow HA1 3TP

a.basso@wmin.ac.uk,bolotoa@wmin.ac.uk,v.s.getov@wmin.ac.uk

Abstract

This paper is in the area of automata-based formalisms of stateful systems. In particular, we have analyzed aspects of Grid systems which can be considered when developing a prototype for dynamic reconfiguration. We describe which parts of a Grid system can be utilized and translated into state-based formal specification, and how the subsequent deductive verification tests for the dynamic reconfiguration can be performed.

Introduction. The Grid Component model (GCM) developed by the CoreGRID project (see Baude et al., 2009) is a purpose-built component model to construct Grid applications. It has been adopted by the Grid Integrated Development Environment (GIDE), allowing for easy composition, monitoring and steering of Grid systems (see Basukoski et al., 2008). The features exposed in the GIDE allow us to use formal specification language and deductive reasoning verification methods in the framework of automated dynamic reconfiguration. It is essential that the properties of a Grid system, starting from the components stateful properties, and including the ones of resources (which we have collected under the umbrella of the Environment) can be monitored and reported in order to be able to comprehend the current states scenario. When monitoring a program we considered the impact on complexity that this type of specification will create in respect to execution time as well as memory consumption. We have devised a process of repetitive static analysis to minimize the impact by optimizing the program instrumentation. We can therefore monitor at runtime only a small section of the components's specification – the behavior of the stateful system – and leave the proofs of the inner functionality of primitive components' behavior to other methods, as in Barros et al. (2005). In this paper we describe how each section of the Grid system – and the environment it lays on – can be translated into formal specification and fed into a resolution based proof engine. Our final aim is to give a response to the tool we are developing within the GIDE indicating to whether the reconfiguration can take place and how.

Automata-based formalism. We began our development on the techniques expressed in Basso et al. (2008a), but changed our path to simplify development by redefining our concept of a two layers automata to a single one. We considered a simple finite state automaton on finite strings, and applied a set of specification "patterns" (following the sections described in the next section). The automata is used for the creation of labels defining various states in which the considered components and resources can be, the derived model is then directly specified in the normal form for CTL, SNF_{CTL}, developed by Bolotov and Fisher (1999). It was shown in Bolotov et al. (2002) that a Buchi word automaton can be represented in terms of SNF_{PLTL}, a normal form for PLTL. In a similar fashion we have represented a Buchi tree automaton in terms of SNF_{CTL}; we use $\text{SNF}_{\text{CTL}}$to specify the tree automaton and (in simulations) extend this specification to the deontic temporal specification in Basso et al. (2008b). Further, we apply a resolution-based verification technique as a verification procedure using the tool in Zhang et al. (2008).

Formalizing components and resources. When considering what parts in the GCM can be used for formal specification, we have considered four main sections, each of which follows specific criteria and can be easily fed into to a table of specification "patterns". We examine the main details below. Please not that not full specification is included for space reasons. As an example, we consider an Application (the outmost component which must be activated first) which contains 4 components Comp1 (a composite component with a sub component SubComp1.1) which is the first to be started after the application is as it is the first and only component, two components CompA and CompB running in parallel from a broadcast of Comp1 (and SubComp1.1 to start in parallel with CompA or CompB), and Comp2 a component from the gathercast of CompA and CompB.

Hierarchical Components Structure. Components in the GCM have a strict hierarchical nature. The application can then be described as: $\texttt{start} \Rightarrow Application$ and components of the application in the form of: $Application \Rightarrow \mathbf{A}\bigcirc Comp1$, $Application \Rightarrow \mathbf{A}\Diamond Comp2$, $Comp1 \Rightarrow \mathbf{A}\bigcirc(SubComp1.1 \wedge (CompA \vee CompB))$

Inferring parallel processes from interfaces. When we consider interfaces in the GCM, we can group them in two different types: one to one, and broadcast/gathercast. In the former we have a simple connection of one server interface to a client one, while in the latter we have a single server interface which can be bound to multiple client ones.

[*]This research has been carried out under the European research and development project GridCOMP (Contract IST-2005-034442) funded partially by the European Commission.

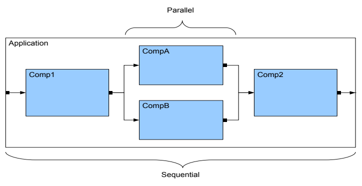


Figure 1: Sequential and Parallel Processes

In either case, interfaces can be very useful to determine whether the communication between components is carried out in a sequential or parallel matter. Imagine a component with a broadcast server interface (or several one to one server interfaces): we can easily assume that the components at the client side of those interfaces can be run in a parallel matter. On the other hand, a component which has only one server interface, can only run in a sequential matter with the component on the client interface side (see Figure: 1). Sequential specification looks like: $Comp1 \Rightarrow \mathbf{A}\diamondsuit Comp2$ while Parallel specification looks like: $Comp1 \Rightarrow \mathbf{A}\bigcirc(CompA \wedge CompB)$.
When in a sequential process is easy to understand that component will be started sequentially, in a parallel process, there is no real certainty - component might be all started at the next step, or first one and then the others, or perhaps none.

State of resources. When considering resources, we are able to formally specify the environment thanks to information provided in the GCM deployment file as well as other metadata information gathered at development time through a development interface. Furthermore the current state of each resource can be monitored at runtime giving us a complete picture of the resources at every given moment in time and any components that might be deployed on or requesting the use of the resource. External resources are defined as: $Comp1 \Rightarrow \mathbf{A}\diamondsuit Res1$. Deployment resources are defines as: $Node1 \Rightarrow Res1$ and at runtime we can have definitions like: $Res1 \Rightarrow \mathbf{A}\square(Comp1 \wedge Com2)$.

State of components. While the states of components could have a wide spectrum of definition points (such as *initialized, started, suspended, terminated, . . .* for the moment we can only consider the ones defined in the GCM - i.e. *started* and *stopped*. In a way this simplifies further the formalism by representing the specification as: $Comp1$ for a started component, and: $\neg Comp2$ for a stopped one.

Conclusions. The formal specification procedure introduced in this paper will be used in reconfiguration scenarios to prevent inconsistency while suggesting possible corrections to the system. While we have applied this framework to a GCM system, such procedure could be applied to other systems, giving the deductive reasoning a chance to assist other verification methods such as model checking by filling the gaps in those areas where these other well established methods cannot be used.

References

T. Barros, L. Henrio, and E. Madelaine. Verification of distributed hierarchical components. In *In Proc. of the International Workshop on Formal Aspects of Component Software (FACS'05)*, volume Electronic Notes in Theor. Computer Sci. 160, pages 41–55, 2005.

A. Basso, A. Bolotov, and V. Getov. Automata-based formal specification of stateful systems. In *In Proc. of Automated Reasoning Workshop*, 2008a.

A. Basso, A. Bolotov, and V. Getov. *Behavioural Model of Component-based Grid Environments.*, volume From Grids To Service and Pervasive Computing, pages 19–30. Springer, 2008b.

A. Basukoski, V. Getov, J. Thiyagalingam, and S. Isaiadis. *Component-based Development Environment for Grid Systems: Design and Implementation.*, volume In: Making Grids Work, pages 119–128. Springer, 2008.

F. Baude, D. Caromel, C. Dalmasso, M. Danelutto, V. Getov, L. Henrio, and C. Pérez. *GCM: A Grid Extension to Fractal for Autonomous Distributed Components.*, volume Annals of Telecommunications, vol. 64(1-2), pages 5–24. Springer, 2009.

A. Bolotov and M. Fisher. A clausal resolution method for ctl branching time temporal logic. *Journal of Experimental and Theoretical Artificial Intelligence.*, 11:77–93, 1999.

A. Bolotov, C. Dixon, and M. Fisher. On the relationship between normal form and w-automata. *Journal of Logic and Computation, Oxford University Press.*, Volume 12(Issue 4):561–581, August 2002.

L. Zhang, U. Hustadt, and C. Dixon. First-order resolution for ctl. Technical Report ULCS-08-010, Department of Computer Science, University of Liverpool, 2008.

Natural Deduction Calculus for Quantified Propositional Linear-time Temporal Logic (QPTL)

Alexander Bolotov[⋆]
[⋆]School of Computer Science
University of Westminster
Watford Road, Harrow HA1 3TP, UK
`A.Bolotov@wmin.ac.uk`

Oleg Grigoriev[†]
[†]Department of Logic, Faculty of Philosophy
Moscow State University, Moscow, 119899, Russia.
`grig@philos.msu.ru`

Abstract

We present a natural deduction calculus for the quantified propositional linear-time temporal logic (QPTL) and prove its correctness. The system extends previous natural deduction constructions for the propositional linear-time temporal logic. These developments open the prospect to adapt search procedures developed for the earlier natural deduction systems and to apply the new system as an automatic reasoning tool in a variety of applications capturing more sophisticated specifications due to the expressiveness of QPTL.

1 Introduction

In this paper we continue our investigation of natural deduction framework for non-classical setting, this time tackling propositional linear-time temporal logic extended with propositional quantification [Sistla (1983)]. We follow the notation adopted in [French and Reynolds (2002)] calling this logic QPTL. Our construction naturally extends previously defined system for PLTL [Bolotov et al. (2006)] by new rules to capture the propositional quantification in the setting of linear time.

While the propositional quantification does not add any expressiveness to the classical logic QPTL is more expressive than PLTL presenting the same potential of expressiveness as linear-time μ-calculus (linear-time propositional temporal fixpoint logic) [Kaivola (1997)], ETL (propositional linear-time temporal logic extended with automata constraints) [Wolper (1981)] and S1S (second order logic of one successor) [Kaivola (1997)] so that each of these formalisms is as expressive as Buchi Automata [Büchi (1962)]. A well-known example distinguishing the expressiveness of these formalisms comparing to PLTL is their ability to "count", for example, to express that some property occurs at every even moment of time [Wolper (1981)]. Nevertheless, each of these logics uses its own specific syntax and it makes sense to consider how easy these logics can be used in specification. We believe that in this list QPTL indeed occupies a special place. For example, ETL and linear time $\mu - calculus$ formulae are very difficult for understanding. It is known, in particular, that formulae with nested fixpoints very quickly become incomprehensive while automata constraints added to the logic are far too complex to apprehend intuitively.

The language of QPTL uses the a set, $Prop$, of atomic propositions: $p, q, r, \ldots, p_1, q_1, r_1, \ldots, p_n, q_n, r_n, \ldots$; Boolean operations; temporal operators: $\Box$ – 'always in the future'; $\Diamond$ – 'at sometime in the future'; $\bigcirc$ – 'at the next moment in time'; and propositional quantifiers $\forall$ ('for all') and $\exists$ ('there exists').

The set of *well-formed formulae* of QPTL, *wff*$_{QPTL}$ is defined as follows.

1. If A is *wff*$_{PLTL}$, then A is *wff*$_{QPTL}$.
2. If A is in *wff*$_{QPTL}$ and α is in Prop then $\exists \alpha A$ are in *wff*$_{QPTL}$.

Note that $\forall x A = \neg\exists\neg A$ an that the $\mathcal{U}$ operation is expressible in QPTL Kaivola (1997). For the semantics of QPTL we utilise the notation of Fisher et al. (2001): it is discrete, linear sequence of states

$$\sigma = s_0, s_1, s_2, \ldots$$

which is isomorphic to the natural numbers, $\mathcal{N}$, and where each state, $s_i,\ 0 \leq i$, consists of the propositions that are true in it at the i-th moment of time. Let σ' be an x-variant of σ. For the propositional quantifiers we evaluate a formula A in σ at the moment $i \in N$ as follows $\langle \sigma, i \rangle \models \exists x A \Leftrightarrow \langle \sigma', i \rangle \models A$, for some σ'. A well-formed formula, A, is satisfiable if, and only if, it is satisfiable in some model, and is valid if satisfied in every possible model,

As QPTL combines the propositional linear time logic with propositional quantification, we work, on the one hand, in our old framework of natural deduction calculi originally developed by Gentzen [Gentzen (1969)] and Jaskowski

[Jaskowski (1967)] and improved by Fitch [Fitch (1952)] and Quine [Quine (1950)]. On the other hand, we build on our previous ND constructions for the logic PLTL [Bolotov et al. (2006)] and first order logic [Bolotov et al. (2005)]. Namely, the rules for the linear-time framework are adopted from the former paper while the ideology for the rules for the propositional quantifiers are taken from the latter. The new rules, for the propositional quantifiers, allow us to decompose formulae eliminating either the $\forall$ (freely) or $\exists$ (with some obvious restrictions) propositional quantifier from the formula or, on the contrary, to synthesise formulae introducing these quantifiers: $\forall$ (with some restrictions) or $\exists$ (without restrictions). Simple example in the setting of classical logic would be to derive a valid formula $\exists x.p \Rightarrow (x \Rightarrow p)$ from $p \Rightarrow (q \Rightarrow p)$. Another, perhaps not obvious, example of a valid formula is $\exists x.x$. Note that in the absence of the temporal operations formula $\exists x.A = A(x/\top) \vee A(x/\bot)$, i.e. in the classical setting propositional quantification is expressible in classical logic. An interesting example of a valid formula with temporal operators would be $\exists x.x \wedge \bigcirc \Box \neg x$ which can be interpreted as 'now will never happen again' [French (2003)]. Obviously, this formula is valid only in the semantics without repeating states.

To capture these and other specific QPTL properties, such as quantified induction, in our construction we introduce a range of new rules, such as for example, a 0-premise rule $\vdash \exists x.x$. We show that our ND rules are sound and that every theorem of the axiomatics for QPTL from French and Reynolds (2002) is a theorem in our system thus providing a completeness argument.

References

A. Bolotov, V. Bocharov, A. Gorchakov, and V. Shangin. Automated first order natural deduction. In *Proceedings of IICAI*, pages 1292–1311, 2005.

A. Bolotov, A. Basukoski, O. Grigoriev, and V. Shangin. Natural deduction calculus for linear-time temporal logic. In *Joint European Conference on Artificial Intelligence (JELIA-2006)*, pages 56–68, 2006.

J. R. Büchi. On a decision method in restricted second-order arithmetics. In *Proc.of International Congress of Logic, Methodology and Philosophy of Science*, pages 1–12. Stanford University Press, 1962.

M. Fisher, C. Dixon, and M. Peim. Clausal temporal resolution. *ACM Transactions on Computational Logic (TOCL)*, 1 (2):12–56, 2001.

F. Fitch. *Symbolic Logic.* NY: Roland Press, 1952.

Tim French. Quantified propositional temporal logic with repeating states. In *TIME 2003*, pages 155–165, 2003.

Tim French and Mark Reynolds. A sound and complete proof system for QPTL. In *Advances in Modal Logic*, pages 127–148, 2002.

G. Gentzen. Investigation into logical deduction. In *The Collected Papers of Gerhard Gentzen*, pages 68–131. Amsterdam: North-Holland, 1969.

S. Jaskowski. On the rules of suppositions in formal logic. In *Polish Logic 1920-1939*, pages 232–258. Oxford Univ. Press, 1967.

R. Kaivola. *Using Automata to Characterise Fixed Point Temporal Logics.* PhD thesis, University of Edinburgh, 1997.

W. Quine. On natural deduction. *Journal of Symbolic Logic*, 15:93–102, 1950.

A. Sistla. *Theoretical issues in the Desing and Verification of Distibuted Systems.* PhD thesis, Harvard University, 1983.

P. Wolper. Temporal logic can be more expressive. In *Proceedings of the 22nd IEEE Symposium on Foundations of Computer Science*, pages 340–348, Nashville, October 1981.

Adapting Piecewise Fertilisation to Reason about Hypotheses

Louise A. Dennis[⋆]
[⋆]Department of Computer Science
University of Liverpool
L.A.Dennis@liverpool.ac.uk

Lucas Dixon[†]
[†]School of Informatics
University of Edinburgh
ldixon@inf.ed.ac.uk

1 Introduction

Fertilisation is the point in an inductive proof when the induction hypothesis is used to discharge (or rewrite) the induction conclusion.

Piecewise Fertilisation was developed by Armando et al. (1999) for handling situations where logical connectives appear in the theorem statement which make the fertilisation process less straightforward.

Of particular interest is the situation where an implication appears in the theorem statement. Typically a theorem of the form

$$\forall x.P(x) \Rightarrow Q(x), \tag{1}$$

produces (in the case where x is a natural number) a step case of the form

$$(P(n) \Rightarrow Q(n)) \wedge P(s(n)) \vdash Q(s(n)), \tag{2}$$

to discharge this piecewise fertilisation breaks this into two sub-problems:

$$P(s(n)) \Rightarrow P(n), \tag{3}$$

and

$$Q(n) \Rightarrow Q(s(n)). \tag{4}$$

This abstract describes preliminary work to take the idea of piecewise fertilisation and combine it with rippling techniques to provide support for reasoning within the hypotheses of inductive proofs.

2 An Example: Counting an element in a List

Let us consider the theorem

$$\forall x, l.x \notin l \Rightarrow cl(x, l) = 0. \tag{5}$$

Where $\in$ is list membership and cl is defined as

$$\begin{aligned} cl(a, []) &= 0, && (6)\\ cl(a, h :: t) &= \text{if } (a = h) \text{ then } s(cl(a, t)) \text{ else } cl(a, t). && (7) \end{aligned}$$

A proof-planning style induction proof on this theorem proceeds by induction on l. The base case is easily discharged, leaving the step case:

$$(\forall z.z \notin t \Rightarrow cl(z, t) = 0) \wedge y \notin (h :: t) \vdash cl(y, h :: t) = 0, \tag{8}$$

which ripples/rewrites to

$$(\forall z.z \notin t \Rightarrow cl(z, t) = 0) \wedge y \notin (h :: t) \vdash \text{ if } (y = h) \text{ then } s(cl(y, t)) \text{ else } cl(y, t) = 0, \tag{9}$$

and then case splits to

$$(\forall z.z \notin t \Rightarrow cl(z, t) = 0) \wedge y \notin (h :: t) \wedge y \neq h \vdash cl(y, t) = 0, \tag{10}$$

$$(\forall z.z \notin t \Rightarrow cl(z, t) = 0) \wedge y \notin (h :: t) \wedge y = h \vdash s(cl(y, t)) = 0. \tag{11}$$

The second of these has a contradiction in the hypothesis (which can be detected by counter-example finding and discharged easily)

In piecewise fertilisation the case-split of (2) into (3) and (4) is done by identifying an embedding between the antecedant of the induction hypothesis and another hypothesis of the step case. In this case $y \notin h :: t$ embeds into $z \notin t$ (taking into account that z is universally quantified). Instead of forming two sub-problems, as in piecewise fertilisation, we suggest annotating the hypothesis, as for rippling, and rewriting directly.

$$(\forall z. z \notin t \Rightarrow cl(z,t) = 0) \land y \notin \boxed{(h :: \underline{t})} \land y \neq h \vdash cl(y,t) = 0 \tag{12}$$

$$(\forall z. z \notin t \Rightarrow cl(z,t) = 0) \land \boxed{y \neq h \land \underline{y \notin t}} \land y \neq h \vdash cl(y,t) = 0 \tag{13}$$

Which then lets us infer that $cl(y,t) = 0$ and prove the step case.

3 Current Status

We have most of this process implemented in the IsaPlanner proof planning system (Dixon and Fleuriot, 2003). We are successfully able to perform rippling in the hypothesis of the induction but are not, currently, able to complete the final steps because of some short comings in the implementation of simplification in IsaPlanner – we are working on this.

We believe that this technique will prove useful for simplifying hypotheses in both fully automated and interactive proofs.

Acknowledgements

This work was funded by EPSRC Platform Grant EP/E005713/1.

References

A. Armando, A. Smaill, and I. Green. Automatic synthesis of recursive programs: The proof-planning paradigm. *Automated Software Engineering*, 6(4):329–256, 1999.

L. Dixon and J. D. Fleuriot. IsaPlanner: A prototype proof planner in Isabelle. In *19th International Conference on Automated Deduction (CADE'2003)*, volume 2741 of *LNCS*, 2003.

