Proceedings of the

Automated Reasoning Workshop 2008

Edited by L. A. Dennis and V. Sorge
30th-31st July, Birmingham, UK

Proceedings of the Automated Reasoning Workshop 2008

L. A. Dennis and V. Sorge

Contents

Invited Talk: Cooperative Reasoning for Automatic Software Verification

Andrew Ireland

Dependable Systems Group, School of Mathematical and Computer Sciences, Heriot-Watt University

The proliferation of software across all aspects of modern life means that software failures can have significant economic, as well as social impact. The goal of being able to develop software that can be formally verified as correct with respect to its intended behaviour is therefore highly desirable. The foundations of such formal verification have a long and distinguished history, dating back over fifty years. What has remained more elusive are scalable verification tools that can deal with the complexities of software systems. However, times are changing, as reflected by a current renaissance within the formal software verification community. My talk will focus on automated reasoning aspects of software verification. In particular, I will argue for the benefits that can be achieved if complementary reasoning techniques are combined cooperatively. As evidence for my argument I will draw upon a past industrial oriented project as well as some new work on reasoning about pointer programs.

Verification of resource-bounded multi-agent systems

Natasha Alechina *Brian Logan* *Nguyen Hoang Nga* *Abdur Rakib*
School of Computer Science
University of Nottingham
{nza,bsl,hnn,rza}@cs.nott.ac.uk

Abstract

In this extended abstract we would like to present some key ideas of our current research on investigating resource requirements for systems of distributed reasoning agents. We are interested in automated verification of properties of such systems, in particular in trade-offs between different resources (time, memory and communication bandwidth). For verification, we use standard model checking techniques.

1 Introduction

We consider systems of reasoning agents which attempt to solve a problem (produce a derivation of the goal formula) and investigate the minimal resource requirements for solving a particular problem. The resources are: time (how many inference steps does the system need to perform, in parallel), memory (how much memory do the agents need to store derived facts) and communication bandwidth (how many messages do the agents need to exchange).

There has been considerable work in the agent literature on distributed problem solving in general and on distributed reasoning in particular [6; 1; 5]. Much of this work analyses the time and communication complexity of distributed reasoning algorithms. However, while we have upper (and some lower) bounds on time and memory requirements for reasoning in distributed systems, we lack tools for reasoning about trade-offs between computational (time, memory) and communication resources in systems of reasonings agents. Our research intends to provide a framework for reasoning about resource requirements and trade-offs between different resources.

2 Proposed framework

We consider agents with different reasoning capabilities (e.g., agents reasoning in propositional logic using resolution, or rule-based agents). Each agent has a knowledge base KB which contains formulas (e.g. clauses for a resolution agent). The actions each agent can perform are:

- Read a clause from its knowledge base into its working memory
- Perform an inference step using formulas in its memory as premises (e.g. resolve two clauses in its memory)
- Copy a formula which is in another agent's memory
- Idle

A 'step' of the whole system is composed of the actions of each agent, in parallel. We measure time requirements for a problem as the number of such system steps.

Adding a new formula to the working memory may result in overwriting some formula which is already in memory. Each agent's memory usage is modelled as the maximal number of formulas in the agent's memory at any given time.

Copy action corresponds to communication; it is a very abstract model of communication which is always guaranteed to succeed and takes one tick of time. To model communication bounds, we introduce message counters for each agent which are incremented with each Copy action.

In the distributed setting we distinguish between symmetric problem distributions, where all agents have the same premises and the same rules of inference, and asymmetric problem distributions where different premises may be assigned to different agents and/or the agents use different rules of inference. Similarly, we can distinguish between symmetric resource distributions: when all agents have the same resource bounds and asymmetric resource distributions: when different agents have different resource bounds.

In [2; 3] we consider two typical distributed reasoning problems.

The first class of problems is finding a resolution derivation of an empty clause from the set of all possible clauses of the form $\sim A_1 \vee \sim A_2 \vee \ldots \vee \sim A_n$ where, for each $\sim A_i$ is either A_i or $\neg A_i$. We study instances of this problem for both symmetric and asymmetric distribution of clauses and resources, and find some interesting trade-offs.