Combining Generalisation Into Instantiation Based Reasoning In EPR

Shasha Feng*

*The University of Manchester
Manchester, M13 9PL
fengs@cs.man.ac.uk

Abstract

Effectively propositional logic (EPR for short) is a fragment of first-order logic which can be effectively translated into propositional logic. Much attention has been drawn to reasoning in EPR because of its ability to represent real life applications such as bounded model checking and planning. In this paper we summarize some successul calculi in EPR reasoning and propose combining generalisation into instantiation-based reasoning.

1 Introduction

Effectively propositional logic (EPR for short) is a fragment of first-order logic whose formulae are those in the Bernays-Schønfinkel class. Much attention has been drawn to reasoning in EPR since several real life applications such as bounded model checking and planning can be naturally and succinctly encoded as EPR formulae.

When skolemised, EPR formulae contains no function symbols and thus have a finite Herbrand Universe, which allows one to translate them to propositional logic using *grounding*: substitutions of constants for variables of the formulae. Grounding and the subsequent use of SAT solvers remains one of the most successful approaches to checking the satisfiability of EPR formulae.

In this paper we summarize some successul calculi in EPR reasoning and propose combining generalisation into instantiation-based reasoning.

2 Existing Methods

2.1 Competent methods

Pérez and Voronkov (2008) analysed reasoning in EPR from the perspective of proof length. They show that propositional resolution for EPR may have exponentially longer refutations than resolution for EPR. It suggests that methods based on ground instantiation may be weaker than non-ground methods. Though their analysis based on resolution, which has so far been found not competitive on EPR formulae by the recent CASC competition, the conclusion finds some support from other calculi which are competent in EPR reasoning, as explained in the following two paragraph.

Model Evolution, proposed by Baumgartner and Tinelli (2003), is seen as a lift of DPLL to first-order logic. The calculus tries to build a model for input formulae. In the model, the presence of a predicate means that all of instances are satisfied, except those which are also instances of the predicate's instance, whose complement also appears in the model. For example, if one model only contains $P(x)$ and $\neg P(c)$, then it stands for a partial model of $\{P(a), P(b), P(d), \ldots\}$.

Instantiation based theorem proving, proposed by Ganzinger and Korovin (2003), is a cycle of generating instances and SAT checking, which instances generation is guided by resolution. SAT checking is performed on an abstraction of current problem. I. e. all the variables occurring are replace by $\perp$ Korovin (2008). So, in SAT checking, $P(x)$ and $P(a)$, abstracted to $P(\perp)$ and $P(a)$, are treated as different propositional variables. In some way, the $\perp$ in $P(\perp)$ can be seen as any constant in the domain. Thus, instantiation based theorem proving makes itself a calculus on the level of EPR instead of propositional logic.

2.2 Generalisation

Recently, Pérez and Voronkov (2008) introduces a generalisation rule and show that resolution for EPR may have exponentially longer proofs than resolution with generalisation. Suppose that in an EPR problem, the constants domain is $\{c_1, c_2, \ldots, c_8\}$, and we have $A[c_3], A[c_4], A[c_5], \ldots, A[c_8]$, where $A[x]$ is a quantifier-free formula with a free variable x. To generalise all the ground cases of $A[c_3], A[c_4], A[c_5], \ldots, A[c_8]$, it would be much convenient to add $\forall x A[x]$ to the formulae set. However, $\forall x A[x]$ only holds when $x \in \{c_3, c_4, \ldots, c_8\}$. So, we can add a constrained formula $\forall x A[x].C$, C represents the constraints about x. Resolution extends to constrained formulae naturally, with the resolvent's constraint being the intersection of the constraints from the two input formulae.

3 Combining Generalisation Into Instantiation Based Reasoning

We propose to combine generalisation rule into instantiation based reasoning. Constrained formulae generated by generalisation rule, together with instances generated by resolution rule, are added to the formulae set, which will be fed to a SAT solver. The constraints will be discarded from constrained formulae after some constraints checking. We illustrate by the following example.

Example 1 Suppose a formulae set is $\{P(x) \vee Q(x), \neg P(a), \neg P(b), \neg Q(b), R(x) \vee \neg Q(x), \neg R(b)\}$. Generalisation rule produces the following constrained formulae. $\neg P(x).x \in \{a, b\}, \neg Q(x).x \in \{b\}, \neg R(x).x \in \{b\}$. These constraints are satisfiable, as $x \in \{b\}$. Thus, we can add $\neg P(x), \neg Q(x), \neg R(x)$ to existing formulae set, waiting for a SAT solver, which will return unsatisfiable.

There are several unsolved problems in our method. One is what should be added when the constaints are unsatisfiable? One solution is to add only those constrained formulae whose constraints, put together, are satisfiable. Heuristic methods may be needed when chosing a subset of constrained formulae. Another solution is kind of splitting according to constants domain. For example, if the constraints formulae are $P(x).x \in S_1, Q(x).x \in S_2$, where $S_1 \cap S_2 = \phi$. We can get two duplications of existing formulae set, with all predicates subscribed by 1 and 2 respectively. Then replace existing formulae set with the union of duplications, and discard those $U_1(c)$ where $c \notin S_1$ and $V_2(d)$ where $d \in S_1$. For the constraints formulae $P(x).x \in S_1, Q(x).x \in S_2$, we add $P_1(x), Q_2(x)$.

Another problem is about lemma generating. In Ganzinger and Korovin (2003) method, when the problem is satisfiable, the SAT solver may return some clauses it derived, which can be fed into the problem as lemmas. With generalisation rule present, it is difficult to tell whether the returned clauses are universally valid or only on a specific subset.

4 Conclusions

We analyzed several successful calculi in EPR reasoning and proposed to combine generalisation into instantiation based reasoning. Some unsolved problems are described. Comparison with de Moura and Bjørner (2008) is also an interesting direction. Moreover, implementation is necessary to discovor how to balance over generalisation and instances generation on different problems.

References

Peter Baumgartner and Cesare Tinelli. The model evolution calculus. In *CADE*, pages 350–364, 2003.

Leonardo Mendonça de Moura and Nikolaj Bjørner. Deciding effectively propositional logic using dpll and substitution sets. In *IJCAR*, pages 410–425, 2008.

Harald Ganzinger and Konstantin Korovin. New directions in instantiation-based theorem proving. In *LICS*, pages 55–64, 2003.

Konstantin Korovin. iprover - an instantiation-based theorem prover for first-order logic (system description). In *IJCAR*, pages 292–298, 2008.

Juan Antonio Navarro Pérez and Andrei Voronkov. Proof systems for effectively propositional logic. In *IJCAR*, pages 426–440, 2008.

Computing Explanations for Entailments in Description Logic Based Ontologies

Matthew Horridge[⋆] Bijan Parsia[⋆] Ulrike Sattler[⋆]
[⋆]The University of Manchester
Oxford Road Manchester, M13 9PL
{matthew.horridge|bparsia|sattler}@cs.man.ac.uk

1 Introduction

Trying to track down and understand the causes of entailments in Description Logic based ontologies[1] can be a wretched and error prone task. Without some kind of tools support many users of ontology editing tools, such as Protégé-4, find that it is impossible to determine the reasons for unsatisfiable classes or other undesirable entailments that can arise during the process of constructing an ontology. Indeed, users of such tools have been seen to switch tool purely for the benefits of explanation (Kalyanpur et al. (2007)). In recent years, there has been a significant amount of interest in generating explanations for entailments in ontologies. Generally speaking the focus has moved from finding and generating explanations that correspond closely with a particular proof technique, such as natural deduction, to finding and generating explanations whose sub-structure is at the level of asserted axioms. This has given rise to explanations that, in their most general form are known as *Justifications* (Kalyanpur et al. (2007)). A justification for an entailment in an ontology is a minimal subset of the ontology that is sufficient for the entailment to hold. More precisely, given an ontology $\mathcal{O}$ such that $\mathcal{O} \models \eta$, $\mathcal{J}$ is justification for η with respect to $\mathcal{O}$ if $\mathcal{J} \subseteq \mathcal{O}$, $\mathcal{J} \models \eta$, and for all $\mathcal{J}' \subsetneq \mathcal{J}$, $\mathcal{J}' \not\models \eta$. Note that there may be multiple, potentially overlapping, justifications for a given entailment.

2 Computing Justifications

Methods of computing justifications are broadly categorised into *glass-box* methods and *black-box* methods. While both types of method depend upon reasoning, a glass-box implementation is specific to a given reasoner and therefore reasoning technique, while a black-box method does not depend on a specific reasoner or reasoning technique.

Glass-box methods usually require thorough and non-trivial modifications to reasoner internals. The tableaux based reasoner Pellet was augmented with tableaux *tracing* whereby, as the tableaux is expanded, Pellet tracks the axioms that are used in the expansion. Computing *all* justifications for an entailment using this technique would require saturation of the completion graph, and would require many optimisations to be rolled back. Therefore a hybrid approach, combining glass-box methods with black-box methods and model diagnosis techniques is used to compute all justifications.

Black-box methods are much easier to implement than glass-box methods as they just require a reasoner[2] that can perform entailment testing, and some (goal directed) procedure to examine subsets of an ontology in order to compute all justifications. Black-box implementations typically use some optimised "expand/shrink" strategy. For example, the signature of an entailment is used as an input to a selection function that is repeatedly used to select larger and larger subsets of the ontology until the entailment in question holds in the subset, at which point axioms in the subset that are not relevant for the entailment are gradually pruned away.

The work presented in Kalyanpur et al. (2007) provides descriptions of a Pellet based glass-box implementation and black-box implementation that uses a simple expand/shrink strategy. Empirical investigation showed that it is practical to compute justifications for a range of ontologies varying in size and expressivity. More recently, we sampled entailments from over twenty published ontologies ranging from $\mathcal{ALC}$ to $\mathcal{SHOIQ}$, and from tens of axioms to tens of thousands of axioms. With the necessary optimisations to the black-box implementation, it was found that black-box justification finding can perform equally well, if not better, than glass-box justification finding—the tracing technique used in the glass box justification finding does impose a slight overhead on satisfiability testing. Given that modifying an existing reasoner to support glass-box justification finding is highly non-trivial, and binds the implementation to a specific reasoner, we recommend the use and continued optimisation of black-box methods.

[1]Description Logics being decidable fragments of First Order Logic
[2]A reasoner being an implementation of a decision procedure for satisfiability testing etc.

3 Fine-grained Justifications: Laconic and Precise Justifications

Given a, potentially very large, ontology, justifications pick out the handfuls of axioms that are responsible for the entailment. This is hugely useful since it allows a user to focus on what could be a very small part of the ontology. Thus, when trying to understand the reasons for an entailment, the user can devote their attention to examining just a few axioms compared to examining the whole ontology. However, it is frequently the case that not all *parts* of axioms are required for an entailment to hold. This has given rise to the notion of *fine-grained* justifications, which are justifications whose axioms do not contain any superfluous parts.

Numerous groups of researchers have identified fine-grained justifications as being important. However, the notion of fine-grained justifications was only recently formalised in Horridge et al. (2008), which split fined-grained justifications into laconic justifications and precise justifications. Using the well known structural transformation given in Plaisted and Greenbaum (1986) to identify the "parts" of axioms, laconic justifications are justifications all of whose axioms do not contain any superfluous parts and, more over, all of whose parts of axioms are as weak as possible. Precise justifications are laconic justifications whose parts of axioms have been transferred into separate axioms. Laconic justifications are geared towards user understanding, while precise justifications are geared towards repair.

The definition of laconic and precise justifications given in Horridge et al. (2008) is given with respect to the deductive closure of an ontology. In practice, laconic and precise justifications are computed with respect to a filter on the deductive closure of an ontology. The filter is used to select justifications that contain axioms that have a strong syntactic resemblance to the axioms in the original ontology. In essence this is essential for usability. The filters used in practice tend to generate from the ontology an expanded set of axioms that contains controlled, stepwise weakenings of the asserted axioms. While this generally results in many more justifications for an entailment, various optimisations can be used to ensure good algorithmic performance on real ontologies. Indeed, results presented in Horridge et al. (2008) show that it is practical to compute laconic justifications using black-box methods for a range of published ontologies, with all laconic justifications for an entailment typically being computed in tens of seconds on a standard laptop computer.

4 Lemmatising Justifications

We have observed that some justifications can be very difficult for people to understand. In a user study we found that approximately 20% of justifications that were generated from over 100 entailments taken from published ontologies could not be understood by a wide range of people. Justifications essentially gather the premises of a proof together and present them to a user. The user is left to fill in the gaps and work out how the interplay between axioms in order to figure out how they result in the conclusion, i.e. the entailment of interest. One of the areas that we are currently exploring is the lemmatisation of justifications so as to make helpful intermediate inference steps explicit. A justification can be lemmatised by replacing one or more axioms with summarising/bridging axiom. The result is a simpler and easier to understand justification. The replacement axiom is essentially a lemma, with the axioms that were replaced being a justification for this lemma. There can be multiple lemmatisations per justification, and the justification for lemmas can themselves be lemmatised. This results in a proof DAG that can be used as an input to a presentation service for use in end user tools such as Protégé-4. It should be noted that the notion of how easy or difficult a justification is to understand is governed by a complexity model, which is used to predict whether or not a justification ought to be lemmatised. We developed a complexity model based on the results from the aforementioned user study and used this to compute lemmatised justifications for entailments from published ontologies. It was found that it was feasible to lemmatise justifications, with many lemmatisations being computed in the order of tens of seconds.

References

Matthew Horridge, Bijan Parsia, and Ulrike Sattler. Laconic and precise justifications in owl. In *ISWC 08 The International Semantic Web Conference 2008, Karlsruhe, Germany*, 2008.

Aditya Kalyanpur, Bijan Parsia, Matthew Horridge, and Evren Sirin. Finding all justifications of owl dl entailments. In *ISWC 07 The International Semantic Web Conference 2007, Busan, Korea*, 2007.

David A. Plaisted and Steven Greenbaum. A structure-preserving clause form translation. *Journal of Symbolic Computation*, 1986.

“Exactly One” Epistemic Logic

Boris Konev, Clare Dixon and Michael Fisher[⋆]

[⋆] Department of Computer Science, University of Liverpool, Liverpool, L69 3BZ, UK

{Konev,CLDixon,MFisher}@liverpool.ac.uk

1 Introduction

Epistemic logic is a modal logic of knowledge (see for example (Fagin et al., 1995)) in which the knowledge of agents, players or processes can be represented. When combined with temporal logics, to represent dynamic aspects, such logics can be used to represent and reason about systems where evolving knowledge plays a key part such as agent based systems, knowledge games, security protocols etc. Many such systems contain subsets of the set of propositions needed to represent the system where exactly one member of each subset holds at any moment. An example of an “exactly one” set in a game playing environment, where all the cards are dealt out to players a, b, c or d and ten_spades_i represents that player i holds the ten of spades card, is the set $\{ten\_spades_a, ten\_spades_b, ten\_spades_c, ten\_spades_d\}$ i.e. the ten of spades card must be held by exactly one player.

Recently we have been investigating, mechanising, and applying, *temporal* logics with additional constraints of the “exactly one” type considered above (Dixon et al., 2007a,b, 2008). In our work, each logic is parametrised by a set of propositions (or, predicates in the first-order case) where exactly one of these propositions is satisfied at any temporal state. We have shown that, if problems can be described in such a logical framework, then not only is the description more succinct, but the decision procedure for the logic is simpler (reducing certain aspects of the decision procedure from *exponential* to *polynomial*).

In this extended abstract we investigate theorem proving for epistemic modal logics allowing *“exactly one”* sets. In particular, we define a tableau procedure for this logic, which incorporates a DPLL (Davis et al., 1962) like mechanism to satisfy the “exactly one” sets and show that this reduces the number of states we must construct. The full details of this work can be found in (Konev et al., 2009).

2 Example

To illustrate the approach we focus on a simple card game from (van Ditmarsch et al., 2005). In this simple game there are three different cards; a heart, a spade and a club. In the most basic scenario, one card is dealt to one player, a further card is placed face down on the table and the final card is returned (face down) to the card holder.

Following (van Ditmarsch et al., 2005) we use simple propositions to represent the position of the cards. So, if $spades_w$ is true, then Wiebe holds a spade, if $clubs_t$ is true, then the clubs card is on the table, if $hearts_h$ is true, then the hearts card is in the holder, etc. Similarly, $K_w spades_w$ means that Wiebe *knows* he holds a spade. And so on.

We can identify six “exactly one” sets, firstly

$$\{spades_w, spades_h, spades_t\}$$

denoting the spades cards can be in exactly one place at any moment, and the same for $clubs_i$ and $hearts_i$ for each of $i = w, h, t$. Further,

$$\{spades_i, hearts_i, clubs_i\}$$

for $i = w, h, t$ denotes for $i = w$ that Wiebe can only hold exactly one card or for $i = h$ exactly one card can be in the holder or $i = t$ exactly one card can be on the table.

3 SX5$_n$— “Exactly One” sets in Epistemic Logic

The logic we consider is called “**SX5**$_n$”. The main novelty in **SX5**$_n$ is that it is parametrised by “exactly-one”-sets $\mathcal{P}_1$, $\mathcal{P}_2$,. . . , denoted **SX5**$_n(\mathcal{P}_1, \mathcal{P}_2, \ldots)$, which are constructed under the restrictions that *exactly* one proposition from every set $\mathcal{P}_i$ is true in any state. Additionally there may be a set A of normal (unconstrained) propositions.

Assuming a set of agents where $Ag = \{1, \ldots n\}$, formulae are constructed from a set $\textsc{Prop} = \{p, q, r, \ldots\}$ of *atomic propositions*, using the usual Boolean connectives: $\neg$ (not), $\vee$ (or), $\wedge$ (and) and $\Rightarrow$ (implies) plus K_i, for $i \in Ag$ (agent i knows).

A *model structure*, M, for $\mathbf{SX5}_n$ is a structure $M = \langle S, R_1, \ldots, R_n, \pi \rangle$, where: S is a set of states; $R_i \subseteq S \times S$, for all $i \in Ag$, is the agent accessibility relation where R_i is an equivalence relation; and $\pi : S \times \mathcal{P} \rightarrow \{T, F\}$ is a valuation.

As usual, we define the semantics of the language via the satisfaction relation '$\models$'. This relation holds between pairs of the form $\langle M, s\rangle$ (where M is a model structure and $s \in S$), and $\mathbf{S5}_n$-formulae. The rules defining the satisfaction relation are given below where the semantics of the Boolean operators is as usual.

$$\begin{array}{lll} \langle M, s\rangle \models q & \text{iff} & \pi(s, q) = \textbf{true} \text{ (where } q \in \textsc{Prop}) \\ \langle M, s\rangle \models K_i\phi & \text{iff} & \forall s' \in S' \text{ if } (s, s') \in R_i \text{ then } \langle M, s'\rangle \models \phi \end{array}$$

If there is a model structure M and state s such that $\langle M, s\rangle \models \varphi$ then φ is said to be *satisfiable*. If $\langle M, s\rangle \models \varphi$ for all states s and all states s then φ is said to be valid. The set of modal relations (for each agent i) are assumed to be equivalence relations.

4 Results

In (Konev et al., 2009) we present a tableau algorithm for $\mathbf{SX5}_n$. Consider an $\mathbf{SX5}_n$ formula φ to be shown satisfiable. The algorithm constructs sets of *extended assignments* of propositions and modal subformulae i.e. a mapping to true or false, that satisfy both the exactly one sets and φ. However, rather than using the usual alpha and beta rules (see for example the modal tableau in (Halpern and Moses, 1992; Wooldridge et al., 1998)) these are constructed using a DPLL-based expansion (Davis et al., 1962). Next the algorithm attempts to satisfy formulae of the form $\neg K_i\psi$ made true in such an extended assignment by constructing R_i successors which are themselves extended assignments which must satisfy particular subformulae (and the exactly one sets).

We show that the tableau algorithm is sound and complete and that given φ an $\mathbf{SX5}_n$ formula then the tableau algorithm runs in time polynomial in

$$\left(k \times |\mathcal{P}_1| \times \ldots \times |\mathcal{P}_n| \times 2^{|A|+k}\right)$$

where $|\mathcal{P}_i|$ is the size of the set $\mathcal{P}_i$ of exactly one propositions, $|A|$ is the size of the set A of non-constrained propositions, and k is the number of K_i operators in φ.

Future work involves applying this logic to more case studies and extending $\mathbf{SX5}_n$with temporal aspects to target evolving knowledge allowing us to be able to represent and reason about problems from complex domains such as security and planning more efficiently.

References

M. Davis, G. Logemann, and D. Loveland. A Machine Program for Theorem-Proving. *Communications of the ACM*, 5 (7):394–397, 1962.

C. Dixon, M. Fisher, and B. Konev. Temporal Logic with Capacity Constraints. In *Proc. of the 6th International Symposium on Frontiers of Combining Systems (FroCoS)*, pages 163–177. LNAI, 2007a.

C. Dixon, M. Fisher, and B. Konev. Tractable Temporal Reasoning. In *Proc. International Joint Conference on Artificial Intelligence (IJCAI)*, pages 318–23. AAAI Press, 2007b.

C. Dixon, M. Fisher, B. Konev, and A. Lisitsa. Practical First-Order Temporal Reasoning. In *Proceedings of TIME 2008 the Fifteenth International Symposium on Temporal Representation and Reasoning*, Montreal, Canada, 16th-18th July 2008. IEEE Computer Society Press.

R. Fagin, J. Y. Halpern, Y. Moses, and M. Y. Vardi. *Reasoning About Knowledge*. MIT Press, 1995.

J. Y. Halpern and Y. Moses. A guide to completeness and complexity for modal logics of knowledge and belief. *Artificial Intelligence*, 54(3):319–379, April 1992.

B. Konev, C. Dixon, and M. Fisher. Playing Cards with Wiebe [Solving Knowledge Puzzles with "Exactly One" $S5^n$]. In *Proceedings of Wiebefest 2009*, Liverpool, UK, March 2009.

H.P. van Ditmarsch, W. van der Hoek, and B.P. Kooi. Playing cards with Hintikka. An introduction to dynamic epistemic logic. *Australasian Journal of Logic*, 3:108–134, 2005.

M. Wooldridge, C. Dixon, and M. Fisher. A Tableau-Based Proof Method for Temporal Logics of Knowledge and Belief. *Journal of Applied Non-Classical Logics*, 8(3):225–258, 1998.

A Decidable Approach to Real-time System Specification

Savas Konur[⋆]

[⋆]Department of Computer Science, University of Liverpool, Liverpool L69 3BX
konur@liverpool.ac.uk

1 Introduction

Most formal methods employed for the specification and development of distributed systems are either event-based or state-based (For a more detailed account for the concepts 'events' and 'states', please see [2]). For system development both views are important [6]. More generally, in early phases of systems development, event-based methods are more suitable; in contrast, in later phases state-based methods are more suitable [1].

We can, therefore, say that a formal method should cover both event-based and state-based views in order to model the behaviour of real-time systems. Most temporal logics employed for the specification of real-time systems, however, do not support both methods. They are either event-based or state-based. For this reason, finding a temporal logic which covers both views, and which can be used in specifying real-time system properties is an important research subject in this area.

On the other hand, choosing a mathematical model of time has been also a primary concern in this context. A basic way to characterise temporal logics is whether points (instants) or intervals are used to model time. It has been turned out that the interval-based scheme provides us with a richer representation formalism than the point-based scheme. Especially, the notion of interval is necessary to represent continuous processes and to make temporal statements which are based on intervals.

Unfortunately, in the area of interval logics, undecidability is very common, and decidability is very rare. Most interval logics have indeed turned out to be (highly) undecidable. In the literature various methods have been proposed to achieve decidability for interval logics. However, most of the methods, such as translating interval logics into point-based ones, cause some syntactic and semantic restrictions. A major challenge in this area is thus to genuinely identify interval-based decidable logics, that is, logics which are not explicitly translated into point-based logics or other semantic restrictions.

In order to overcome these drawbacks we introduce a decidable event- and state-based interval temporal logic, called TL [2]. The logic TL covers both event- and state-based views. In TL intervals are used as primitive objects of the model by allowing quantification over only interval objects. Unlike many other interval logics, we do not translate the logic TL into a point-based variant, and we therefore try to minimise semantic restrictions. Since we restrict ourselves to genuine intervals, this logic is theoretically more challenging.

Another important feature of TL is that it incorporates the notion of *duration*, denoting the length of a state (or an event), and *accumulation*, denoting the total duration of a state. These notions have been found useful for reasoning about time durations of dynamic systems.

In [2] we prove that TL has the finite model property, and the satisfiability problem is decidable. Its unique 'quasi-guarded' character of the quantification is very important to guarantee the decidability. [2] also provides a tableau based decision procedure to find a complexity bound for satisfiability, showing that this problem can be solved in NEXPTIME.

2 Real-Time System Applications

In order to show the usefullness of the logic TL in specifying properties of real-time systems, in [2] we apply it to well-known mine pump and gas burner examples. The mine pump was first described in [3], where a discrete-time interval logic is used. However, in distributed systems the discrete-time approach cannot specify precisely the behaviour of the computer programs that run on different components of the system with different clocks. Since TL is a continuous-time interval logic, it resolves this problem. The gas burner system was described in [5, 4], where it was modeled by an undecidable interval logic. Since TL is a decidable logic, it removes this limitation. To the best of our knowledge, TL is the first decidable logic which formalizes the gas burner system.

2.1 Water Pump

A mine has water seepage which must be removed by operating a pump. If the water is above danger-mark, an alarm must be sounded, an operating of pump must stop. The mine also has pockets of methane which escape. When there is methane, an alarm must be sounded, and all electrical activity must be shut down to prevent explosion.

The mine pump system was first described in [3], where a logic called *QDDS* is used. QDDS is a discrete-time variant of Duration Calculus, and it can specify the behaviour of computer programs running on a component with a unique clock. However, in this case it can only specify approximately the behaviour of physical components, and if the system is distributed, it cannot precisely specify the behaviour of the computer programs that run on different components of the system with different clocks. Thus, in specifying the hybrid and distributed systems, the continuous time should be used.

In the logic TL, system states are modeled by intervals which have the finite variability meaning that in any finite interval of time there is only a finite number of discontinuous points. Thus, the time in TL is continuous, and we can reason about the behavioural timing properties of computer programs. Therefore, TL is more preferable than QDDC in this perspective.

An example specification is given as follows: The alarm will sound within δ seconds of the water level becoming dangerous. Alarm will persist while the water level is dangerous:

$$\texttt{Alarmcontrol}_1 \equiv [DWater]\, \langle Alarm \rangle^{f}_{>}\, (\ell \leq \delta)$$

2.2 Gas Burner System

A gas burner is a device to generate a flame to heat up products using gas. The gas burner system is a safety-critical system as excessive leaking of gas may lead to an accident. The gas burner system was first specified in [4], where an undecidable logic Duration Calculus [7] used. To the best of our knowledge, so far the gas burner system has not been formalized with a decidable language. So, TL is the first decidable logic to specify the requirements of the gas burner system.

One example specification is given as follows: An ignite failure happens when gas does not ignite within 0.5 s:

$$\texttt{IgniteFail} \equiv [Gas]\, [Ignition]\, (\ell \leq 0.5 \vee ([Flame]^{df}_{>}\, (\ell > 0.5)) \wedge [Flame]^{se} \bot)$$

3 Conclusion and Future Research

In this paper, we briefly discussed a new decidable interval temporal logic, called TL. TL uses intervals as primitive objects of model. TL can specify both event-based and state-based statements. TL also incorporates the notion of duration. To show the usability of TL we applied it to well-know gas-burn and water pump systems. We addressed that using TL removed some limitations with previous specifications. Future research will include implementing the tableau method presented in [2], so that (un)satisfiability of a formula can be returned automatically.

References

[1] E. Kindler and T. Vesper. Estl: A temporal logic for events and states. *ICATPN'98, LNCS 1420*, pages 365–384, 1998.

[2] S. Konur. *An Interval Temporal Logic for Real-time System Specification*. PhD thesis, University of Manchester, 2008.

[3] P. K. Pandya. Specifying and deciding quantified discrete-time duration calculus formulas using dcvalid. *Workshop on Real-Time Tools*, 2001.

[4] A. P. Ravn, H. Rischel, and K. M. Hansen. Specifying and verifying requirements of real-time systems. *IEEE Transactions on Software Engineering*, 19(1), 1993.

[5] E. V. Sorensen, A. P. Ravn, and H. Rischel. Control program for a gas burner: Part 1. informal requirements. Technical Report ID/DTH EVS2, ProCoS Case Study 1, 1990.

[6] W. van der Aalst. Three good reasons for using a petri-net-based workflow management system. *Proceedings of the International Working Conference on Information and Process Integration in Enterprises*, pages 179–201, 1996.

[7] C. Zhou and M. R. Hansen. *Duration Calculus: A Formal Approach to Real-Time Systems*. EATCS Series, Springer, 2004.

Verification of Pervasive Systems[*]

Savas Konur and Michael Fisher[⋆]
[⋆]Department of Computer Science, University of Liverpool, Liverpool L69 3BX
{konur, mfisher}@liverpool.ac.uk

1 Introduction

Pervasive computing refers to a general class of mobile systems that can sense their physical environment, i.e., their context of use, and adapt their behaviour accordingly. Pervasive systems live around us providing services to the inhabitants of a home, the workers of an office or the drivers in a car park. We know that requirements for current and future pervasive systems involve a great diversity of types of services [6], such as multimedia, communication or automation services.

The success of pervasive computing depends crucially on verifying interoperability requirements for the interaction between the devices and their environment. These requirements introduce an important layer of abstraction because they allow modularity in the verification process: it suffices to show that each mobile device or fixed component meets the interoperability requirements, and that the interoperability requirements entail the desired high-level properties, such as "Are visitors properly prevented from accessing confidential information on our wireless network?"

Our focus is on the verification of designs; in particular we focus on the design of basic component behaviours and the protocols which dictate access to them and interaction between them. It is important to note that our intention is not to develop pervasive computing systems as such, but rather to draw motivation from, and test our ideas in, a number of planned and existing pervasive systems.

The project brings together qualitative techniques, including deductive methods, model checking, and abstraction methods, with quantitative techniques, including probabilistic and performance analysis, in order to tackle the problem of verifying pervasive systems. Working together, we aim to make a step change in verification technology by developing novel techniques and learning which techniques are most effective in different contexts. We will be investigating problems which are both new and challenging (hence new techniques and methods will be required), but are still sufficiently close to existing work that our established techniques provide a solid foundation for solving them. The outcomes will directly benefit system designers and, indirectly, end users. They will include techniques applicable to a wide range of application domains, and results and lessons learned from three specific applications including a message forwarding system, a homecare system and RFID system infrastructure.

2 Case Studies

The formal techniques are used to verify the consequent interoperability requirements, and their effectiveness is evaluated through some case studies, which include a message forwarding system - *Scatterbox* [4], a home-care application - *MATCH* [3] and underlying *RF technology* [1]. These systems are briefly described as follows:

The Scatterbox system has been designed to serve as a test bed for context-aware computing in a pervasive computing environment. It provides a content-filtering service to its users by forwarding relevant messages to their mobile phone. The user's context is derived by tracking his/her location and monitoring his/her daily schedule. As messages arrive, Scatterbox forwards them to subscribed users provided the user's available context suggests they are in a situation where they may be interrupted.

MATCH is an event driven home-care system. An event is a requirement that when a given condition is met the system takes appropriate action. Typical examples of events include: "if the front door is left open and nobody is downstairs, then send a message to a stakeholder", "if the lights are left on and nobody is in the house, then turn the lights off", "If Bill is laid down but not in bed, then contact Gill", etc. The MATCH system consists of a set of components and a set of users. The set of components can be split into 4 main categories: *sensors*, *alert triggers*, *tasks* and *outputs*.

Radio frequency (RF) technology is used in many applications for automated indentification of objects or people. A RF system consists of two main components: RF chips and RF readers. RF chips are small microchips supporting wireless data transmission. Data is stored (remotely or not) in the RF chip and can remotely be retrieved by a RF reader. RF chips can be incorporated into products, animals, or people for the purpose of identification and tracking using radio waves.

[*]Work is part of an ongoing project on "Verifying Interoperability Requirements in Pervasive Systems", funded by EPSRC (EP/F033567) and involving collaboration with the universities of Birmingham and Glasgow.

RF readers query these chips for some identifying information about the objects to which chips are attached. Current and emerging applications using this technology include amongst others electronic toll collection, documents such as electronic passports and visas, and RF passes for public transportation.

Our case studies will be drawn from three layers typical within pervasive systems: *individual component*, *protocols between individual components* and *information access*.

The interoperability of components depends on the components exhibiting the right behaviour as individuals. We will identify the relevant properties, and formulate them using logic-based languages (such as temporal and first-order logics and model-checking languages). We aim to develop specific approaches to describing behavioural requirements for trusted pervasive computing components. The protocols between individual components will include application-level protocols specific to the case studies (for example, protocols related to the chemotherapy sensors and mobile equipment), as well as low-level protocols of authentication and data distribution. We will formalise the expected properties of the protocols for later analysis using some process algebra languages. The case studies will identify issues and problems of access control (typically what component or group of components has the right to access a certain resource in a pervasive computing infrastructure) and privacy. We will develop and refine models of access control systems, and techniques for proving properties about them. We will also identify the deficiencies of the languages and their ability to scale up for pervasive computing.

3 Formal Verification

After specifying the requirements of the use cases with a suitable formal language, we will develop suitable technologies to verify these requirements formally. Current state of the art formal methods appear incapable of coping with the verification demand introduced by pervasive systems, because reasoning about such systems requires combinations of multiple dimensions such as quantitative, continuous and stochastic behaviour to be considered, and requires proving properties which are quite subtle to express. In order to tackle the challenge of pervasive system verification, the project aims to leverage the power of established techniques, notably

model checking, a logic-based approach to analysing properties of state-based systems. There has been work (some of which was carried out by the investigators) on extensions such as *parametrised model checking, infinite state model checking* and *probabilistic model checking*, and this will be developed further within the project.

using deduction and abstraction, two closely linked, approaches that can be used either to reduce the verification problem to a scale suitable for model checking, or to tackle the larger problem directly.

process calculi allowing high level descriptions of interaction, communication and synchronisation.

Part of our effort will involve pushing each technique further, but the majority of it will be on *pushing the combination*, i.e. bending and synthesising techniques such as [2, 5] to make them give meaningful results in our case studies.

References

[1] url=http://en.wikipedia.org/wiki/RFID.

[2] R. H. Bordini, L. A. Dennis, B. Farwer, and M. Fisher. Automated verification of multi-agent programs. In *Proc. 23rd IEEE/ACM Int. Conf. Automated Software Engineering (ASE)*, pages 69–78, 2008.

[3] J. S. Clark and M. R. McGee-Lennon. Match: Mobilising advanced technologies for care at home. Poster at *Delivering Healthcare for the 21st Century*, Glasgow, 2007.

[4] S. Knox, A. K. Clear, R. Shannon, L. Coyle, S. Dobson, A. Quigley, and Paddy Nixon. Scatterbox: Mobile message management. *Journal Revue d'Intelligence Artificielle*, 22, 2008.

[5] S. Konur. A decidable temporal logic for events and states. In *Proc. 13th International Symposium on Temporal Representation and Reasoning (TIME)*, pages 36–41. IEEE, 2006.

[6] R. Want, T. Pering, G. Borriello, and K.I. Farkas. Dissapearing hardware. *Pervasive Computing*, 1, 2002.

TSPASS - a Fair Monodic Temporal Logic Prover

Michel Ludwig Ullrich Hustadt

Department of Computer Science, University of Liverpool, Liverpool, UK
{Michel.Ludwig, U.Hustadt}@liverpool.ac.uk

1 Introduction

First-Order Temporal Logic, FOTL, is an extension of classical first-order logic by temporal operators for a discrete linear model of time (isomorphic to $\mathbb{N}$). The set of valid formulae of this logic is not recursively enumerable. However, the set of valid *monodic* formulae is known to be finitely axiomatisable.

A first resolution-based calculus for monodic first-order temporal logic was introduced in Degtyarev et al. (2003). Then, a more machine-oriented version, the fine-grained first-order temporal resolution calculus, was described in Konev et al. (2005). A refinement of fine-grained temporal resolution, the ordered fine-grained temporal resolution with selection calculus, is presented in Hustadt et al. (2005). However, while these calculi represent important steps towards fully automated reasoning in the monodic fragment, they still all have one major drawback: they contain inference rules, reflecting the inductive nature of reasoning in this logic, whose applicability is only semi-decidable as they depend on first-order side conditions which in general are not decidable. This poses a problem for the development of refutation-complete theorem provers based on these calculi.

In more detail, resolution-based calculi for monodic first-order temporal logic require that a given set of monodic temporal formulae is transformed in a satisfiability equivalence preserving way into a clausal form consisting of four types of temporal clauses, namely *initial*, *universal*, *step* and *eventuality* clauses. These clauses are then used in inferences by the rules of the monodic fine-grained temporal resolution calculus. The majority of the rules, the so-called step resolution rules, are based on standard (ordered) first-order resolution between different types of temporal clauses. The remaining inference rules, the ground and the non-ground *eventuality resolution* rule, reflect the induction principle that holds for monodic temporal logic over a flow of time isomorphic to the natural numbers. We present the non-ground eventuality resolution rule; the ground version is similar:

$$\frac{\forall x(\mathcal{A}_1(x) \Rightarrow \bigcirc\mathcal{B}_1(x)) \quad \dots \quad \forall x(\mathcal{A}_n(x) \Rightarrow \bigcirc\mathcal{B}_n(x)) \qquad \Diamond L(x)}{\forall x \bigwedge_{i=1}^{n} \neg\mathcal{A}_i(x)} \; (\Diamond^{\mathcal{U}}_{res}),$$

where $\forall x(\mathcal{A}_i(x) \Rightarrow \bigcirc \mathcal{B}_i(x))$ are complex combinations of step clauses such that for all $i \in \{1, \dots, n\}$, the *loop* side conditions $\forall x(\mathcal{U} \wedge \mathcal{B}_i(x) \Rightarrow \neg L(x))$ and $\forall x(\mathcal{U} \wedge \mathcal{B}_i(x) \Rightarrow \bigvee_{j=1}^{n}(\mathcal{A}_j(x)))$, with $\mathcal{U}$ being the current set of all universal clauses, are both valid. The formula $\bigvee_{j=1}^{n} \mathcal{A}_j(x)$ is called a *loop formula* (for $\Diamond L(x)$).

In the realisation of the eventuality resolution rules a special resolution-based algorithm, called loop search algorithm, is used to find $\forall x(\mathcal{A}_i(x) \Rightarrow \bigcirc \mathcal{B}_i(x))$ for an eventuality $\Diamond L(x)$ satisfying the loop side conditions of the eventuality resolution rule. To do so, the loop search algorithm constructs a sequence of sets containing universal and step clauses, which are then saturated under a subset of the rules of fine-grained step resolution. For each attempt to apply a eventuality resolution rule an instance of the loop search algorithm needs to be executed. As a consequence of the undecidability of the validity of the loop side conditions, executions of the loop search algorithm cannot be guaranteed to terminate. It is therefore possible, for example, that a partial loop formula, which is essential for a refutation, has been discovered by the algorithm but, due to an infinite saturation process, this loop formula is never used in the computation of a consequence of an application of the eventuality resolution rule, and, therefore, is not available to construct the refutation. Moreover, if we try to solve the problem by delaying the application of the eventuality resolution rules until the set of temporal clauses is first saturated under inferences with all other rules, then it might happen that the loop search algorithm will not be executed at all as the saturation under these other inference rules is also not guaranteed to terminate.