The second class of problems corresponds to a system of two rule-based agents which share the same set of rules:

RuleB1 : $A_1, A_2 \rightarrow B_1$ **RuleC1 :**$B_1, B_2 \rightarrow C_1$
RuleB2 : $A_3, A_4 \rightarrow B_2$ **RuleC2 :** $B_3, B_4 \rightarrow C_2$ **RuleD1 :** $C_1, C_2 \rightarrow D_1$
RuleB3 : $A_5, A_6 \rightarrow B_3$
RuleB4 : $A_7, A_8 \rightarrow B_4$

These rules correspond to a binary tree, where the leaves are $A_1, A_2, \ldots, A_8$ which represent facts in the working memory and the goal is to derive D_1 the root of the tree. We consider various distributions of leaves between the agents. This synthetic abstract problem is chosen because we can easily increase or decrease the problem size, since it is parameterised by the number of leaves.

3 Model-checking encoding

We encode the systems of distributed reasoners using a standard model checker, and verify resource bounds using existing model checking techniques. For the examples reported here, we have used the Mocha model checker [4], due to the ease with which we can specify concurrently executing agents in *reactive modules*, the description language used by Mocha. The actions of resolving a clause or firing a rule or copying a fact from another agent and idling are encoded as a Mocha *atoms* which describe the initial condition and transition relation for a group of related state variables.

We have considered several alternative model-checking encodings. Given a set of agents with combined knowledge bases KB and a goal formula ϕ, we can determine the set of formula Ω which may possibly feature in a derivation. To represent agent states, we either need to say, for each formula in Ω, whether it is in agent i's memory in this state, or to say which formulas are stored in one of a fixed number of 'locations' in agent i's memory. Transitions of the system correspond to applications of inference rules of each agent to formulas which are in agent's memory in the state. The constraint on memory corresponds to a restriction that at most n_M formulas can be present in agent's memory in any state. The communication constraint can be easily expressed using a communication counter for each agent, which increases with each Copy action performed. The specification language of Mocha is ATL, which includes the branching time logic CTL. We can express properties such as "agent i may derive formula ϕ in n_T steps while exchanging fewer than n_C messages". To obtain the actual derivation we can verify the negation of the property, which returns a counterexample trace showing how the system reaches the state where ϕ is proved.

We have presented some experimental results in [2; 3], which show resource requirements for optimal derivation of the goal formula and interesting trade-offs between computational (time, memory) and communication resources.

Acknowledgements: This work was supported by the Engineering and Physical Sciences Research Council [grant number EP/E031226].

References

[1] P. Adjiman, P. Chatalic, F. Goasdoué, M.-C. Rousset, and L. Simon. Distributed reasoning in a peer-to-peer setting. In *Proceedings of the 16th European Conference on Artificial Intelligence, ECAI'2004*, pages 945–946. IOS Press, 2004.

[2] N. Alechina, B. Logan, H. N. Nguyen, and A. Rakib. Verifying time, memory and communication bounds in systems of reasoning agents. In *Proceedings of the Seventh International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2008)*. ACM Press to appear, 2008.

[3] N. Alechina, B. Logan, H. N. Nguyen, and A. Rakib. Verifying time and communication costs of rule-based reasoners. *Electronic Notes in Theoretical Computer Science to appear*, 2008.

[4] R. Alur, T. A. Henzinger, F. Y. C. Mang, S. Qadeer, S. K. Rajamani, and S. Tasiran. MOCHA: Modularity in model checking. In *Proceedings of the Computer Aided Verification*, volume 1427 of LNCS, pages 521–525. Springer Berlin / Heidelberg, 1998.

[5] E. Amir and S.A. McIlraith. Partition-based logical reasoning for first-order and propositional theories. *Artificial Intelligence*, 162(1-2):49–88, 2005.

[6] Michael Fisher and Michael Wooldridge. Distributed problem-solving as concurrent theorem proving. In *MAAMAW 1997*, volume 1237 of *Lecture Notes in Computer Science*, pages 128–140. Springer, 1997.

Automata-based Formal Specification of Stateful Systems

Alessandro Basso, Alexander Bolotov, Vladimir Getov
School of Computer Science, University of Westminster
Watford Road, Harrow HA1 3TP
bassoa@wmin.ac.uk,bolotoa@wmin.ac.uk,v.s.getov@wmin.ac.uk

Abstract

In light of our previous research[1] on runtime re-configuration of a component model (Grid Component Model)[1], we have been considering stateful systems, bringing new constraints into consideration. We describe in this short document how an automata-based system is used to formally specify a stateful system at two levels: the components' level, and the infrastructure level.

1 Introduction

The CoreGrids's Grid Component model (GCM) [7] is an extension of Fractal [6], a modular and extensible component model, built to accommodate requirements in distributed systems. The GCM specification defines a set of notions characterizing this model, an API (Application Program Interface), and an ADL (Architecture Description Language) [2]. Component models enable modular design of software applications that can be easily reused and combined, ensuring greater reliability. This is important in distributed systems where asynchronous components must be taken into consideration. In these models, components interact together by being bound through interfaces. The recent development of a GRID Integrated Development Environment (GIDE) based on the GCM specification [8] opens new possibilities for dynamic reconfiguration scenarios. We are able to take advantage of pre-built features in the GIDE to form a basis for a reconfiguration framework which exploits the underlying properties of the specification language and deductive reasoning verification methods used in our research into safe, automated dynamic reconfiguration. It is essential that these properties (starting with the components stateful properties) can be monitored and reported in order to be able to comprehend the current states scenario. We have considered the fact that monitoring a program with respect to a temporal formula can have an impact on the monitored program, with 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' specification - the behaviour of the stateful system - and leave the proofs of the inner components' behaviour to other methods [3]. In this paper we describe the concept of behaviour of stateful component systems and give an outline of the work in progress into a two layers automata for the formal specification of such a system.

2 Behaviour of stateful component systems

In the classical approach to behaviour specification, the term "behaviour" is referred to the inner component's functionality. When we consider the state behaviour of a component instead, we are looking for those requirements that will guarantee that the components behaviour agrees with the rest of the system. In a small system this could be trivial, but what about a distributed system where changes might be needed at runtime such as replacing a component which depends on a resource that is not available? We consider past developments within the GCM and other state aware grid systems [9] to define a set of states to be generated and monitored by specific software [7]. This lifecycle is restricted, in fact it only models the deployment state of the system, not its operational characteristics. For example once a component is in running state, it is considered available, but the service may fail for other unforeseen circumstances. We can represent these lifecycle states as in Figure 1. In GCM, the life-cycle controller allows to separate partially the states of the controller and of the content. When a component is created, it is

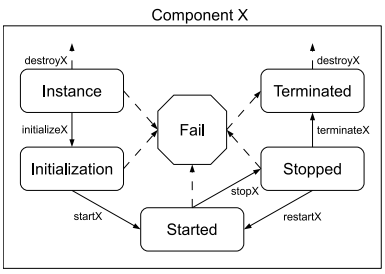


Figure 1: Lifecycle States

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

firstly instantiated and any operations (such as dependencies checks) are processed. The started state is indicative for the services required/provided by the component/resources that are being deployed and available for use; it does not refer to the internal operations of the component, but only the resources being used. If a component would reach the failure state, it may still be available to the system (although its required resources would become available), and the services provided by the component would be terminated. Due to the hierarchical composition of components, we can plot recursively resources required and services provided in the behavioural states system. When a component is functionally stopped, invocation on controller interfaces are enabled and the content of the component can be reconfigured, after which the Fractal specification allows for the component to be restarted by using the start call on the component again.

3 Automata-based Model

In building the specification protocol, we plan to follow well known automata constructions. We take a simple finite state automaton on finite strings, for the components specification, and a more complex finite state automata on infinite trees to define the environment. The former is used for the creation of labels defining the various states in which the considered component is, and are then fed upon request on to the various states of the latter automata. In the construction of this tree automaton, every state is labelled according to state of components (passed over from the component level automaton) and resources. The transition function is not only related to the state transition of components, but also tightly bound to the deontic logic accessibility relation. We expect to be able to specify the automaton in the normal form for CTL, $\mathrm{SNF}_{\mathrm{CTL}}$, developed in [5]. In [4], it was shown that a Buchi word automaton can be represented in terms of $\mathrm{SNF}_{\mathrm{PLTL}}$, a normal form for PLTL. We anticipate that in a similar fashion we can represent a Buchi tree automaton in terms of $\mathrm{SNF}_{\mathrm{CTL}}$; we use $\mathrm{SNF}_{\mathrm{CTL}}$to specify the tree automaton and then extend this specification to a deontic temporal specification [1] and apply a resolution-based verification technique as a verification procedure.