To solve this problem we have developed a calculus which only contains inference rules whose applicability is decidable. The basic idea underlying the calculus is that with each clause set which occurs in the sequence of clause sets constructed by an invocation of the loop search algorithm we can associate a unique marker literal which is added to every clause occurring in such a set. In the loop search algorithm we can then work with just one clause set in which clauses are separated by those marker literals, instead of constructing a sequence of clause sets. Furthermore, we do not even have to use different instances of the loop search algorithm for each application of an eventuality resolution rule, but can instead use one global clause set incorporating all of them. This then allows us to fairly perform step resolution inferences

Problem	Clauses Generated		Time		Result
	TeMP	TSPASS	TeMP	TSPASS	
0	19611	5707	0.481s	0.386s	Satisfiable
1	21812	833	0.519s	0.075s	Unsatisfiable
2	-	4827	-	0.372s	Unsatisfiable
12	689	793	0.029s	0.073s	Unsatisfiable
18	32395	5262	0.975s	0.389s	Unsatisfiable

Table 1: Results obtained for the robot specification examples

which essentially drive forward all computations in a parallel way, whereas they were executed sequentially in the original approach. This fair inference architecture has been presented in Ludwig and Hustadt (2008a) and in more detail in Ludwig and Hustadt (2008b).

2 Implementation and Experimental Results

The fair inference architecture has been implemented in the theorem prover TSPASS[1], which is based on the first-order resolution prover SPASS 3.0.

The practical performance of TSPASS has been analysed on different temporal problems and we have found it to be competitive with TeMP. In the monodic first-order temporal logic prover TeMP (Hustadt et al., 2004), which is based on the sequential reasoning approach described above, the first-order prover Vampire is used as a black box which saturates sets containing the first-order translations of temporal clauses.

The experimental setting was as follows: the experiments were run on a PC equipped with an Intel Core 2 6400 CPU and 3 GB of main memory. The execution timeout on each problem was set to 12 minutes. For TeMP the input problems were first transformed into its clausal input form and then TeMP was started on this clausal input without any additional settings. TSPASS was instructed to perform subsumption-based loop search testing.

Table 1 shows the satisfiability status, the number of clauses generated and the median CPU time in seconds over three different runs of TeMP and TSPASS for five representative examples (out of 39) based on the specifications of simple foraging robots and some associated properties. The robot specification results from a novel application of monodic first-order temporal logic in the verification of the behaviour of robot swarms. Further details can be found in (Behdenna et al.). The specification of the robot transition system was given as problem 0, and the remaining problems verify some properties of the transition system. Each of these problems contains at least seven eventualities. TeMP and TSPASS both terminate on the satisfiable problem 0, but TeMP cannot solve the unsatisfiable problem 2 within the given time limit. Additionally, on average TeMP derives more clauses and requires more execution time than TSPASS, except for problem 12. We attribute this observation to the subsumption-based loop search test in TSPASS and to the fact that inferences in TSPASS which have been computed once for a loop search instance do not have to be computed again for further loop search saturations. Further details and more examples can be found in (Ludwig and Hustadt, 2008b).

References

A. Behdenna, C. Dixon, and M. Fisher. Deductive verification of simple foraging robotic behaviours. To appear.

A. Degtyarev, M. Fisher, and B. Konev. Monodic temporal resolution. In *Proc. CADE-19*, volume 2741 of *Lecture Notes in Computer Science*, pages 397–411. Springer, 2003.

U. Hustadt, B. Konev, A. Riazanov, and A. Voronkov. TeMP: A temporal monodic prover. In *In Proc. IJCAR-04*, volume 3097 of *LNAI*, pages 326–330. Springer, 2004.

U. Hustadt, B. Konev, and R. A. Schmidt. Deciding monodic fragments by temporal resolution. In *Proc. CADE-20*, volume 3632 of *LNAI*, pages 204–218. Springer, 2005.

B. Konev, A. Degtyarev, C. Dixon, M. Fisher, and U. Hustadt. Mechanising first-order temporal resolution. *Information and Computation*, 199(1-2):55–86, 2005.

M. Ludwig and U. Hustadt. Fair monodic temporal reasoning. In *Proc. ARW'08*, 2008a.

M. Ludwig and U. Hustadt. Implementing a fair monodic temporal logic prover. 2008b. Submitted.

[1]http://www.csc.liv.ac.uk/˜michel/software/tspass/

Classification of Quasigroup-structures with respect to their Cryptographic Properties

Quratul-ain Mahesar[⋆] and Volker Sorge[†]
[⋆]School of Computer Science, University of Birmingham
Edgbaston, Birmingham
B15 2TT, United Kingdom
{Q.Mahesar|V.Sorge}@cs.bham.ac.uk

1 Introduction

Computationally simple but cryptographically strong ciphers play an important role for efficient Computer Security tasks. It is suggested in (Knapskog, 2008) that there is a need for simple cryptographic primitives to implement security in an environment with end users connected with terminals having limited storage and processing power. Constructing ciphers using the algebraic structures of Quasigroup leads to particular simple yet efficient ciphers. Quasigroups are structures very similar to groups with the primary difference that they are in general not associative. Stream ciphers can be constructed from quasigroups using a permutation based scrambling technique (Maruti, 2006). For security considerations the main goal of the scrambler is to maximize the entropy of the produced output. Depending on the quasigroup used the cryptographic strength of the cipher can vary significantly. In order to find strong ciphers quasigroups have to be generated and the resulting ciphers are tested with respect to standard statistical methods. Quasigroups are then categorised as cryptographically strong or weak according to the outcome of these tests.

In our research we aim to tackle the problem from a different angle. We consider quasigroups that have already been categorised with respect to their cryptographic properties. We then automatically classify these quasigroups with respect to their algebraic properties, with the goal to identify properties common to cryptographically strong quasigroups in order to use them for a goal directed construction of quasigroups for strong ciphers. Our work builds on previous work (Sorge et al., 2008) that was concerned with the generation of classification theorems in quasigroup theory. A bootstrapping algorithm was designed to successively refine a classification of quasigroups of a given finite order by constructing algebraic invariants using machine learning techniques, until a full classification into non-equivalent classes was achieved. The procedure incorporated a set of diverse reasoning techniques, including first order resolution theorem proving, model generation, satisfiability solving and computer algebra methods, and was successfully applied to produce a number of novel classification theorems for loops and quasigroups with respect to isomorphism and isotopism.

2 Quasigroup Ciphers

Following (Pflugfelder, 1990), a quasigroup Q can be defined as a group of elements $(1, 2, 3...n)$ along with a multiplication operator '$*$', such that for every element $x, y \in Q$, there exists a unique solution $z \in Q$ such that the following two conditions hold (1) $x * a = z$, and (2) $y * b = z$, where the elements a, b and z belong to the Quasigroup Q. These conditions ensure that a quasigroup can also be viewed as a Latin square; that is, each element of Q occurs exactly once in each row and each column of the multiplication table defining '$*$'. Conditions (1) and (2) essentially postulate the existence of unique left and right divisors for each element in Q. This gives rise to an explicit definition of left and right division operations:

Let $(Q, \circ)$ be a Quasigroup, then two operations $\backslash$ and $/$ on Q can be defined as:

$(3)\quad x * (x\backslash y) = y \quad$ and $\quad x\backslash(x * y) = y \qquad (4)\quad (y/x) * x = y \quad$ and $\quad (y * x)/x = y$

The following is an example of a Quasigroup Q of order 4 given in terms of multiplication tables for all three operations:

*	1	2	3	4
1	2	3	1	4
2	4	1	3	2
3	3	4	2	1
4	1	2	4	3

\	1	2	3	4
1	3	1	2	4
2	2	4	3	1
3	4	3	1	2
4	1	2	4	3

/	1	2	3	4
1	4	2	1	3
2	1	4	3	2
3	3	1	2	4
4	2	3	4	3

Quasigroup Encryption

We can now define a quasigroup cipher in terms of encryption and decryption function following (Dimitrova and Markovski, 2004). Let $(Q, *, \backslash, /)$ be a Quasigroup and $a_1, a_2, a_3, ..., a_n \in Q$. We define the encryption function E with respect to the key $a \in Q$ as

$$E_a(a_1, a_2, a_3, ..., a_n) = b_1, b_2, b_3, ...b_n$$

where $b_1, b_2, b_3, ..., b_n \in Q$ are computed by (i) $b_1 = a * a_1$, and (ii) $b_i = b_{i-1} * a_i$, for $i = 2, \ldots, n$.

Quasigroup Decryption

The decryption process is similar to the encryption but the left division operation '$\backslash$' is used as operation. The decryption function D is then define as:

$$D_a(a_1, a_2, a_3, ...a_n) = e_1, e_2, e_3...e_n$$

where the original plaintext is computed by (i) $e_1 = a\backslash a_1$, and (ii) $e_i = a_{i-1}\backslash a_i$, for $i = 2, \ldots, n$.

3 Examining Cryptographic Properties

The cryptographic properties of quasigroup ciphers are primarily determined by subjecting the resulting pseudo-random sequences to statistical tests for randomness. In (Markovski et al., 2004) eight bespoke statistical tests are performed by random walk on torus examining the properties of strings obtained from specific quasigroup transformations. This can provide an empirical classification of Quasigroups with respect to their cryptographic hardness. However, exhaustive classification of quasigroups is prohibitive even for small sizes of quasigroups due to the sheer number of different structures to consider. For example, there are over $2 \cdot 10^{30}$ different isomorphism classes of quasigroups of order 10 (McKay et al., 2004). Moreover, (Markovski et al., 2004) shows that quasigroups belonging to the same isomorphism class can behave differently with respect to their cryptopgraphic properties and therefore considering quasigroups up to isomorphism would not be enough. In (Koscielny, 2002) a system for generating quasigroups for cryptographic applications is presented giving a set of procedures implemented in Maple 7. It is also stated that practical ciphers should be constructed using quasigroups of order between 32 and 256. Since the generation of structures of this size is non-trivial, the construction of larger quasigroups is done via composition of smaller ones and cryptographic properties are lifted from the smaller to larger structures. Nevertheless the final cryptographic hardness can only be ensured using randomness test.

The goal of our work is to use these results as a bases on which to start an algebraic classification process, establishing properties that discriminate small quasigroups with good cryptographic properties from those with poor cryptographic behaviour using the automated bootstrapping approach from (Sorge et al., 2008). Once invariants of this nature have been established they have to be examined with respect to their behaviour under compositions of quasigroups. After appropriate relationships between algebraic and cryptographic properties can be established they can subsequently be exploited to aid the modular construction of larger quasigroups for more effective ciphers.

References

V. Dimitrova and J. Markovski. On quasigroup sequence random generator. *Proceedings of the 1st Balkan Conference in Informatics*, pages 393–401, 2004.

S. J. Knapskog. New cryptographic primitives (plenary lecture). *7th Computer Information Systems and Industrial Management applications*, 2008.

C. Koscielny. Generating quasigroups for cryptographic applications. *Int. J. Appl. Math. Comput. Sci.*, 12(4):559–569, 2002.

S. Markovski, D. Gligoroski, and J. Markovski. Classification of quasigroups by random walk on torus. *IJCAR workshop on Computer Supported mathematical Theory Development*, 2004.

Satti Maruti. A quasigroup based cryptographic system. *CoRR*, 2006.

Brendan D. McKay, Alison Meynert, and Wendy Myrvold. Small Latin Squares, Quasigroups and Loops. Preprint, 2004.

Hala O. Pflugfelder. *Quasigroups and Loops: Introduction*, volume 7 of *Sigma Series in Pure Mathematics*. Heldermann Verlag, Berlin, Germany, 1990.

V. Sorge, S. Colton, R. McCasland, and A. Meier. Classification results in quasigroup and loop theory via a combination of automated reasoning tools. *Comment.Math.Univ.Carolinae*, 49(2):319–339, 2008.

Symmetry Reduction of Partially Symmetric Systems

Christopher Power
Department of Computing Science
University Of Glasgow
power@dcs.gla.ac.uk

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

Abstract

Previous research into symmetry reduction techniques have shown them to be successful in combatting the state-space explosion problem. We provide a brief overview of a fully automated technique for the application of symmetry reduction to partially symmetric systems.

1 Introduction

As software becomes more complex the need for development techniques capable of uncovering errors at design time is critical. A model checker [2] accepts two inputs: a specification $\mathcal{P}$, described in a high level formalism and a set of testable properties, ϕ. A model checker generates and exhaustively searches a finite state model $\mathcal{M}(\mathcal{P})$ to confirm if a property holds, or conversely, report a violation of the system specification. The intuition being, bugs found in the model of the system will reveal bugs in the system design. However, the application of model checking is limited as the state-space of even moderately sized concurrent systems can be too large for state-of-the art machines to exhaustively search. Although verification algorithms have a linear run time complexity, this is offset as the number of states in a model grows exponentially as parameters are added. Consequently, research often focuses on techniques to reduce the impact of the state-space explosion.

One such technique is symmetry reduction [1]. If a concurrent system is comprised of many replicated processes then checking a model of the system may involve redundant search over equivalent, or symmetric, areas of the state-space. Symmetry reduction is concerned with exploiting these underlying regularities by only storing one representative of a structure. For highly symmetric systems, this can result in a reduction factor exponential to the number of system components. However, standard symmetry reduction techniques often discount many symmetries. These symmetries are discounted due to a process having a transition that distinguishes it from other processes while all other behaviour is shared [4].

1.1 Partial Symmetry Reduction

A system can be considered partially symmetric if the majority of updates can be performed by most processes in a specific context. Therefore, a large degree of similarity can be observed between the processes in a system. In an attempt to exploit these symmetries it is better to represent all similar behaviour between processes as a single program and annotate the exceptions.

An annotation is a partition of the set of process indices [6]. A state is marked with an annotation if it lies on a path containing a transition that distinguishes two components. The annotation places the violating process indices into separate partitions. Upon further exploration of the path only processes in the same partition may be permuted. This can lead to the situation where a state is reached along two paths: one with many asymmetric transitions and the other containing only symmetric transitions. In order to make the largest reduction in the state space, it is assumed the state was reached through the symmetric path. The asymmetric state is said to have been subsumed by the symmetric one. This leads to a reduction technique only suitable for reachability analysis as it creates a quotient structure that is not bisimulation-equivalent to the original.

1.2 Detecting Partial Symmetries

A system comprised of n processes can be abstractly represented as a state transition diagram where local states are nodes and local transitions edges. These transitions have the form

$$A \xrightarrow{\phi,\mathcal{Q}} B$$

The transition predicate ϕ takes two variables as input: a state s defining the context in which the edge is to be executed and a process id i. The predicate $\phi(s, i)$ returns the value true if in state s, process i is allowed to make a transition from local state A to local state B. Predicate ϕ can be written in any logical formalism that allows for basic arithmetic operation to be conducted on state and index variables. $\mathcal{Q}$ defines the partitioned set of permutations that preserve predicate ϕ. It is critical for an adaptive symmetry reduction algorithm dealing with a partially symmetric system to be capable of finding $\mathcal{Q}$, thus enabling states to be annotated.

Current approaches to the reduction of partially symmetric systems suggest symmetries be specified by the user, manually or with the aid of a proposition solver [6]. This hinders reduction by adding the overhead of repetitive computation and requires the user to be an expert in the technique. For symmetry reduction to be viable it must be possible to calculate the state annotations without explicitly building the state space or placing the onus on the user. Therefore, we propose a technique capable of calculating $\mathcal{Q}$ directly from the source text of a program.

It has been establish [1] that there is a correspondence between symmetries in a model's underlying communication structure and those in the subsequently generated Kripke structure. To determine the symmetry present in a system, a structure called a static channel diagram [3], a graphical representation of potential communication within the system, can be generated directly from a model specification. Typically the static channel diagram of a system is a small graph and symmetries of this graph can be efficiently found.

Symmetry of the static channel diagram induces symmetry in the model corresponding to the system, provided certain conditions are satisfied. These are generally conditions on assignments to process ID sensitive variables and can be efficiently checked. The set of strictly valid symmetries can therefore be calculated and tagged to each individual update in the high level system description [3]. These tags represent the values of i for which predicate $\phi(s, i)$ returns true. If the update is enabled in the context of state s

This information can be used by an adaptive symmetry reduction algorithm [6] to explore a partially symmetric state space. From an initial state s the algorithm attempts to generate and explore the reachable part of the state space. Successor states of s are found by iterating through all edges valid in the context of state s. Due to the tags, information regarding the solution to $\phi(s, i)$ is available. However, as we are dealing with a partially symmetric system, two slightly differing processes may have the same available update. The update common to these processes will be tagged with a different set of *personal symmetries*. Therefore, at each state, tags can be concatenated for identical edge updates regardless of the process that initiates them. This allows for the largest possible value of $\mathcal{Q}$ to be returned and the coarsest annotation appended to the state. The algorithm continues by reconciling any symmetry violations so only states not subsumed by others are explored.

2 Conclusion

The use of an adaptive symmetry reduction algorithm allows for potentially larger reductions in the state space of a partially symmetric system to be achieved. Encouraging experimental results for this technique have been shown using the SMC model checker [5]. However, as with all current approaches, information regarding partial symmetries is assumed to be known prior to reduction. The proposed technique would alleviate this issue by enabling annotations to be calculated directly from the source text. This in turn allows the application of an adaptive symmetry reduction algorithm to be fully automated.

References

[1] E.M. Clarke, E.A. Emerson, S. Jha, and A.P. Sistla. Symmetry reductions in model checking. In *Computer Aided Verification*, volume 1427 of *Lecture Notes in Computer Science*, pages 147–158. Springer, 1998.

[2] E.M. Clarke, O. Grumberg, and D.A. Peled. *Model checking*. Springer, 1999.

[3] A.F. Donaldson, A. Miller, and M. Calder. Finding symmetry in models of concurrent systems by static channel diagram analysis. *Electronic Notes in Theoretical Computer Science*, 128(6):161–177, 2005.

[4] E.A. Emerson and R.J. Trefler. From asymmetry to full symmetry: New techniques for symmetry reduction in model checking. In *Conference on Correct Hardware Design and Verification Methods*, pages 142–156. Springer, 1999.

[5] A.P. Sistla, V. Gyuris, and E.A. Emerson. SMC: a symmetry-based model checker for verification of safety and liveness properties. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 9(2):133–166, 2000.

[6] T. Wahl. Adaptive symmetry reduction. In *Computer Aided Verification*, volume 4590 of *Lecture Notes in Computer Science*, page 393. Springer, 2007.

Solving Mutilated Problems

Ramin Ramezani and Simon Colton
[⋆]Computational Creativity Group, Department of Computing, Imperial College, London, UK
raminr,sgc@doc.ic.ac.uk

1 Introduction and Motivation

Constraint solving, theorem proving and machine learning provide powerful techniques for solving AI problems. In all these approaches, information known as background knowledge needs to be provided, from which the system will infer new knowledge. Often, however, the background information may be obscure or incomplete, and is usually presented in a form suitable for only one type of problem solver, such as a first order theorem prover. In real world scenarios, there may not be enough background information for any single solver to solve the problem, and we are interested in cases where it may be possible to combine a machine learner, theorem prover and constraint solver in order to best use their incomplete background knowledge to solve the problem. We present here some preliminary experiments designed to test the feasibility of such an approach. We concentrate on the scenario of a police investigation of a murder case. In such a scenario, there may be previous solved cases which bear resemblance to the current case. Given that the previous cases were solved, one can imagine employing a machine learning system to learn a set of rules which can classify suspects in a case as either guilty or not guilty. The rule set could then be applied to the current case. If only one person was classified as guilty, this would solve the problem. While this reasoning may not be sound, it would at least help to identify a *prime suspect*. In addition, in the current case, there may be information describing the particulars of the case, arising from physical evidence, motives, alibis, general knowledge, etc. If so, it may be possible to define a set of constraints that the guilty suspect must satisfy, and then use a constraint solver to rule out suspects. If only one suspect satisfies all the constraints, again the problem is solved. Alternatively, the same information about the case may be used as axioms in a theorem proving setting. In such a setting, one could attempt to prove a set of conjectures, each one stating that a particular suspect is guilty/not guilty. If only one suspect is proved to be guilty (or alternatively, it is possible to prove that all but one suspects are not guilty), then the problem is once again solved.

2 The Mutilated Aunt Agatha Problem

To show the feasibility of using three different types of solvers to attack the same problem, we looked at the "Who Killed Aunt Agatha" problem from the TPTP library (i.e., problem PUZ001 in (Sutcliffe and Suttner, 1998), originally from (Pelletier, 1986)). The background knowledge for this problem is stated in English as follows: Someone who lives in Dreadbury Mansion killed Aunt Agatha. Agatha, Butler and Charles live in Dreadbury Mansion and are the only people who live therein. A killer always hates his victim and is never richer than the victim. Charles hates no one that Aunt Agatha hates. Agatha hates everyone except the Butler. The Butler hates everyone not richer than Agatha. The Butler hates everyone Aunt Agatha hates. No one hates everyone and Agatha is not the butler. This problem is usually posed as a logic puzzle for theorem provers, where the aim is to prove that Aunt Agatha killed herself. However, in a more general setting, the answer wouldn't be given, i.e., we would be asked to find out who killed Aunt Agatha. With this tweak, we can make it amenable to the three different solving approaches as described above.