4 Conclusions

The need for a safe and reliable way to dynamically reconfigure systems at runtime, especially distributed, resource depending and long running, has led to the need for a formal way to describe and verify them before risking to take some action. Further research is been carried out in the following aspects: [1] We have assumed that the automaton on the bottom layer simply goes through the component's states (i.e. the states of components are the states of the automaton), but the specification here may result to be more complex. [2] We need to define the type of the tree automaton which would work on the environmental level and we need to specify how exactly the two automata interact, namely, how the bottom layer automaton feeds the environmental one (i.e. how this passing of labels is carried on).

References

[1] A. Basso, A. Bolotov and V. Getov. Behavioural Model of Component-based Grid Environments. Proc. CoreGRID Symposium, Springer, August 2008 (to appear).

[2] T. Barros, L. Henrio, A. Cansado, E. Madelaine, M. Morel, V. Mencl and F. Plasil. Extension of the Fractal ADL for the Specification of Behaviours of Distributed Components In Proc. Of 5th Fractal Workshop (part of ECOOP'06), July 3rd, 2006, Nantes, France, Jul 2006.

[3] T. Barros, L. Henrio and E. Madelaine. Verification of Distributed Hierarchical Components. In Proc. of the International Workshop on Formal Aspects of Component Software (FACS'05). Electronic Notes in Theor. Computer Sci. 160. pp. 41-55 (ENTCS), 2005.

[4] A. Bolotov, C. Dixon and M. Fisher. On the Relationship between Normal Form and W-automata. Journal of Logic and Computation, Volume 12, Issue 4, August 2002, pp. 561-581, Oxford University Press.

[5] 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.

[6] E. Bruneton, T. Coupaye, and J.B. Stefani. Recursive and dynamic software composition with sharing. In Seventh Int. Workshop on Component-Oriented Programming (WCOP02), at ECOOP 2002, Malaga, Spain, 2002.

[7] CoreGrid Programming Model Institute. Basic Features of the Grid Component Model Deliverable D.PM.04, CoreGRID, March 2007.

[8] A. Basukoski, V. Getov, J. Thiyagalingam, and S. Isaiadis. Component-oriented Development Environment for Grid: Design and Implementation. In Making Grids Work, Springer, 2008 (to appear).

[9] J. Tatemura. CDDLM Configuration Description Language Specification GFD-R-P.085, August 2006.

Analyzing the Behaviour of a Swarm of Foraging Robots Using Temporal Specification and Verification

Abdelkader Behdenna[*]*, Clare Dixon, and Michael Fisher*

Department of Computer Science, University of Liverpool, Liverpool, L69 3BZ, UK

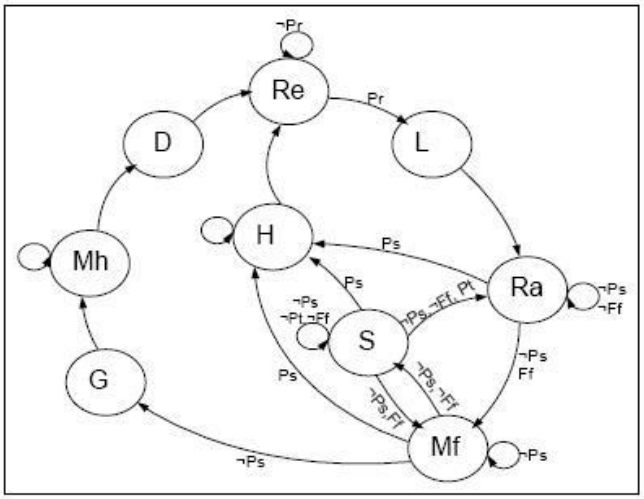
{A.Behdenna,CLDixon,MFisher}@liv.ac.uk

1 Introduction

Robot swarms are large numbers of simple robots, often with similar or identical behaviour. The swarm of robots work together to achieve some task and, as a swarm, are a much more fault tolerant system, as the failure of a small number of the robots should not affect the overall swarm behaviour. However, given the behaviour of the individual robots and the interaction and communication between each other it is not easy to predict the behaviour of the swarm as a whole to show the desired task can or will be achieved and that undesirable behaviour will not occur. In this work we consider the behaviour of swarm of foraging robots, as discussed in [5] and utilise temporal specification and verification. This work is based on previous temporal specifications of aspects of robotic swarms [6], but improves upon this by considering more sophisticated swarms and properties and by carrying out more comprehensive verification using deductive temporal techniques [2].

2 Formalising the Underlying Transition System

In [5] a description of a swarm of robots foraging for food in an arena is given. The behaviour of each robot is described using a transition system where states relate to the robot resting (Re), scanning the arena for food (S), performing a random move whilst searching for food (Ra), homing (H) etc (see below). Additionally there are guards on the edges of some of the transitions representing time-out conditions, for example, when the time for searching for food is over (Ps) the robot must start moving home towards the nest.



Re : Resting
L : Leaving home
Ra : Random move
Mf : Moving to food
S : Scanning
H : Homing
G : Grabbing
Mh : Moving to home
D : Depositing
Pr,Ps,Pt : Maximum resting/searching/scanning time is reached
Ff : Food found

Initially we represent this structure using propositional linear-time temporal logic. For example $\Box((S \wedge Ps) \Rightarrow \bigcirc H)$ represents that it is always the case that if the robot is scanning the arena and the time for searching the arena is finished then in the next moment the robot will be homing.

To ensure that each robot can not indefinitely spend time searching for food we added a number of constraints of the form $\Box\Diamond Ps$ denoting that infinitely often the searching time will have been used up. These constraints will be re-set every cycle through the system for example Ps is set to false once the robots leave the nest.

[*] Also at *Departement d'informatique de l'Ecole Normale Supérieure de Cachan, Cachan, France.*

3 Proving Properties of the Transition System

TRP++ [2] is a resolution-based theorem prover for Propositional Linear Time Temporal Logics. Using the above formalisation of the transition system and the constraints we have been able to show a wide variety of simple properties. For example, we have established $\Box\Diamond D$, meaning that a robot will always find and grab a piece of food if there is an accessible one.

The specification, together with the set of properties established, was then extended, to incorporate:

- multiple robots interacting together;
- multiple items of food that the robots are searching for;
- regenerative and non-regenerative items of food; and
- counters, specifically to allow real-time aspects to be verified.

A range of properties were verified, again using TRP++, though the time taken increases dramatically as the number of robots and items of food increase. However, the descriptions are very modular, allowing robots, items of food, and their interactions, to be modified with ease.

4 Conclusions

Temporal deductive tools have been used for the automated verification of simple properties of (small) swarm robotic systems. This work has used TRP++, an implementation of a temporal resolution system, and is based on models of swarm robot behaviour provided by leading robotics researchers.

Our current work is extending this approach in two ways:

1. Extension from fixed numbers of robots to *arbitrary* numbers of robots, by using *first-order temporal specifications* and associated verification tools [4; 3].
2. Extension to the representation of a probabilistic transition system, incorporating both verification via PRISM [1] and simulation using OCaml.

Acknowledgements We thank Alan Winfield and Jin Sa from the Bristol Robotics Lab for their comments on this work, and Ullrich Hustadt and Boris Konev for their support with the theorem provers TRP++ and TeMP.

References

[1] A. Hinton, M. Z. Kwiatkowska, G. Norman, and D. Parker. PRISM: A Tool for Automatic Verification of Probabilistic Systems. In *Proc. 12th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, volume 3920 of *LNCS*, pages 441–444. Springer, 2006. ISBN 3-540-33056-9.

[2] U. Hustadt and B. Konev. TRP++ 2.0: A temporal resolution prover. In *Proceedings of Conference on Automated Deduction (CADE-19)*, number 2741 in LNAI, pages 274–278. Springer, 2003.

[3] U. Hustadt, B. Konev, A. Riazanov, and A. Voronkov. TeMP: A Temporal Monodic Prover. In D. A. Basin and M. Rusinowitch, editors, *Proceedings of the Second International Joint Conference on Automated Reasoning (IJCAR 2004)*, volume 3097 of *LNAI*, pages 326–330. Springer, 2004. ISBN 3-540-22345-2.

[4] 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.