To show that – in principle – such problems are amenable to a machine learning approach, we firstly invented some data which embodies the axioms of the problem. In particular, we wrote down the details of five case studies with three people in, one of whom had been murdered. We specified who was richer than who, who hated who, who was killed and who the murderer had turned out to be. This was done in such a way that (a) there was a direct mapping from Agatha, Butler and Charles to one of the people in the case study, where the Agatha character was always killed and (b) all the axioms from the problem statement about who could possibly hate who, etc., were upheld. In the first instance, the data reflected the fact that the murderer and the victim were always the same person – the Agatha character. This data was produced in the syntax of the Progol machine learning system (Muggleton, 1995). We ran Progol and it hypothesised the rule that $killed(A, A)$. Given that Progol's output is generated in Prolog syntax, it was very easy to check that this profile applied to only Aunt Agatha in the current case. To make matters more interesting, in the second instance, we generated the data to still satisfy the axioms, but we varied the murderer/victim combination. In this instance, Progol hypothesised the following rule: $killed(A, B) \leftarrow hates(A, B), \neg richer(A, B)$. Again, when we applied this to the data about the current case, only Aunt Agatha fitted the profile.

To show that such problems are amenable to a constraint solving approach, we wrote a constraint satisfaction problem (CSP) in the syntax of the Sicstus Prolog CLPFD module (Carlsson et al., 1997). In brief, the CSP had one variable which could take one of three values representing Agatha, Butler and Charles respectively, and was constrained as per the axioms of the problem. Sicstus solved the problem in such a way that it ruled out Butler and Charles, but could not rule out Agatha, hence solving the problem of who killed Aunt Agatha. Finally, we specified six conjectures to the Otter theorem prover (McCune, 1994). The axioms in the conjectures represented the information from the problem statement, and the conjectures were respectively: Agatha killed/didn't kill Agatha; Butler killed/didn't kill Agatha; Charles killed/didn't kill Agatha. Otter successfully proved that Agatha killed Agatha, and that Butler and Charles didn't kill Agatha. If failed to prove any of the other conjectures. This shows that such *whodunnit* problems are amenable to solution by theorem provers.

The Aunt Agatha problem becomes more interesting if we remove information from each of the three problem statements in such a way that neither Progol, Sicstus nor Otter can solve the problem. We can then investigate methods for combining these reasoning systems in such a way that a solution can still be found. Our experiments are still preliminary, and we plan in future to investigate many different ways to mutilate the problem, yet still solve it via a combination of systems. So far, we have only investigated one opportunity for combining different reasoning systems. In particular, from the theorem proving and CSP problems, we removed the axiom that "no-one hates everyone". This is crucial to solving the problem, because without it, Sicstus cannot rule out Butler as the killer, and Otter can similarly prove that both Butler and Agatha killed Agatha. We investigated whether the data from the machine learning approach could be used to recover the missing axioms. In particular, we employed the HR automated theory formation (Colton, 2002) to form a theory about the previous case studies. Details are omitted, but using HR's `forall`, `exists`, `negate` and `compose` production rules, HR made the conjecture that in all case studies: $\nexists x$ s.t. $person(x) \wedge (\forall y, (person(y) \rightarrow hates(x, y)))$. This states that, in all cases, there is no person who hates everyone. Hence we see that HR has recovered the missing axiom, which could be used by the constraint solver or prover to solve the problem.

3 Future Work

We are building a system which is able to take a general problem statement, such as a *whodunnit* problem and translate it to the syntax of various solvers such as Sicstus, Otter and Progol. Moreover, in situations where none of the solvers are initially successful, the system will be able take the partial solutions from each solver and see whether these can be used together to fully solve the problem. In addition, the system will employ a theory formation program such as HR to discover potential axioms exhibited by the data which enable a solution to be found. This will give us a platform to investigate more exotic combinations of reasoning systems which are able to solve ill-formed problems. For instance, the axioms provided to a theorem prover could be used to generate artificial data to supplement the given data in a machine learning problem, hopefully enabling the learner to solve the problem. We believe that such combined AI systems will enable more powerful, more flexible solvers to be built and employed.

Acknowledgements

This work is funded by EPSRC grant EP/F036647. We would like to thank John Charnley and Murray Shanahan for their input into the mutilated Aunt Agatha problem.

References

M Carlsson, G Ottosson, and B Carlson. An open-ended finite domain constraint solver. In *Proc. Programming Languages: Implementations, Logics, and Programs*, 1997.

S Colton. *Automated Theory Formation in Pure Mathematics*. Springer-Verlag, 2002.

W McCune. Otter 3.0 Reference Manual and Guide. Technical Report ANL-94/6, Argonne National Laboratory, Argonne, USA, 1994.

S Muggleton. Inverse entailment and Progol. *New Generation Computing*, 13:245–286, 1995.

F Pelletier. Seventy-five Problems for Testing Automatic Theorem Provers. *Journal of Automated Reasoning*, 2(2):191–216, 1986.

G Sutcliffe and C Suttner. The TPTP problem library: CNF release v1.2.1. *Journal of Automated Reasoning*, 21(2): 177–203, 1998.

Diagrammatic Reasoning for Software Verification

Matt Ridsdale, Mateja Jamnik[⋆]
[⋆]University of Cambridge
Computer Laboratory
William Gates Building
15 JJ Thomson Avenue
Cambridge CB3 0FD, U.K.
mer39@cam.ac.uk, mj201@cl.cam.ac.uk

Nick Benton, Josh Berdine[†]
[†]Microsoft Research Cambridge
Roger Needham Building
7 JJ Thomson Avenue
Cambridge CB3 0FB, U.K.
nick@microsoft.com, jjb@microsoft.com

1 Introduction

Diagrams have historically been seen as informal aids to understanding, rather than rigorous mathematical tools. However they are widely used in mathematics teaching and informal communication. Recent work has challenged the historical view, with formal diagrammatic reasoning systems implemented for geometry, arithmetic, and other areas of mathematics. (Mumma, 2009; Jamnik, 2001; Barker-Plummer and Bailin, 1997).

This work is about formalising diagrams for program verification. Program verification using symbolic logic is a well-developed field, but anecdotes suggest that diagrams are widely used by researchers in informal reasoning and communication about the subject. Typically, a "boxes and arrows" diagram is used to indicate the input state to the program, and modifications to this diagram trace the program's execution. Humans can informally "verify" an algorithm this way by satisfying themselves that it works correctly for some typical input. We therefore specify a formal semantics of box-and-arrow diagrams and define operations corresponding to the program commands of our language.

1.1 Why diagrams?

As compared to symbolic logic, diagrammatic languages can be seen as domain-specific, while symbolic logics such as predicate logic are generic. This specificity enables domain knowledge to be encoded in the representation, which allows diagrams to concisely capture information which would require a verbose symbolic description, and can support inference by decreasing the proof search space. Diagrams offer other advantages too: diagrammatic proofs can be easier for humans to understand, and they can support the inference of general conclusions by reasoning about specific examples.

Our system will be used to investigate whether the above advantages can be exploited in this problem domain.

2 Reasoning With Diagrams

We intend to make use of diagrams in at least three aspects of program verification: writing the program specifications, deciding entailment problems between program states, and reasoning about programs. We briefly describe an approach to each of these.

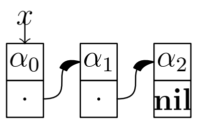


Figure 1: Examples of what our diagrams look like diagrams. This diagram represents a linked list.

Fig. 1 shows some examples of the kinds of diagrams we are using. We define a syntax and semantics of diagrams such that each square represents a memory location, which can hold values (α_i or nil) or pointers to other memory locations. Adjacent squares represent adjacent cells in memory, and the roman letters outside of the squares represent program variables, which store memory locations or values. Thus each diagram represents a set of memory states.

The semantics of programs can be specified as a partial function from sets of program states to sets of program states, as in the well-known notion of *Hoare triples*. The Hoare triple $\{P\}\ \mathsf{prog}\ \{Q\}$ asserts that if prog is run in a state in which the predicate P holds, and if prog terminates, then the predicate Q holds in the resulting state. In our system, P and Q will be replaced with diagrams describing sets of states, enabling us to write program specifications using diagrams.

In software verification, there are two levels of inference that can be performed: entailments between static program states, and reasoning about the effects of program commands. For example in the Hoare triple

$$\{x = 4 \wedge y = x\}\, z := y\, \{x = 4 \wedge y = x \wedge z = 4\}$$

we must first deduce that $(x = 4 \wedge y = x \Rightarrow y = 4)$ before we can deduce that the triple holds. This can be accomplished with diagrams by defining operations which strengthen, weaken or preserve a diagram's meaning. An example was given in Ridsdale et al. (2008) of a diagrammatic proof of entailments between program states (Fig. 2). Fig. 2 shows a pair of lists which are implicitly connected, as the value y at the end of the first list equals the address of the head of the second list. A diagrammatic proof that the lists are connected need consist of only a single operation, replacing both instances of y with a pointer. We argue this is more intuitive than the corresponding symbolic proof in separation logic.

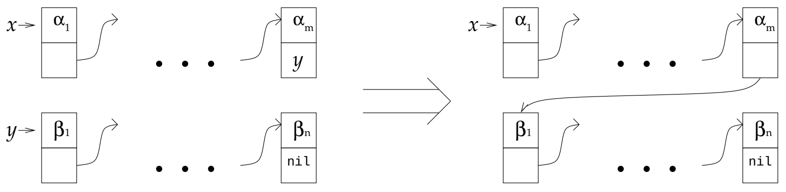


Figure 2: A *list segment* and a *list*, which are implicitly connected.

3 Reasoning About Programs

Our approach to reasoning about programs is based on Jamnik (2001), on diagrammatic proofs in arithmetic, in which a diagrammatic proof consists of a program which acts on diagrams. We have a set of diagrammatic operations corresponding to commands in our programming language, and use those to model the program to be verified with a program on diagrams. The hope is that a way can be found to do this such that the resulting diagrammatic program is easier to verify than the original program, but we have not yet been able to determine that this will be the case.

References

Dave Barker-Plummer and Sidney C. Bailin. The role of diagrams in mathematical proofs. *Machine Graphics and Vision*, 6(1):25–56, 1997.

Mateja Jamnik. *Mathematical Reasoning with Diagrams: From Intuition to Automation*. CSLI Press, Stanford, CA, USA, 2001.

John Mumma. Proofs, pictures, and euclid. http://www.contrib.andrew.cmu.edu/ jmumma/list.html. *Synthese (To appear)*, 2009. URL `http://www.contrib.andrew.cmu.edu/ jmumma/list.html`.

Matt Ridsdale, Mateja Jamnik, Nick Benton, and Josh Berdine. Diagrammatic reasoning in separation logic. In *Diagrammatic Representation and Inference, 5th International Conference, Herrsching, Germany. Proceedings*, volume 5223 of *Lecture Notes in Computer Science*, pages 408–411. Springer, 2008.

Semantic Embedding of Promela-Lite in PVS

Shamim H Ripon[*] Alice Miller
[*]Department of Computing Science
University of Glasgow
{shamim,alice}@dcs.gla.ac.uk

1 Introduction

Promela-Lite [3] is a specification language which captures the essential features of Promela [4]. Unlike Promela a full grammar and type system of the language, and Kripke structure semantics of Promela-Lite specification have been defined and used to prove the correctness of automatic symmetry detection techniques used in Promela.

Mechanical verification is widely used as a tool to verify the properties of a language. It allows one to identify potential flaws in the language and gives confidence in the language definition. Theorem provers are heavily used as a tool to mechanically verify language properties. The language is to be embedded into the theorem prover for this purpose. Here we outline work in progress to embed Promela-Lite syntax and semantics into the general purpose theorem prover PVS [5] and use these embeddings to interactively prove both consistency with the syntax/semantics definitions and language properties.

2 Promela-Lite

Promela-Lite includes the core features of Promela including parameterised processes, channels (first class) and global variables, but omits some features such as rendez-vous channels, enumerated and record types, and arrays. The syntax of Promela-Lite is specified by using the standard BNF form [1] summarised in Figure 1(a). The language has primitive datatypes and channel types of the form $chan\{\overline{T}\}$, where $\overline{T}$ is comma separated list of types (details in [3]). A Promela-Lite specification consists of a series of channel and global variable declarations, one or more proctypes and an `init` process. Part of Promela-Lite syntax is shown as a production rule in Figure 1(b), where only the syntax of `expr` is presented.

$$\begin{array}{rcl}
\langle type \rangle & ::= & \texttt{int} \\
 & | & \texttt{pid} \\
 & | & \langle chantype \rangle \\
 & | & \langle typevar \rangle \\
\langle chantype \rangle & ::= & \langle recursive \rangle^{?}\ \texttt{chan}\ \{\langle type - list, `,' \rangle\} \\
\langle recursive \rangle & ::= & \texttt{rec}\ \langle typevar \rangle \\
\langle typevar \rangle & ::= & \langle name \rangle
\end{array}$$

(a) Promela-Lite type systax

$$\begin{array}{rcl}
\langle expr \rangle & ::= & \langle name \rangle \\
 & | & \langle number \rangle \\
 & | & \texttt{\_pid} \\
 & | & \texttt{null} \\
 & | & \texttt{len}(\langle name \rangle) \\
 & | & (\langle expr \rangle) \\
 & | & \langle expr \rangle \circ \langle expr \rangle \quad (\text{where} \circ \in \{+,-,*\})
\end{array}$$

(b) Syntax for `expr`

Figure 1: Promela-lite syntax

A Promela-Lite specification is considered to be well-typed if its statements and declarations are well-typed according to the typing rules. The typing rules of Promela-Lite are defined by following the notation used in [2]. For example, the typing rule for $\langle expr \rangle \circ \langle expr \rangle$ in Figure 1(b) is defined as follows:

$$\frac{\Gamma \vdash e_1 : int \qquad \Gamma \vdash e_2 : int \qquad \circ \in \{+,-,*\}}{\Gamma \vdash e_1 \circ e_2 : int}$$

The semantics of a Promela-Lite specification $\mathcal{P}$ is denoted as a Kripke structure $\mathcal{M}$. If $\mathcal{P}$ is well-typed according to the typing rules then the Kripke structure $\mathcal{M}$ is well-defined. A function $eval_{p,i}$ is defined that evaluates an expression e for a process i at state s, where $proctype(i) = p$. For the syntax of `expr` mentioned earlier, $eval_{p,i}$ is used to evaluate the expressions at a given state as follows (details in [3]):

$$eval_{p,i}(s, e_1 \circ e_2) \ = \ eval_{p,i}(s, e1) \circ eval_{p,i}(s, e_2) \qquad \text{where } \circ \in \{+,-,*\}$$

[*]The author is supported by EPSRC grant EP/E032354/1

3 PVS Embedding

An embedding is a semantic encoding of one specification language into another. There are two main variants for the embeddings: *shallow* and *deep* embeddings [6]. In a shallow embedding, a program or a specification is translated into a semantically equivalent representation of the host logic. In a deep embedding, the language and the semantics are fully formalised in the logic of the specification language. This allows reasoning about the language itself, not just concrete programs.

Our mechanisation is based on deep embedding. Figure 2 briefly outlines the steps that we follow to mechanise Promela-Lite in PVS. The syntax of the language is defined using the abstract datatype mechanism of PVS which allows recursive definitions over the terms of the language. The definition is similar to the traditional use of BNF to define the syntax. The datatype definition enumerates constructors, lists their parameters, and provides recogniser predicates. Part of the datatype definition to define the syntax of `expr` ($\langle expr\rangle \circ \langle expr\rangle$) is shown here:

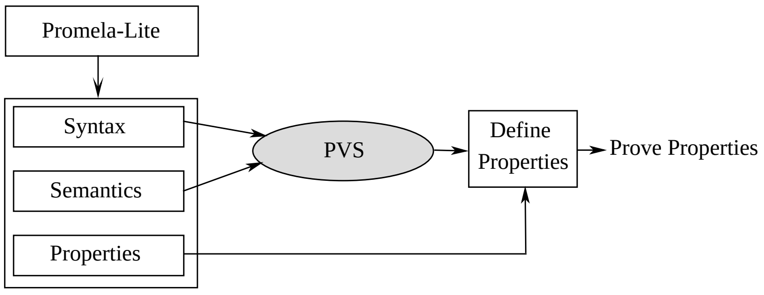


Figure 2: Mechanisation steps

```
expr_syntax : DATATYPE
 BEGIN
  plus (e1:expr_syntax, e2:expr_syntax) : plus?
  minus(e1:expr_syntax, e2:expr_syntax) : minus?
  star (e1:expr_syntax, e2:expr_syntax) : star?
  ...
END expr_syntax
```

The semantics are defined recursively by following the original definitions. Care has to be taken to ensure that these definitions follow the typing rules. Following the definitions of how expressions are evaluated, a recursive definition is given to the semantics, part of which is shown as follows:

```
sem(e : expr_syntax, env: environment, s : STATE): RECURSIVE VALUE =
 CASES e OF
  plus(e1,e2) : integer_value(intvalue(sem(e1,env,s)) + intvalue(sem(e2,env,s))
  ...
 ENDCASES  MEASURE BY <<
```

After embedding the semantics, the properties to be proved will be defined in the form of theorems and supporting lemmas, and proved using the interactive theorem prover.

References

[1] Alfred V. Aho, Ravi Sethi, and Jeffrey D. Ullman. *Compilers – Principles, Techniques, and Tools*. Addison-Wesley, 1986.

[2] Luca Cardelli. *The Computer Science and Engineering Handbook*, chapter Type Systems, pages 2208 – 2236. CRC Press, Boca Raton, 1997.

[3] Alastair F. Donaldson and Alice Miller. Automatic symmetry detection for promela. *Journal of Automated Reasoning*, 41:251–293, 2008.

[4] Gerard J. Holzman. *The SPIN MODEL CHECKER: Primer and Reference Manual*. Addison-Wesley, 2003.

[5] Sam Owre, J.M. Rushby, and N Shankar. PVS: A Prototype Verification System. In Deepak Kapur, editor, *CADE'92*, volume 607 of *LNAI*, pages 748–752. Springer-Verlag, June 1992.

[6] R. Boulton, A. Gordon, M.J.C. Gordon, J. Herbert, and J. van Tassel. Experience with embedding hardware description languages in HOL. In *TPCD'93*, pages 129–156. North-Holland, 1993.

The Ackermann Approach for Modal Logic, Correspondence Theory and Second-Order Reduction: Extended Abstract

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

Abstract

We introduce improvements for second-order quantifier elimination methods based on Ackermann's Lemma and investigates their application in modal correspondence theory. In particular, we define refined calculi and procedures for solving the problem of eliminating quantified propositional symbols from modal formulae. We prove correctness results and use the approach to compute first-order frame correspondence properties for modal axioms and modal rules. Our approach can solve two new classes of formulae with wider scope than existing classes known to be solvable by second-order quantifier elimination methods.

1 Second-Order Quantifier Elimination

An application of second-order quantifier elimination is correspondence theory in modal logic. Propositional modal logics, when defined axiomatically, have a second-order flavour, but can often be characterized by classes of model structures which satisfy first-order conditions. Frequently, with the help of second-order quantifier elimination methods, these first-order conditions, called frame correspondence properties, can be automatically derived from the axioms. For example, using the standard relational translation method the modal axiom **D** $= \forall p[\Box p \rightarrow \Diamond p]$ translates to this second-order formula:

$$\forall P \forall x[\forall y[R(x,y) \rightarrow P(y)] \rightarrow \exists z[R(x,z) \wedge P(z)]]. \tag{1}$$

This formula is equivalent to a first-order formula, namely $\forall x \exists y[R(x,y)]$, and is the first-order correspondence property of axiom **D**. It can be derived automatically with a second-order quantifier elimination method by eliminating the second-order quantifier $\forall P$ from (1).