[5] W. Liu and A.F.T. Winfield and J. Sa and J. Chen and L. Dou. Strategies for energy optimisation in a swarm of foraging robots. In *AB'06 Swarm Robotics Workshop*, 2006.

[6] A. Winfield, J. Sa, M-C. Fernández Gago, C. Dixon, and M. Fisher. On Formal Specification of Emergent Behaviours in Swarm Robotic Systems. *International Journal of Advanced Robotic Systems*, 2(4):363–370, December 2005.

Applying Machine Learning to Heuristic Selection in an Automatic Theorem Prover

James P. Bridge[⋆] jpb65@cam.ac.uk
Lawrence C. Paulson[⋆] Larry.Paulson@cl.cam.ac.uk
Sean B. Holden[⋆] Sean.Holden@cl.cam.ac.uk

[⋆]*The University of Cambridge Computer Laboratory, William Gates Building*
15 JJ Thomson Avenue, Cambridge CB3 0FD, United Kingdom

Abstract

Machine learning has been applied to the automation of heuristic selection in an automated, first order logic theorem prover. Heuristic selection based on features of the conjecture to be proved and the associated axioms does better than any single heuristic. A comparison between static features, measured prior to the proof search process, and dynamic features that measure changes arising in the early stages of proof search has been made.

1 Introduction

The algorithms used in automated theorem provers involve many choices leading to a large number of possible heuristics. The best heuristic to use depends on the conjecture and axioms that together form the problem that is the subject of the proof search. There is no simple connection between the problem and the choice of best heuristic to use, and even experts rely on a mixture of experience, experimentation and intuition. The work described in this paper uses machine learning (Support Vector Machines) to find the connection between measured features of the problem and the choice of best heuristic to use.

Machine learning has been applied to automated theorem proving in a number of ways including the automatic selection of heuristics, for example [1], but previous schemes have taken a nearest example or a case-based reasoning approach with a restricted number of features being considered and without the use of modern machine learning techniques such as support vector machines.

2 Method

The E theorem prover [3] was used. This is an automated theorem prover for first order logic with equality.

The experiment consisted of using machine learning to create a black-box procedure relating easily measured structural features of a conjecture and axioms to a selection from a limited set of possible heuristics. The automated procedure was applied to the TPTP [1] library of problems and the results compared with each of the heuristics applied to the whole problem set. Additionally, results were produced for applying the best heuristic in each problem (i.e. an upper limit that would arise from a perfect heuristic selection process).

The experimental work was carried out in three phases. First an initial classification experiment was done to determine the feasibility of making useful predictions on the basis of features that may be measured on conjectures and axioms. Secondly classification experiments were done for each of six possible heuristic selection outcomes. Finally the results of the second phase were combined to create a practical heuristic selection scheme which was compared to the individual heuristics in terms of number of proofs found and time taken.

Two types of features were measured. Static features measured on the conjecture and axioms prior to any proof attempt and dynamic features which measure changes taking place in the early stages of the proof search. Examples of features are average clause length, depth and weight and the proportion of clause types such as Horn clauses and unit clauses. In the current work the names of functions and variables are assigned no significance. Additionally the features are local to each problem, that is no attempt was made to measure connections between different problems.

In the initial experiment the aim was to classify problems, on the basis of feature values, into easy (solvable within a reasonable time) and hard (no solution found within the time limit). The problems were all taken from the SET area of the TPTP library. SVMLite [2] was used to do the classification with a number of different kernel functions tried as well as variations of parameter values. A limited set of sixteen dynamic features were measured. A single, fixed heuristic was used.

[1] Thousands of Problems for Theorem Provers see www.TPTP.org

The results (see next section) were successful enough to continue to the second and third phases of the work.

In the second phase the work was extended to heuristic selection. A set of the five most useful heuristics from E's auto mode was used. The initial set of sixteen dynamic features was extended to thirty nine and in addition fourteen static features were measured. The two were also combined to give a third set of fifty three features. One aim of the work was to determine if there was a significant difference in applying machine learning on the three sets. The five heuristics were run on all the problems in the TPTP library.

First classification experiments were done, one for each heuristic (labeled H1 to H5) plus the case of no heuristic being successful (labeled H0). For each heuristic the problems were separated into two classes. The problems in the positive class were those for which the heuristic found a solution faster than any of the other heuristics. SVMLite was used for the classifications with the radial basis kernel function as that had produced the best results in the first phase experiment.

The outcome was a series of models each of which would take a feature set as input and output a margin [2] value. For classification purposes only the sign of the margin is significant.

The final phase involved combining the models for the heuristics H1 to H5 and comparing the margin values produced. The most positive margin determined the heuristic to be selected. The learned heuristic selection scheme was compared with using each of the heuristics on their own. Additionally the case of selecting the best heuristic (knowing the final timings for all of them) was studied for comparison. This last case gives an upper limit to the performance possible when selecting perfectly between the five available heuristics.

3 Results and Conclusions

In the initial experiment, predicting easy and hard problems, a test set of 178 conjectures were classified. 123 were correctly classified. Of the 55 that were incorrectly classified there were 25 false positives and 30 false negatives.

In the classification experiments for the five heuristics the proportion of correct classifications was very close to the proportion of cases in the positive class (in the range of 80% to 90%). For the case of none of the heuristics being best (the H0 case) the number of correct classifications exceeded the proportion of cases in the positive class (74% to 60%).

Comparing static, dynamic and combined feature set results the static feature set did best and the combined set worst but by only a small margin. The difference may reflect the relative number of features in each set rather than the significance of the features themselves.

The heuristic selection results are given in the following table.

Table 1: Number of Theorems proved out of 3345 Conjectures

Static	Dynamic	Combined	H1	H2	H3	H4	H5	Optimum
1755	1764	1751	1739	1541	1626	1616	1542	2066

For all three feature sets (static, dynamic and combined) the results are better than any of the individual heuristics. In contrast to the classification results, the dynamic feature set does better than the static feature set. Though the improvement over heuristic H1 is only marginal it is clear that successful selection criteria have been learned as a random selection choice would do significantly worse than the best single heuristic.

Acknowledgements

Many thanks must go to Stephan Schulz, the author of E, for patiently responding to e-mail queries and making suggestions as to potentially useful features. Also T. Joachims for making SVMLite freely available for academic research.

References

[1] M. Fuchs. Automatic selection of search-guiding heuristics. *Proc. of the 10th FLAIRS, Daytona Beach, pages 1–5. Florida AI Research Society*, 8, 1997.

[2] T. Joachims. Making large-scale svm learning practical. *Advances in Kernel Methods - Support Vector Learning, B. Schlkopf and C. Burges and A. Smola (ed.), MIT-Press*, 1999.

[3] S. Schulz. E - a brainiac theorem prover. *Journal of AI Communications*, 15:111–126, 2002.

[2] The margin is a measure of the distance beyond a dividing hyperplane in a transformed sample space at which the image of the sample lies.

Tackling "Until Induction" in Natural Deduction for PLTL

Alexander Bolotov

*Harrow School of Computer Science, University of Westminster
Watford Road, Harrow HA1 3TP, UK.
A.Bolotov@wmin.ac.uk*

Abstract

We investigate the problem of induction in the natural deduction construction of propositional linear-time temporal logic. The well known induction with the "always in the future" operator has been an obstacle in our previous developments of the proof search. Here we modify the formulation of the calculus by introducing new rules to deal with the "until" operator, show the correctness of a modified system and define proof search strategies for new rules based upon fixpoint characterisation of the Until operator.

1 Introduction

In this paper we continue our investigation into proof searching technique for the natural deduction proof system for the propositional linear-time temporal logic PLTL [5] as part of a long term project. ND systems for classical propositional logic and first-order logic [1; 2] followed by their extensions to non-classical framework of propositional intuitionistic logic [9], propositional linear-time temporal logic PLTL [3] and computation tree logic CTL [4].

We argue that one of the advantages of our generic approach to build natural deduction is this possibility to follow some generic set of rules adapting and extending it correspondingly to the needs of the logical framework in question. Thus, developing our proof search strategy for PLTL [5], we extended strategies used in the classical case. Note also, that to the best of our knowledge, no other proof search algorithms for temporal ND systems exist. Indeed, the only other ND construction for linear-time logic [8] and branching-time logic [10], which we are familiar with, are not equipped with relevant proof searching techniques.

The induction with the "always" operator has been an obstacle in our previous developments of the proof search. Here we modify the formulation of the calculus by introducing new, more powerful rules, to deal with the "until" operator, show the correctness of a modified system and define proof search strategies for new rules based upon fixpoint characterisation of the Until operator.