Several second-order quantifier elimination methods exist. These methods belong to two categories: (i) substitution-rewrite approaches which exploit monotonicity properties, and (ii) saturation approaches, which are based on exhaustive deduction of consequences. Methods following the substitution-rewrite approach include the Sahlqvist-van Benthem substitution method for modal logic, the DLS algorithm introduced by Szalas in (1993) and together with Doherty and Lukaszewicz in (1997), the SQEMA algorithm for modal logic introduced by Conradie, Goranko and Vakarelov in (2006). Methods following the saturation approach include the SCAN algorithm of Gabbay and Ohlbach (1992), and hierarchical resolution of Bachmair, Ganzinger and Waldmann (1994).

Here, I am interested in methods using the substitution-rewrite approach to second-order quantifier elimination. In particular, my focus is on methods that are based on a general substitution property found in Ackermann (1935). This result, called *Ackermann's Lemma*, tells us when quantified predicate symbols are eliminable from second-order formulae. For propositional and modal logic Ackermann's Lemma can be formulated as follows. In any model,

$$\exists p[(\alpha \rightarrow p) \wedge \beta(p)] \quad \text{is equivalent to} \quad \beta^p_\alpha, \tag{2}$$

provided these two conditions hold: (i) p is a propositional symbol that does not occur in α, and (ii) p occurs only negatively in β. The formula β^p_α denotes the formula obtained from β by uniformly substituting α for all occurrences of p in β. This property is also true, when the polarity of p is switched, that is, all occurrences of p in β are positive and the implication in the left conjunct is reversed. Applied from left-to-right the equivalence (2) of Ackermann's Lemma eliminates the second-order quantifier $\exists p$. In fact, all occurrences of p are eliminated. This idea can be turned into an algorithm for eliminating existentially quantified propositional symbols. I refer to this algorithm as the *basic Ackermann algorithm*.

2 A Refined Ackermann Approach

Based on the basic Ackermann algorithm I introduce a refined second-order quantifier elimination approach for modal logic. Like the SQEMA algorithm, rather than translating the modal axiom into second-order logic and then passing it to a second-order quantifier elimination method, the approach performs second-order quantifier elimination directly in modal logic. Only in a subsequent step the translation to first-order logic is performed. For example, given the second-order modal formula $\forall p[\Box p \rightarrow \Diamond p]$, the approach first eliminates $\forall p$ from the formula and returns the formula $\Diamond \top$. Subsequently this is translated to first-order logic to give the expected seriality property $\forall x \exists y[R(x, y)]$.

The approach is defined for propositional multi-modal tense logics, more precisely, the logic $\mathbf{K}^{n}_{(m)}(\smile, \pi+)$ with forward and backward looking modalities, nominals, and second-order quantification over propositional symbols.

A main motivation for this work has been to gain a better understanding of when quantifier elimination methods succeed, and to pinpoint precisely which techniques are crucial for successful termination. I define two new classes of formulae for which the approach succeeds: the class $\mathcal{C}$ and an algorithmic version called $\mathcal{C}^{>}$. The classes define normal forms for when Ackermann-based second-order quantifier elimination methods succeed. $\mathcal{C}$ and $\mathcal{C}^{>}$ subsume both the Sahlqvist class of formulae and the class of monadic-inductive formulae of Goranko and Vakarelov (2006). I present minimal requirements for successful termination for all these classes. This allows existing results of second-order quantifier elimination methods to sharpened and strengthened.

I consider two applications of the approach:

(i) Computing correspondence properties for modal axioms *and modal rules*. For example, equivalently reducing axiom **D** to the seriality property, or equivalently reducing the modal rule $\Box p / \Diamond p$ to $\forall x \exists y \exists z[R(x, y) \wedge R(z, y)]$.

(ii) Equivalently reducing of second-order modal problems. For example, the second-order modal formula $\forall p \forall q[\Box(p \vee q) \rightarrow (\Box p \vee \Box q)]$ equivalent reduces to $\forall p[\Diamond p \rightarrow \Box p]$, or the axiom **D** equivalently reduces to $\Diamond\top$.

While the approach follows the idea of the basic Ackermann algorithm and is closely related to the DLS algorithm and the SQEMA algorithm, I introduce a variety of enhancements and novel techniques.

First, which propositional symbols are to be eliminated can be flexibly specified, and the approach is not limited to eliminating all propositional symbols. Second, in order to be able to ensure effectiveness and avoid unintended looping, the approach is enhanced with ordering refinements. In the approach an ordering on the non-base symbols (these are the symbols that we want to eliminate) must be specified and determines the order in which these symbols are eliminated. At the same time the ordering is used to delimit the way that the inference rules are applied. It turns out, that the adoption of ordering refinements allows for a more in-depth analysis of how the inferences are performed and a better understanding of the properties of the approach. Third, for reasons of efficiency, and improved success rate, it is beneficial to incorporate techniques for pruning the search space. A general notion of redundancy is thus included. It is designed so that it is possible to define practical simplification and optimization techniques in a flexible way. Fourth, the approach is defined in terms of calculi given by sets of inference rules. This has the advantage that the approach can be studied independently of practical issues such as rule application order, strategies and heuristics. It allows for a more fine-grained analysis of the computational behaviour of the approach and more general results can be formulated.

3 Results

The following results have been obtained.

1. Any derivation in the approach is guaranteed to terminate and the obtained formula is logically equivalent to the input formula. This means the refined modal Ackermann calculus is correct and terminating.

2. Any problem in the class $\mathcal{C}^{>}$ is effectively and successfully reducible by the rules of the approach using some ordering.

3. For the subclass $\mathcal{C}$ of $\mathcal{C}^{>}$, the sign switching rule, redundancy elimination are not needed, and the ordering is immaterial.

4. Whenever the approach successfully eliminates all propositional symbols for a modal formula α then (a) $\neg\alpha$ is d-persistent and hence canonical, and (b) the formula returned is equivalent to α.

5. All modal axioms equivalent to the conjunction of formulae reducible to clauses in $\mathcal{C}$ and $\mathcal{C}^{>}$ are elementary and canonical.

These results are improvements for substitution-rewrite approaches based on Ackermann's Lemma, and present strengthenings of Sahlqvist's theorem and the corresponding result for monadic-inductive formulae.

The significance of the last result is that axioms that are equivalent to first-order properties and are canonical can be used to provide sound and complete axiomatizations of modal logics.

4 Further details

For a full account of the approach and the results, I refer to Schmidt (2008) and Chapter 13 in Gabbay et al. (2008).

References

D. M. Gabbay, R. A. Schmidt, and A. Szałas. *Second-Order Quantifier Elimination: Foundations, Computational Aspects and Applications*. College Publications, 2008.

R. A. Schmidt. Improved second-order quantifier elimination in modal logic. In *Proc. JELIA'08*, volume 5293 of *LNAI*, pages 375–388. Springer, 2008. The long version is under review for publication in a journal.

Synthesising Tableau Decision Procedures

Renate A. Schmidt[⋆]
[⋆]University of Manchester
renate.schmidt@manchester.ac.uk

Dmitry Tishkovsky[†]
[†]University of Manchester
dmitry.tishkovsky@manchester.ac.uk

Abstract

We formalise sufficient conditions for synthesising sound, complete, and tableau decision procedures for a logic of interest. Given a specification of the formal semantics of a logic, the method generates a set of tableau inference rules which can then be used to reason within the logic. The method guarantees that the generated rules form a calculus which is sound and constructively complete. 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. The process of generating tableau rules can be completely automated and produces, together with blocking, an automated procedure for generating tableau decision procedures for logics.

1 Synthesising Tableau Calculus

Our interest is the problem of automatically generating a tableau calculus for a given logic. We assume that the logic is defined by a high-level specification of the formal semantics of the logic. Our aim is to turn this specification into a set of inference rules that provide a sound and complete tableau calculus for the logic. In addition, for a decidable logic we further want to generate a terminating calculus.

In previous work we have described a framework for turning sound and complete tableau calculi into decision procedures (Schmidt and Tishkovsky, 2008). The prerequisites in the framework are that the logic admits an effective finite model property shown by a filtration argument, and that

(i) the tableau calculus is sound and constructively complete, and

(ii) a weak form of subexpression property holds for tableau derivations.

Constructive completeness is a slightly stronger notion than completeness. It means that for every open branch in a tableau there is a model which reflects all the expressions (formulae) occurring on the branch. The subexpression property says that every expression in a derivation is a subexpression of the input expression with respect to a finite subexpression closure operator.

In order to be able to exploit the ‘automatic termination framework’ results in (Schmidt and Tishkovsky, 2008), our goal is to generate tableau calculi that satisfy the prerequisites (i) and (ii). It turns out that provided that the semantics of the logic is well-defined in a certain sense, the subexpression property can be imposed on the generated calculus. Crucial is the separation of the syntax of the logic from the ‘extras’ in the meta-language needed for the semantic specification of the logic. The process can be completely automated and gives, together with the unrestricted blocking mechanism and the results in (Schmidt and Tishkovsky, 2007, 2008), an automated procedure for generating tableau decision procedures for logics, whenever they have an effective finite model property with respect to a well-defined first-order semantics.

That the generated calculi are constructively complete has the added advantage that models can be effectively generated from open, finished branches in tableau derivations. This means that the synthesised tableau calculi can be used for model building purposes.

The method works as follows.

(a) The user defines the formal semantics of the given logic in a many-sorted first-order language so that certain well-definedness conditions hold.

(b) 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.

Theorem 1 *There exists a procedure for synthesising a sound and constructively complete tableau calculus from a well-defined specification of the semantics of the logic.*

Under certain conditions it is possible to refine the rules of the calculus in order to reduce branching and redundancy in the syntax of the calculus.

2 Synthesising Decision Procedures

If the logic can be shown to admit finite filtration, then the generated calculus can be automatically turned into a terminating calculus by adding the unrestricted blocking mechanism from (Schmidt and Tishkovsky, 2007):

$$\text{(ub)}: \frac{x \approx x,\ y \approx y}{x \approx y \mid x \not\approx y}.$$

Let $t < t'$, whenever the first appearance of term t' in the branch is strictly later than the first appearance of term t. The conditions that blocking must satisfy are:

(i) If $t \approx t'$ appears in a branch and $t < t'$ then all further applications of rules which introduce new terms to formulae containing t' are not performed within the branch.

(ii) In every open branch there is some node from which point onwards before any application of a rule introducing a new term all possible applications of the (ub) rule have been performed.

Theorem 2 *If for every formula φ and model satisfying φ, there is a finite filtrated model satisfying φ, then extending the tableau calculus which is synthesised in Theorem 1 with the* (ub)*-rule gives a terminating calculus.*

This gives a nondeterministic decision procedure which can be turned into a deterministic decision procedure by using appropriate search strategies that are fair.

3 Applications

The method can be applied to many familiar logics that are first-order representable. These include propositional intuitionistic logic, many standard, Kripke complete modal logics, and a wide set of description logics such as $\mathcal{ALCO}$ and $\mathcal{SO}$. $\mathcal{ALCO}$ is the description logic $\mathcal{ALC}$ with singleton concepts, or nominals. $\mathcal{SO}$ is the extension of $\mathcal{ALCO}$ with transitive roles. For these logics, the synthesised tableau calculi are very close to common 'classical' tableau calculi used for satisfiability checking in these logics.

4 Further Details

For further details we refer to (Schmidt and Tishkovsky, 2009), which introduces the method of synthesising tableau calculi, and to (Schmidt and Tishkovsky, 2008), which describes 'automatic termination' of a tableau calculi for logics having the effective finite model property.

5 Conclusion

The results of the paper can be regarded as a mathematical formalisation and generalisation of tableau development methodologies. The formalisation separates the creative part of tableau calculus development which needs to be done by a human developer and the automatic part of the development process which can be left to an automated (currently first-order) prover and an automated tableau synthesiser.

The tableau calculi generated are Smullyan-type tableau calculi, i.e., ground semantic tableau calculi. We believe that other types of tableau calculi can be generated using the same techniques.

Our future goal is to further reduce human involvement in the development of calculi by finding appropriate automatically verifiable conditions for optimal calculi to be generated.

References

Renate A. Schmidt and Dmitry Tishkovsky. Automated synthesis of tableau calculi, 2009. Submitted to *TABLEAUX'09*, http://www.cs.man.ac.uk/~dmitry/papers/astc2009.pdf.

Renate A. Schmidt and Dmitry Tishkovsky. A general tableau method for deciding description logics, modal logics and related first-order fragments. In Alessandro Armando, Peter Baumgartner, and Gilles Dowek, editors, *Proc. IJCAR'08*, volume 5195 of *Lect. Notes Comput. Sci.*, pages 194–209. Springer, 2008.

Renate A. Schmidt and Dmitry Tishkovsky. Using tableau to decide expressive description logics with role negation. In *Proc. ISWC'07*, volume 4825 of *Lect. Notes Comput. Sci.*, pages 438–451. Springer, 2007.

On the Readability of Diagrammatic Proofs

Gem Stapleton[⋆]
[⋆]University of Brighton
Brighton, UK
g.e.stapleton@bton.ac.uk

Mateja Jamnik[†]
[†]University of Cambridge
Cambridge, UK
mateja.jamnik@cl.cam.ac.uk

Judith Masthoff[‡]
[‡]University of Aberdeen
Aberdeen, UK
j.masthoff@abdn.ac.uk

Abstract

Recently, much effort has been placed on developing diagrammatic logics, with a focus on obtaining sound and complete reasoning systems. A hypothesis of the diagrammatic reasoning community is that many people find diagrammatic proofs easier to read than symbolic proofs. This hypothesis has not been thoroughly tested, although significant effort has been directed towards understanding what makes diagrams more readable than symbolic formulae. We are interested in how to automatically find readable diagrammatic proofs. To achieve this aim, significant research is required that builds on the existing state-of-the-art. This extended abstract summarizes our plans for research on this topic.

1 Introduction

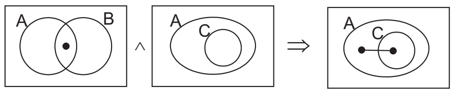
Diagrammatic reasoning has only recently, in the last decade, enjoyed research attention which demonstrated that diagrams *can* be used for formal reasoning. However, diagrammatic proofs have been so far constructed without much attention paid to their readability: a major hypothesis of the diagrammatic reasoning community, which is often assumed, is that they are more readable than symbolic proofs due to their intuitive visual appeal. This may not be the case for an arbitrary diagrammatic proof, and some diagrammatic proofs will be more appealing, easier to understand, or more economical with information than other diagrammatic proofs. In order to take full advantage of the widely acknowledged advantages of diagrammatic reasoning, we need to understand what constitutes, and how to create, readable diagrammatic proofs.

In symbolic reasoning, theorem provers have come a long way in devising strategies, criteria and heuristics for constructing proofs that are human readable, yet these symbolic proofs are still complex for humans to understand. By contrast, diagrammatic reasoners have paid virtually no attention to this readability issue. This extended abstract discusses a plan of work for addressing this issue in a principled way by producing a framework of readability criteria that can be employed in diverse diagrammatic reasoners in the form of heuristics or other higher level strategies. Indeed, we hope that diagrammatic readability criteria, and any principles learned about readability, will extend to symbolic reasoning systems too. Our main hypothesis is: *readable diagrammatic proofs can be constructed automatically by devising and employing readability criteria.* This represents a major challenge in diagrammatic and automated reasoning.

2 Diagrammatic Proofs

We demonstrate the kind of diagrammatic proofs we plan to examine, some possible readability criteria, and how these can be used to guide the search for a more readable diagrammatic proof with a toy example. Initially we will focus our work on the domain of spider diagrams [4] which are used to prove theorems that can be expressed in monadic first order logic with equality. Thus, spider diagrams make statements which provide constraints on set cardinality, such as $|A - B| = 2$, $|A \cap B| \geq 3$, and $A \subseteq C$ (equivalently, $|A - C| = 0$). Diagrams of this type are frequently seen in mathematics text books to intuitively illustrate set theory concepts. Spider diagrams are based on Euler diagrams, augmenting them with shading and so-called spiders. Visually, spiders are trees (dots connected by lines), placed in regions of the diagram; each spider represents the existence of an element in the set represented by the region in which it is placed, thus providing lower bounds on set cardinality. Shading is used to place upper bounds on set cardinality: in a shaded region, all elements are represented by spiders. Here is an example theorem expressed symbolically and using spider diagrams:

$$\exists x\,(x \in A \cap B \wedge C \subseteq A) \Rightarrow \exists x\,\big((x \in A - C \vee x \in A \cap C) \wedge C \subseteq A\big)$$



Two diagrammatic proofs of this theorem are shown in Fig. 1. Notice that the second proof is not only shorter than the first one, but also that its diagrams consist of fewer elements and can therefore be considered to be less cluttered. Spider diagrams are very strongly related to other systems, such as those in [7], and form the basis of more expressive notations making them an ideal choice for investigating readability. We have developed theorem provers for spider diagrams and their Euler diagram fragment but they do not yet incorporate readability criteria except for finding shortest proofs [8].

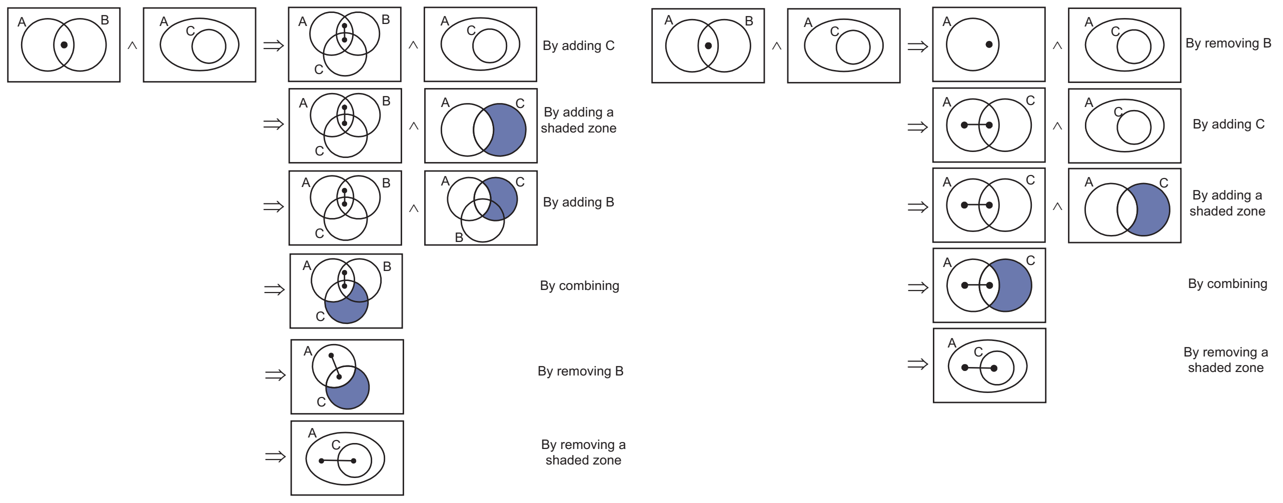


Figure 1: Two diagrammatic proofs.

We propose to devise and use *readability criteria* in order to construct more readable and intuitive diagrammatic proofs than currently possible. Lessons about the representation and use of heuristics, tactics [5], methodicals [6], strategies and so forth in symbolic theorem proving will inform our analysis. In addition, and most importantly, we will also experimentally test what people find more readable in order to identify readability criteria and relevant proof situations for them and also, later, to assess which readability criteria are better than others. Other possible indicators of readability are, for example: (1) diagram clutter: it is easier to read and understand diagrams that have fewer elements in them, (2) length of proof: often shorter proofs are more readable than longer ones, (3) known lemmas: if an inference rule leads to a statement proved before, then this lemma can be used rather than derived again, (4) the size of the step that an inference rule makes: too big steps may be obscure, but too small steps may be tedious, and (5) topological properties matching semantics: diagrams that are better matched to their semantics may be more readable.

Investigating symbolic theorem provers, especially proof planners [1], that already use techniques to guide search [2] will inform our work. In particular, proof planners such as λ*Clam* [6] and ISAPLANNER [3] use methods and methodicals to structure the search. Similarly, ΩMEGA uses control rules and strategies. It is not clear what will be the best framework for employing readability criteria in order to guide the search to produce readable proofs; devising this framework is one of our main goals. Once we have this framework, we will implement it in a diagrammatic theorem prover for spider diagrams. The hope is that any principles learned are general enough that they will extend to symbolic reasoners as well.

Acknowledgements Gem Stapleton is supported by EPSRC grant EP/E011160/1 for the Visualisation with Euler Diagrams project. Mateja Jamnik is supported by an EPSRC Advanced Fellowship GR/R76783/01.

References

[1] A. Bundy. The use of explicit plans to guide inductive proofs. 9th Conf. on Automated Deduction, Springer, 111–120, 1988.

[2] L. A. Dennis, M. Jamnik, and M. Pollet. On the comparison of proof planning systems: LambdaClam, Omega and IsaPlanner. ENTCS 151(1):93–110, 2006.

[3] L. Dixon. *A Proof Planning Framework for Isabelle*. PhD thesis, University of Edinburgh, UK, 2005.

[4] J. Howse, G. Stapleton, and J. Taylor. Spider diagrams. *LMS J. of Computation and Mathematics*, 8:145–194, 2005.

[5] L.C. Paulson. *Isabelle: A generic theorem prover*. LNCS 828, Springer, 1994.

[6] J.D.C. Richardson, A. Smaill, and I. Green. System description: proof planning in higher-order logic with lambda-clam. *15th Conf. on Automated Deduction*, Springer, 129–133, 1998.

[7] S.-J. Shin. *The Logical Status of Diagrams*. Cambridge University Press, 1994.

[8] G. Stapleton, J. Masthoff, J. Flower, A. Fish, and J. Southern. Automated theorem proving in Euler diagrams systems. *J. of Automated Reasoning*, 39:431–470, 2007.

Efficient Ground Satisfiability Solving in an Instantiation-based Method for First-order Theorem Proving

Christoph Sticksel[⋆]
[⋆]University of Manchester
csticksel@cs.man.ac.uk

1 Introduction

In the domain of first-order theorem proving, recently so-called instantiation-based methods have gained some attention as they promise advantages over longer established methods. Although modern instantiation-based methods for first-order logic were proposed only in this decade, they are already competitive with and in some areas outperform classical methods that have a history of development since the 1970s. Because of the novelty of instantiation-based methods, not many results exist for equational reasoning and reasoning modulo theories. An integration of these features would make a calculus more mature and open up new areas of application. The main subject of this research is the Inst-Gen calculus as initially presented in Ganzinger and Korovin (2003).

The central rationale behind the Inst-Gen calculus and its implementation in the iProver system is the combination of industrial-strength ground satisfiability solving with true first-order reasoning. In particular, the iProver approach aims to harness the strengths of an off-the-shelf ground satisfiability solver in a first-order procedure for satisfiability modulo theories (SMT). The lack of complete handling of quantification is a shortcoming of many SMT provers and their limited support of quantified clauses makes them essentially reason on ground satisfiability. However, they are efficient in ground solving and iProver therefore delegates detecting unsatisfiability to a ground SMT solver and focuses on generating instances from first-order clauses such that ground unsatisfiability can be witnessed.

The work presented here is concerned with efficient integration of ground solving modulo equality into the first-order instantiation process of Inst-Gen. So to speak, the first-order reasoning needs to stand firmly on the ground, but robust integration of ground reasoning also guides the instantiation process through provision of satisfiable models and is used to justify simplifications.

2 The Inst-Gen Calculus

Formally, the set of first-order clauses S is abstracted to a set of ground clauses $S\bot$ by substituting all variables with a distinguished constant $\bot$ with a substitution that is also denoted by $\bot$. If this ground abstraction $S\bot$ is unsatisfiable, then, by Herbrand's Theorem, S is unsatisfiable as well and the procedure can terminate. In the case of satisfiability of $S\bot$, the first-order clause set S is satisfiable if it is saturated under instantiation inferences of the calculus. Otherwise, one needs to continue generating instances either as witnesses for the unsatisfiability of $S\bot$ or to reach saturation.

If the ground abstraction $S\bot$ has been proved to be satisfiable by the ground solver, the first-order reasoning can make use of a ground model $I_\bot$ for $S\bot$ by means of a selection function sel that constrains possible inferences between clauses to their respective selected literals. Although Inst-Gen inferences are sound regardless of the selection function, the selection function adds an amount of goal-direction that is necessary to be efficient in practice. In each clause C, the selection $\mathrm{sel}(C) = L$ returns a literal L in C such that the ground literal $L\bot$ is true in $I_\bot$. See Korovin (2009) for a detailed description of the calculus, especially for arguments of its completeness.

It is important to note that literal selection in the Inst-Gen calculus is different from literal selection in resolution, as described e.g. in Bachmair and Ganzinger (2001). The selection function does select exactly one literal from every clause, but is not fixed throughout the derivation. It may need to be adapted to reflect changes of the ground model $I_\bot$ when instances are added to S.

3 The Saturation Process, Ground Solving and Selection

The implementation of the iProver system as described in Korovin (2008) uses a variant of the given-clause algorithm. It keeps two disjoint sets of *active clauses* $\mathcal{A}$ and *passive clauses* $\mathcal{P}$ where all inferences between clauses in $\mathcal{A}$ have been generated. In every step, a clause C_g (the *given clause*) is taken from $\mathcal{P}$, all possible inferences with clauses in $\mathcal{A}$ are added to $\mathcal{P}$ and C_g is moved from $\mathcal{P}$ to $\mathcal{A}$. If $\mathcal{P}$ is empty, then all clauses are in $\mathcal{A}$ and it is saturated, which establishes satisfiability of S. Separately, all clauses in $\mathcal{A}$ and $\mathcal{P}$ are grounded with the $\bot$ constant and checked for unsatisfiability in the ground solver. The procedure then either terminates with unsatisfiable as a result or obtains a model $I_\bot$ to be used for the selection function.

The model $I_\bot$ is given by a ground solver that may choose to discard as much of the model from the previous step as it sees fit, therefore giving rise to an uncontrollable number of changes to the selection function. When $\mathrm{sel}(C)$ for a clause C in the active set $\mathcal{A}$ changes, the clause C has to be moved to the passive set $\mathcal{P}$ as with a different selection new inferences may become possible that have not been generated, violating the invariant of saturation of $\mathcal{A}$ under inferences.

Therefore, it seems beneficial to attempt to keep the selection in the active clause set $\mathcal{A}$ and only to change it when necessary, i.e. when there is no model of $S\bot$ containing the selected literals $\bigcup_{C\in\mathcal{A}} \mathrm{sel}(C)\bot$. This lazy strategy allows for a deviation from the particular model the ground solver provides by maintaining a separate model that follows the literal selection and the heuristics involved while still being sound and complete in the case of clauses without equality.

With equality or other background theories, induced equalities have to be considered on potentially many selected literals, making it seem inefficient to check if a set of selected literals can be extended to a model. Possible alternative strategies for selection would either eagerly follow the model as it is obtained from the ground solver, thus weakening the selection heuristics and causing more frequent moves of clauses from the active set. Alternatively, the incompleteness of checking if the selected literals are a model could be accepted and deferred until the active set becomes saturated. Then, a full check on all selected literals would either bring up inconsistent literals and subsequently move clauses or find the selected literals to be consistent and thus prove satisfiability.

4 Results and Future Work

Which strategy is most successful will certainly depend on the solver, its strategies and the amount of cooperation, and, of course, on the clause set itself. Ongoing work is evaluating CVC3 and Z3, two leading SMT solvers, as ground satisfiability solvers modulo equality and modulo theories as well as the most successful strategies for literal selection as described above. Early experiments with the problems of the TPTP library show that the lazy strategy leaves only relatively few problems with an inconsistent set of selected literals on saturation. Moreover, in most cases the problem is proved satisfiable with a literal selection that is a model for $S\bot$ after only one or two further checks, indicating that the lazy strategy might be viable for ground solving modulo equality and theories.

Comparing a version of iProver using MiniSAT for ground solving without equality with an early implementation integrating CVC3 as a ground solver modulo equality on the TPTP library (detailed figures in table 1) show that on the one hand, there is a significant overhead for the SMT solving such that SAT solving is still faster in many cases. On the other hand, there are cases where ground equational reasoning is profitable and the overhead pays off such that additional problems can be solved. As expected, ground solving with equality is most useful for pure equational problems.

Equational	Num.	only in iProver +		faster in iProver +	
atoms	Probl.	CVC3	MiniSAT	CVC3	MiniSAT
0%	3130	6	346	522	1736
0% - 10%	1915	24	118	54	449
10% - 20%	2622	49	76	205	408
20% - 30%	1593	15	44	47	336
⋮					
100%	1515	205	26	85	100

Table 1: Solved problems in TPTP v3.5.0 grouped by percentage of equational atoms. No significant number of problems available with between 30% and 100% of equational atoms. Run on AMD Athlon XP 2200+ with 500 MB and a timeout of 3 mins.

Future work will use the good grip the iProver system is getting on ground solving modulo equations and theories to progress with instance generation of first-order clauses modulo equality and theories where lifting the ground model to obtain a literal selection is even more important and thus help iProver climb up to the important application areas of satisfiability modulo theories.

Acknowledgements

Joint work with Konstantin Korovin and Renate Schmidt, based on the iProver system, as in described in Korovin (2008).

References

Leo Bachmair and Harald Ganzinger. Resolution Theorem Proving. In Alan Robinson and Andrei Voronkov, editors, *Handbook of Automated Reasoning*, volume 1. Elsevier Science and MIT Press, 2001.

Harald Ganzinger and Konstantin Korovin. New Directions in Instantiation-Based Theorem Proving. In *Symposium on Logic in Computer Science, LICS 2003. Proceedings*, pages 55–64, 2003.

Konstantin Korovin. An Invitation to Instantiation-Based Reasoning: From Theory to Practice. In A. Podelski, A. Voronkov, and R. Wilhelm, editors, *Volume in memoriam of Harald Ganzinger*, Lecture Notes in Computer Science. Springer, 2009. Invited paper. To appear.

Konstantin Korovin. iProver - An Instantiation-Based Theorem Prover for First-Order Logic (System Description). In *International Joint Conference on Automated Reasoning, IJCAR 2008. Proceedings*, volume 5195 of *Lecture Notes in Computer Science*, pages 292–298. Springer Berlin / Heidelberg, 2008.

First-Order Logic Concept Symmetry for Theory Formation

Pedro Torres[*] and Simon Colton
Computational Creativity Group
Department of Computing, Imperial College London, UK
ptorres,sgc@doc.ic.ac.uk

1 Introduction

SURICATA (Torres and Colton, 2008) is a hybrid automated theory formation system which uses both production rules and structured language biased search to produce new concepts and make conjectures about those concepts. The idea of implementing a highly-configurable theory formation system able to work with arbitrary first-order production rules started from the observation that it was helpful for users of automated theory formation systems to be able to define their own production rules. We implemented a generic first-order production rule (Torres and Colton, 2006) for the HR system (Colton, 2002) and showed how new user-defined production rules led to the discovery of novel conjectures in quasigroup theory.

During theory formation, SURICATA searches for regularities in newly formed concepts — defined as first-order logic (FOL) formulas — in order to make conjectures. Currently, these conjectures arise by comparing the available examples of concepts (see final part of section 2). To improve SURICATA's ability to make conjectures, we explore here a large set of new and relevant regularities which arise from intrinsic symmetries of the concepts under study. An example of concept symmetry is $\forall x, y, z.(\phi(x, y, z) \leftrightarrow \phi(y, x, z))$. The motivation for studying these regularities is two-fold: (i) the symmetry group of a set is traditionally a useful mathematical tool to capture the properties of a set and hence we hope that the characterisation of a concept through its symmetries will lead to improved ways to search for concepts; (ii) the symmetries studied here partially propagate through FOL formulas, and hence through FOL production rules, decreasing the time complexity of the search for symmetries in the space of concepts. Group theory has been previously applied to detecting structural symmetries of models of concurrent systems (Donaldson and Miller, 2005).

Given a set of concepts with some particular symmetries, new concepts built from them will present symmetries which we know to be true by construction, without having to actually prove them: they come for free. To the best of our knowledge, the exact way in which these symmetries propagate within first-order logic and which symmetries are in fact relevant for theory formation, has not been studied in detail. Our goal is to get a better understanding of how exactly symmetries propagate and how that can be exploited within theory formation. We present preliminary results here.

2 First-order Theory Formation Setting

A *concept* is a relation between any n types. We will consider here concepts of the form $\phi(x_1, \dots, x_n)$, where ϕ is some first-order logic formula with free variables $x_1, \dots, x_n$. We write $x_i : \tau_i$ to say that variable x_i *has type* τ_i and, if each x_i has type τ_i, we say that concept $\phi(x_1, \dots, x_n)$ has type $\tau_1 \times \dots \times \tau_n$. To make types explicit, we write $\phi(x_1 : \tau_1, \dots, x_n : \tau_n)$. We use the symbol $\mathcal{C}$ for the set of all concepts and $\mathcal{C}[\tau]$ for the set of all concepts of type τ.

A *production rule* is a function $\pi : \mathcal{D} \to \mathcal{C}$, where $\mathcal{D} \subseteq \mathcal{C}^q$, for some q. Suppose we have $\phi(x : \tau_1, y : \tau_2)$ and $\varphi(y : \tau_2, z : \tau_3)$. Then, for instance, we can define a production rule as: $(\phi(x, y), \varphi(y, z)) \mapsto \exists y.(\phi(x, y) \wedge \varphi(y, z))$. In this expression, ϕ and φ act as placeholders for concepts of the type described.

From an initial set of background concepts $\{\phi_1, \dots, \phi_k\}$, by recursively applying production rules with appropriate domains, we can define new concepts. Given a set of initial concepts $\mathcal{C}_0$ and a set of production rules Π, we recursively define $\Pi^*(\mathcal{C}_0)$ as the set of all concepts which: (i) are in $\mathcal{C}_0$ or (ii) can be written as $\pi(\phi_1, \dots, \phi_n)$ where π is in Π and every ϕ_i is in $\Pi^*(\mathcal{C}_0)$. The set $\Pi^*(\mathcal{C}_0)$ can be described as the set of all concepts that can be produced from the concepts in $\mathcal{C}_0$ by using the production rules in Π. If we fix $\mathcal{C}_0$, the set Π defines exactly which concepts are in the search space.

One way of making conjectures in a production rule based system is to compare the concepts produced. If at some point during theory formation, we have two concepts of the same arity and types for which all the available examples of one are also examples of the other and vice versa, it is possible to conjecture that they are equivalent. Similar situations may lead to implication and non-existence conjectures. The conjectures that such a system may come up with are therefore all of the form $[\forall \vec{x}.(\phi_1(\vec{x}) \to \phi_2(\vec{x}))]$, $[\forall \vec{x}.(\phi_1(\vec{x}) \leftrightarrow \phi_2(\vec{x}))]$ or $[\forall \vec{x}.\neg\phi_1(\vec{x})]$ where $\vec{x}$ is some set of variables and ϕ_1 and ϕ_2 are concepts in $\Pi^*(\mathcal{C})$.

[*]First author supported by Fundação para a Ciência e a Tecnologia, grant SFRH/BD/12437/2003.

3 Concept Symmetries

Types of symmetries We will assume here that concept variables all have the same type, τ. If a concept has variables with different types, we can always consider the symmetries we are about to describe restricted to those variables of the same type. Symmetries in concepts can take several forms. We will consider the symmetries defined below.

1. Let $\mathcal{P}_n$ be the set of all permutations of the set $\{1, \ldots, n\}$ and let $\sigma \in \mathcal{P}_n$. Given an n-tuple of variables $\vec{x}$, we write $\sigma\vec{x}$ to denote the tuple of variables obtained by reordering $\vec{x}$ according to σ. A concept $\phi(\vec{x})$ of arity n has *variable symmetry* if there exists $\sigma \in \mathcal{P}_n$ such that $\forall\vec{x}.\ (\phi(\vec{x}) \leftrightarrow \phi(\sigma\vec{x}))$. E. g. $\forall x, y, z.(\phi(x, y, z) \leftrightarrow \phi(y, z, x))$.

2. A concept $\phi(\vec{x})$ has *variable collapse symmetry* if there exists a set of substitutions of each variable x_i by another variable x_j (i and j may be equal) such that $\forall\vec{x}.\ (\phi(S\vec{x}))$ is true, where S acts on $\vec{x}$ to perform the mentioned set of variable substitutions. E. g. $\forall x, y.(\phi(x, x, y))$ is true.

3. A concept $\phi(\vec{x})$ is said to have *ground symbol symmetry* if there exists a bijective function $f : \tau \to \tau$ such that $\forall\vec{x}.(\phi(\vec{x}) \leftrightarrow \phi(\hat{f}\vec{x}))$, where $\hat{f}$ is a function which acts as f on one of the components of $\vec{x}$ and leaves all other components unchanged E. g. $\forall x, y.(\phi(x, y) \leftrightarrow \phi(fx, y))$, with $\tau = \mathbb{Z}$ and $fx = -x$.

4. Let τ be a type and define a group $G = (\tau, *)$. We define the (left) g-action of G on itself as the function $\alpha_g : G \to G$ such that $\alpha_g(x) = g * x$. A concept $\phi(\vec{x})$ has *G-action symmetry* if $\forall g, \vec{x}.(\phi(\vec{x}) \leftrightarrow \phi(\hat{\alpha}_g\vec{x}))$, for some $\hat{\alpha}_g$, where $\hat{\alpha}_g$ is a function which acts as α_g on one of the components of $\vec{x}$ and leaves all other components unchanged. E. g. $\forall g, x, y.(\phi(x, y) \leftrightarrow \phi(x, g * y))$. Group action symmetries can be generalised to arbitrary algebraic structures.

For each symmetry on the examples, we can write a corresponding conjecture. Searching for symmetries corresponds to having additional sophisticated conjecture making techniques which do more than mere testing sets of examples for equality or inclusion. Each of these techniques actually searches for intricate patterns in the examples which would take various steps to find in the normal theory formation setting of "concept formation followed by elementary conjecture making techniques", if at all reachable.

Symmetry Propagation Suppose that concepts ϕ_1 and ϕ_2 satisfy, $\forall x, y.(\phi_1(x, y) \leftrightarrow \phi_1(y, x))$ and $\forall x, y.(\phi_2(x, y) \leftrightarrow \phi_2(y, x))$. Then, we know that any new concept formed from these two concepts will have some kind of symmetry (not necessarily the same). For example, concept ϕ defined as $\phi_1(x, y) \wedge \phi_2(x, y)$ would still preserve the same symmetry, i.e. $\phi(x, y) \leftrightarrow \phi(y, x)$, for all x and y. Moreover, more intricate concepts such as $\phi_1(x, y) \wedge \exists z.\ (\phi_3(z) \wedge \phi_2(y, x))$ where ϕ_3 is some arity 1 concept, will still preserve the same symmetry. In the general case, symmetry propagation is more involved as variables from different concepts can interact in non-trivial ways. Our main observation is that, since in an automated theory formation system all concepts are produced using logical production rules, if we know how symmetries propagate through FOL basic connectives, we will know symmetries of concepts in the theory formed without having to prove them. They will be true by construction. Knowing symmetries of initial concepts allows us to know that certain symmetries will be present in the final concepts. It does not, however, give us a full account of all the symmetries of the final concepts. To know the remaining symmetries of a concept we will still have to compute them. Nevertheless, if we take symmetry propagation into account, we need to check for fewer symmetries in total to get the complete picture.

Current work We are currently implementing a SURICATA module which computes all the symmetries of a concept and which, given some initial concepts and a set of production rules, computes the symmetries of all producible concepts making use of symmetry propagation knowledge. We hope to show that the use of this knowledge makes the computation of symmetries more time efficient. Using this new symmetry module we will be able to perform theory formation using a greedy search on the number of symmetries of concepts and, also, to search for concepts which present particular bespoke symmetries, which is a novel paradigm in theory formation.

References

S. Colton. *Automated Theory Formation in Pure Mathematics*. Springer-Verlag, 2002.

A F Donaldson and A Miller. Automatic symmetry detection for model checking using computational group theory. *Lecture Notes in Computer Science*, 3582:481–496, 2005.

P Torres and S Colton. Using Model Generation in Automated Concept Formation. In *Proceedings of ARW'06*, 2006.

P Torres and S Colton. Automated Meta-Theory Induction in Pure Mathematics. In *Proceedings of ARW'08*, 2008.

Model-Checking Auctions, Coalitions and Trust[1]

Matt Webster[⋆] Louise Dennis[⋆] Michael Fisher[⋆]
[⋆]Department of Computer Science, University of Liverpool, Liverpool, L69 3BX, UK
{matt,louised,mfisher}@liverpool.ac.uk

Abstract

In this paper we tackle the verification of auction models that are at the heart of many market-based multi-agent systems. Specifically, we program auctions with auctions and trust in a BDI-based programming language and then use agent model checking to verify logical properties concerning beliefs within the multi-agent system.

1 Auctions and Verification in Multi-Agent Systems

The basic idea of an *auction* (Klemperer, 2004) is at the heart of many multi-agent scenarios. Not only are auctions central to e-commerce applications, but are implicit within many *market-based* approaches to agent computation. These include resource-allocation, telecommunications, electricity supply management, agent mobility, task allocation and scheduling. However, although much work has been carried out on the deep analysis of auction mechanisms, such as through formal mechanism design (Wooldridge et al., 2007), the analysis of *implementations* of auction mechanisms has lagged behind. While there has been *some* work on the formal verification of auction implementations, such as (Doghri, 2008), this has been lacking an agent perspective. Thus, the more sophisticated agent aspects such as *goals*, *intentions*, *beliefs* and *deliberation* have not been verified within an auction context. It is this we aim to tackle in this paper.

We have been developing techniques for verifying multi-agent systems over a number of years. The agents involved are rational (and, so, are capable of "intelligent" autonomous behaviour) and are represented in a high-level language describing their beliefs, intentions, etc. In particular, we have previously looked at the verification of BDI languages such as GWENDOLEN, and both homogeneous and heterogeneous multi-agent systems (Dennis and Fisher, 2008). In order to carry out formal verification, formal descriptions of both the system being considered and the properties to be verified are required. Producing formal descriptions for multi-agent systems is well understood, with the BDI approach to rational agency being particularly popular (Rao and Georgeff, 1995). Our approach, based on *model checking*, is outlined in (Bordini et al., 2008; Webster et al., 2009), and is available from `http://sourceforge.net/projects/mcapl`.

A multi-agent program, originally programmed in some agent programming language is executed in an interpreter via an interface to the *Agent Infrastructure Layer (AIL)*. Each agent uses AIL data structures to store its internal state comprising, for instance, a belief base, a plan library, a current intention, and a set of further intentions (as well as other temporary state information). They also use the specific interpreter for the agent programming language built using AIL classes and methods. Therefore, the AIL streamlines the development and verification of heterogenous multi-agent systems. The interpreter defines the reasoning cycle for the agent programming language which interacts with the model checker, essentially notifying it when a new state is reached that is relevant for verification. This allows the model checker, AJPF (*Agent JPF*), to create a Java product automata from the program and a property defined in the AJPF property specification language.

The product automata runs in the JPF virtual machine (see `http://javapathfinder.sourceforge.net`). This is a Java virtual machine specially designed to maintain backtrack points and explore, for instance, all possible thread scheduling options (that can affect the result of the verification). The JPF model checker is extensible and configurable, which allows us to optimise its performance for AIL-based systems, for instance AJPF has a specialised *Property Listener* which terminates execution of the product automata if it detects that the property has been, and will remain, satisfied for the rest of the run.

1.1 Example: Auction with Coalitions and Trust

A basic sealed-bid English auction scenario was extended to include the possibility of *coalitions* (Sandholm and Lesser, 1997). In our model, a coalition is when several agents collaborate by pooling their bid resources in order to win the auction. For example, if three agents x, y, z bid 100, 150 and 200 respectively, then z ought to win every time. However, if x and y form a coalition, their collective bid of 250 will be enough to win the auction.

[1]Work funded by EPSRC project "Model Checking Agent Programming Languages" (EP/D052548).

Our simple coalition scenario consists of five agents: one auctioneer, four bidders. At the start of the auction all bidders submit a bid and the auctioneer announces the winner. One of the agents is a coalition-forming agent: if it loses the first auction, it then tries to form a coalition with an agent it trusts. After forming a coalition the agent bids again. Then, if its coalition is successful in winning the auction, it stops. However, if its coalition is unsuccessful then it no longer believes that it can trust the other agent in the coalition, and will try to form another coalition with another agent it trusts.

For illustrative purposes a specification for the coalition-forming agent is laid out below in the GWENDOLEN agent programming language; the complete specification is given in (Webster et al., 2009).

AGENT: `ag2`
Initial Beliefs: $my\_name(ag2), trust(ag4), trust(ag5)$
Initial Goals: $+!_p bid$
Plans:

$$\downarrow^{Ag} \mathbf{tell}(B) : \top \texttt{ <- } +B$$

$$+!_p bid : my\_name(Name) \land \neg\uparrow^{ag1} \mathbf{tell}(bid(150, Name)) \texttt{ <- } \uparrow^{ag1} \mathbf{tell}(bid(150, Name))$$

$$+win(A) : \begin{bmatrix} my\_name(Name) \land \neg win(Name) \land trust(Ag) \\ \land\neg formed\_coalition(Ag') \land \neg\uparrow^{Ag} \mathbf{tell}(coalition(Name)) \end{bmatrix} \texttt{ <- } \begin{bmatrix} \uparrow^{Ag} \mathbf{tell}(coalition(Ag)); \\ +formed\_coalition(Ag) \end{bmatrix}$$

$$+win(A) : \begin{bmatrix} my\_name(Name) \land \neg win(Name) \land trust(Ag) \\ \land formed\_coalition(Ag') \land \neg\uparrow^{Ag} \mathbf{tell}(coalition(Name)) \end{bmatrix} \texttt{ <- } \begin{bmatrix} \uparrow^{Ag} \mathbf{tell}(coalition(Ag)); \\ +formed\_coalition(Ag); \\ -trust(Ag') \end{bmatrix}$$

$$+agree(A, X) : \top \texttt{ <- } \uparrow^{ag1} \mathbf{tell}(bid(150 + X, ag2))$$

The GWENDOLEN agent specification was converted automatically into a lower-level Java/AIL program and verified that $\Diamond \mathsf{B}(a_1, win)$ where a_1 forms a coalition with one of the trusted agents after losing the auction to a_2, i.e. eventually the agent believes that a_1 will win. If it chooses a_4 first, it wins and stops. If it chooses a_3 first, it loses the auction and distrusts that agent, trying subsequently with a_4. It then wins the auction. Verification took place in 20m 30s using 22 MB on a Windows XP PC with an Intel Core 2 Duo (E6750 @ 2.66GHz) CPU and 2 GB of RAM.

2 Concluding Remarks

In this paper we have discussed the verification by model-checking of agent-based auction software. We have shown that it is a realistic proposition to model-check the properties of interesting multi-agent implementations within a reasonable time. For example, as we reach the situation where agents form coalitions together, based on a dynamic notion of trust, in order to compete with other agents, then we are not far from realistic agent systems. Clearly, for bigger scenarios improved efficiency will be required (and, indeed, this is something we are actively working on), but the examples implemented and verified in this paper show that small, but non-trivial, market-based multi-agent systems can be automatically verified.

References

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.

L. A. Dennis and M. Fisher. Programming Verifiable Heterogeneous Agent Systems. In Koen Hindriks, Alexander Pokahr, and Sebastian Sardina, editors, *Sixth International Workshop on Programming in Multi-Agent Systems (ProMAS'08)*, Estoril, Portugal, May 2008.

Ines Doghri. Formal verification of WAHS: an autonomous and wireless P2P auction handling system. In *Proc. 8th International Conference on New Technologies in Distributed Systems (NOTERE)*, pages 1–10, New York, NY, USA, 2008. ACM. doi: http://doi.acm.org/10.1145/1416729.1416754.

Paul Klemperer. *Auctions: Theory and Practice*. Princeton University Press, Princeton, USA, 2004.

A. S. Rao and M. Georgeff. BDI Agents: From Theory to Practice. In *Proceedings of the First International Conference on Multi-Agent Systems (ICMAS)*, pages 312–319, San Francisco, CA, June 1995.

Tuomas Sandholm and Victor R. Lesser. Coalitions Among Computationally Bounded Agents. *Artificial Intelligence*, 94(1-2), 1997.

Matt Webster, Louise Dennis, and Michael Fisher. Model-Checking Auctions, Coalitions and Trust. Technical Report ULCS-09-004, Department of Computer Science, University of Liverpool, 2009. http://www.csc.liv.ac.uk/research/techreports/tr2009/ulcs-09-004.pdf.

Michael Wooldridge, Thomas Ågotnes, Paul E. Dunne, and Wiebe van der Hoek. Logic for Automated Mechanism Design - A Progress Report. In *Proceedings of the Twenty-Second AAAI Conference on Artificial Intelligence*, pages 9–. AAAI Press, 2007.

CTL-RP: A Computational Tree Logic Resolution Prover [*]

Lan Zhang[⋆] Ullrich Hustadt[⋆] Clare Dixon[⋆]
[⋆] Department of Computer Science, University of Liverpool
Liverpool, L69 3BX, UK
{Lan.Zhang,U.Hustadt,CLDixon}@liverpool.ac.uk

1 Introduction

Temporal logic is considered an important tool in many different areas of Artificial Intelligence and Computer Science, including the specification and verification of concurrent and distributed systems. Computational Tree Logic CTL (Clarke and Emerson, 1982) is a branching-time temporal logic. Here we present the first resolution theorem prover for CTL, CTL-RP, which implements the sound and complete clausal resolution calculus $\mathsf{R}^{\succ,S}_{\mathrm{CTL}}$ (Zhang et al., 2008) based on an earlier calculus by Bolotov (2000). The calculus $\mathsf{R}^{\succ,S}_{\mathrm{CTL}}$ is designed in order to allow the use of classical first-order resolution techniques to emulate the rules of the calculus. We take advantage of this approach in the development of our prover CTL-RP which uses the first-order theorem prover SPASS (Weidenbach et al., 2007).

2 Normal form for CTL $\mathrm{SNF}^{\mathrm{g}}_{\mathrm{CTL}}$ and clausal resolution calculus $\mathsf{R}^{\succ,S}_{\mathrm{CTL}}$

The calculus $\mathsf{R}^{\succ,S}_{\mathrm{CTL}}$ operates on formulae in a clausal normal form called Separated Normal Form with Global Clauses for CTL, denoted by $\mathrm{SNF}^{\mathrm{g}}_{\mathrm{CTL}}$. The language of $\mathrm{SNF}^{\mathrm{g}}_{\mathrm{CTL}}$ clauses is defined over an extension of CTL in which we label certain formulae with an index ind taken from a countably infinite index set Ind and it consists of formulae of the following form.

$\mathbf{A}\Box(\mathbf{start} \Rightarrow \bigvee_{j=1}^{k} m_j)$	(initial clause)	$\mathbf{A}\Box(\bigwedge_{i=1}^{n} l_i \Rightarrow \mathbf{E}\bigcirc \bigvee_{j=1}^{k} m_{j\,\langle ind\rangle})$	(**E**-step clause)
$\mathbf{A}\Box(\mathbf{true} \Rightarrow \bigvee_{j=1}^{k} m_j)$	(global clause)	$\mathbf{A}\Box(\bigwedge_{i=1}^{n} l_i \Rightarrow \mathbf{A}\Diamond l)$	(**A**-sometime clause)
$\mathbf{A}\Box(\bigwedge_{i=1}^{n} l_i \Rightarrow \mathbf{A}\bigcirc \bigvee_{j=1}^{k} m_j)$	(**A**-step clause)	$\mathbf{A}\Box(\bigwedge_{i=1}^{n} l_i \Rightarrow \mathbf{E}\Diamond l_{\langle LC(ind)\rangle})$	(**E**-sometime clause)

where **start** is a propositional constant, l_i $(1 \leq i \leq n)$, m_j $(1 \leq j \leq k)$ and l are literals, that is atomic propositions or their negation, ind is an element of Ind. The symbols ind and $LC(ind)$ represent indices and limit closure of indices, respectively. As all clauses are of the form $\mathbf{A}\Box(P \Rightarrow D)$ we often simply write $P \Rightarrow D$ instead.

We have defined a set of transformation rules which allows us to transform an arbitrary CTL formula into an equi-satisfiable set of $\mathrm{SNF}^{\mathrm{g}}_{\mathrm{CTL}}$ clauses, a complete description of which can be found in (Zhang et al., 2008). The transformation rules are similar to those in (Bolotov, 2000), but modified to allow for global clauses.

$\mathsf{R}^{\succ,S}_{\mathrm{CTL}}$ consists of two types of resolution rules, *step* resolution rules (SRES1 to SRES8) and *eventuality* resolution rules (ERES1 and ERES2). Motivated by refinements of propositional and first-order resolution, we restrict the applicability of step resolution rules by means of an atom ordering $\succ$ and a selection function S, which helps to prune the search space dramatically. Due to lack of space, we only present two of the step resolution rules and one of the eventuality resolution rules. In the following l is a literal, P and Q are conjunctions of literals, and C and D are disjunctions of literals.

SRES2 $$\frac{P \Rightarrow \mathbf{E}\bigcirc(C \vee l)_{\langle ind\rangle},\ Q \Rightarrow \mathbf{A}\bigcirc(D \vee \neg l)}{P \wedge Q \Rightarrow \mathbf{E}\bigcirc(C \vee D)_{\langle ind\rangle}}$$

SRES3 $$\frac{P \Rightarrow \mathbf{E}\bigcirc(C \vee l)_{\langle ind\rangle},\ Q \Rightarrow \mathbf{E}\bigcirc(D \vee \neg l)_{\langle ind\rangle}}{P \wedge Q \Rightarrow \mathbf{E}\bigcirc(C \vee D)_{\langle ind\rangle}}$$

ERES1 $$\frac{P^{\dagger} \Rightarrow \mathbf{E}\bigcirc\mathbf{E}\Box l,\ Q \Rightarrow \mathbf{A}\Diamond\neg l}{Q \Rightarrow \mathbf{A}(\neg P^{\dagger}\,\mathcal{W}\,\neg l)}$$

where $P^{\dagger} \Rightarrow \mathbf{E}\bigcirc\mathbf{E}\Box l$ represents a set of $\mathrm{SNF}^{\mathrm{g}}_{\mathrm{CTL}}$ clauses which together imply $P^{\dagger} \Rightarrow \mathbf{E}\bigcirc\mathbf{E}\Box l$.

We develop a new completeness proof with a different approach from (Bolotov, 2000). The proof also shows that some eventuality resolution rules in (Bolotov, 2000), which are the most costly rules of the calculus, are redundant. The inference rules of $\mathsf{R}^{\succ,S}_{\mathrm{CTL}}$ can be used to decide the satisfiability of a given set N of $\mathrm{SNF}^{\mathrm{g}}_{\mathrm{CTL}}$ clauses by computing the saturation N' of N using at most an exponential number of inference steps; N is unsatisfiable iff N' contains a clause $\mathbf{true} \Rightarrow \mathbf{false}$ or $\mathbf{start} \Rightarrow \mathbf{false}$. This gives a complexity optimal EXPTIME decision procedure for CTL.

[*]This work is supported by EPSRC grant EP/D060451/1.

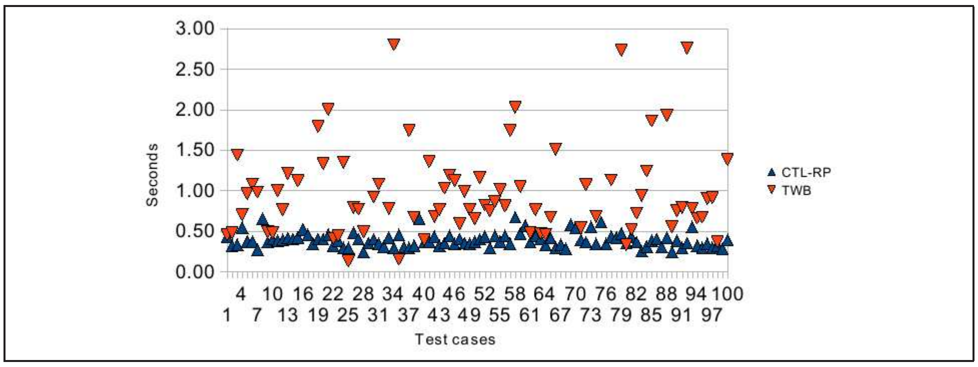


Figure 1: Performance on a set of benchmark formulae

3 CTL-RP

In order to obtain an efficient CTL theorem prover and to reuse existing state-of-the-art first-order resolution theorem provers, we adopt an approach analogous to that used in (Hustadt and Konev, 2004) to implement a resolution calculus for PLTL to implement the calculus $\mathsf{R}^{\succ,S}_{\mathrm{CTL}}$ and the associated decision procedure for CTL. A formal description of the approach and related proofs are presented in detail in (Zhang et al., 2008).

In our implementation of $\mathsf{R}^{\succ,S}_{\mathrm{CTL}}$, we first transform all $\mathrm{SNF}^{\mathrm{g}}_{\mathrm{CTL}}$ clauses except **A**- and **E**-sometime clauses into first-order clauses. Then we are able to use first-order ordered resolution with selection to emulate step resolution. For this part of the implementation we are using the theorem prover SPASS. **A**- and **E**-sometime clauses cannot be translated to first-order logic. Therefore, we continue to use the eventuality resolution rules ERES1 and ERES2 for inferences with **A**- and **E**-sometime clauses, respectively, and use the loop search algorithm presented in (Bolotov and Dixon, 2000) to find suitable premises for these rules. We utilise first-order ordered resolution with selection to perform the most costly task of "looking for merged clauses" in the loop search algorithm and we compute the results of applications of the eventuality resolution rules in the form of first-order clauses.

Besides CTL-RP, there is only one other CTL theorem prover we know of, namely a CTL module for the Tableau Workbench (TWB) (Abate and Goré, 2003). We have created several sets of benchmark formulae that we have used to compare CTL-RP version 00.09 with TWB version 3.4. The comparison was performed on a Linux PC with an Intel Core 2 CPU@2.13 GHz and 3G main memory, using the Fedora 9 operating system. In Figure 1, we show the experimental results on one of those sets of benchmark formulae. This set of benchmark formulae consists of one hundred formulae such that each formula specifies a randomly generated state transition system. The graph in Figure 1 indicates the CPU time in seconds required by TWB and CTL-RP to establish the satisfiability or unsatisfiability of each benchmark formula in the set of benchmark formulae. CTL-RP shows a much more stable performance on these benchmarks than TWB.

References

P. Abate and R. Goré. The Tableaux Workbench. In *Proc. TABLEAUX'03*, pages 230–236. Springer, 2003.

A. Bolotov. *Clausal Resolution for Branching-Time Temporal Logic*. PhD thesis, Manchester Metropolitan University, 2000.

A. Bolotov and C. Dixon. Resolution for Branching Time Temporal Logics: Applying the Temporal Resolution Rule. In *Proc. TIME'00*, pages 163–172. IEEE, 2000.

E. M. Clarke and E. A. Emerson. Design and Synthesis of Synchronization Skeletons Using Branching-Time Temporal Logic. In *Logic of Programs, Workshop*, volume 131 of *LNCS*, pages 52–71. Springer, 1982.

U. Hustadt and B. Konev. TRP++: A Temporal Resolution Prover. In *Collegium Logicum*, pages 65–79. Kurt Gödel Society, 2004.

C. Weidenbach, R. A. Schmidt, T. Hillenbrand, R. Rusev, and D. Topic. System description: Spass version 3.0. In *Proc. CADE-21*, volume 4603 of *LNCS*, pages 514–520, 2007.

L. Zhang, U. Hustadt, and C. Dixon. First-order Resolution for CTL. Technical Report ULCS-08-010, Department of Computer Science, University of Liverpool, 2008.