2 Search strategies for until induction

Referring readers to [3] and [5], we present only the new rules that distinguish a new formulation of the system which we call PLTL$^{\mathcal{U}}_{ND}$ (to stress an emphasis on the $\mathcal{U}$ operator). Now instead of introduction or elimination rules for $\mathcal{U}$ we have rules capturing the behaviour of this operator as a minimal fixpoint.

$$\frac{i{:}\,\Box(B \Rightarrow C), i{:}\,\Box((A \wedge \bigcirc C) \Rightarrow C)}{i{:}\,A\,\mathcal{U}\,B \Rightarrow C}\,\mathcal{U}_{ind_1} \qquad \frac{i{:}\,A,\ i:\bigcirc(A\,\mathcal{U}\,B)}{i{:}\,A\,\mathcal{U}\,B}\,\mathcal{U}_{rec} \qquad \frac{i{:}\,A\,\mathcal{U}\,B}{i \leq j, j:B,\ i{:}\,\Box(B \Rightarrow C), i{:}\,\Box((A \wedge \bigcirc C) \Rightarrow C)}\,\mathcal{U}_{ind_2}$$

In the last rule C is a new proposition, j is flagged [3] and it binds i. The second rule, $\mathcal{U}_{rec}$ allows us to move the time in which $A\,\mathcal{U}\,B$ is true one step closer to the initial state each time the rule is applied.

The $\mathcal{U}_{ind_1}$ rule. Recall that $A\,\mathcal{U}\,B$ as a minimal fixpoint has the representation as in the equation $A\,\mathcal{U}\,B = \mu X.(B \vee A \wedge \bigcirc X)$ where μ is a minimal fixpoint operator. The idea of $\mathcal{U}_{ind_1}$ rule came to the authors after reading the sections on the axiomatics of temporal logic in [7; 6]. Recall that the rule has two premises: $x : \Box(B \Rightarrow C), x : \Box(A \wedge \bigcirc C \Rightarrow C)$ from which we derive $x : A\,\mathcal{U}\,B \Rightarrow C$. We wish to link the application of the $\mathcal{U}_{ind_1}$ rule with the searching procedure in [5], and thus, introduce the following:

1. Given that proof contains $A\,\mathcal{U}\,B$ and the current goal is G, we let $C = G$, and set up the following subgoals for the searching procedure: $x : \Box(B \Rightarrow G)$, $x : \Box(A \wedge \bigcirc G \Rightarrow G)$. If these formulae are derivable then the $\mathcal{U}_{ind_1}$ is applicable which results in $x : A\,\mathcal{U}\,B \Rightarrow G$, and since $x : A\,\mathcal{U}\,B$ is in the proof, we obtain $x : G$ by modus ponens. Thus, $x : A\,\mathcal{U}\,B$ has led us to the current goal, and in this sense, we do not need to consider it any longer.

2. If the above fails we let $C = B$ and set up the following subgoal for the searching procedure: $x : \Box(A \wedge \bigcirc B \Rightarrow B)$. Note that the other premise required for the application of $\mathcal{U}_{ind_1}$ in this case would be $x : \Box(B \Rightarrow B)$, which is always derivable. Given that both formulae derivable the $\mathcal{U}_{ind_1}$ is applicable which results in $x : A\,\mathcal{U}\,B \Rightarrow B$ and since $x : A\,\mathcal{U}\,B$ is in the proof, we obtain $x : B$ by modus ponens.

Let us informally explain this searching technique. Having derived $x : B$, by semantics of $\mathcal{U}$, we just simply guarantee that $A\,\mathcal{U}\,B$ is true at x simply because B is true at x. Thus, we can now consider a more complex formula $A\,\mathcal{U}\,B$ in the proof as redundant and hence will use $x : B$ in the other searching procedures.

3. If the above fails we let $C = A$ and set up the following subgoal for the searching procedure: $x : \Box(B \Rightarrow A)$. Note that the other premise required for the application of $\mathcal{U}_{ind_1}$ in this case would be $x : \Box(A \wedge \bigcirc A \Rightarrow A)$, which is always derivable. Given that both of these formulae are derivable the $\mathcal{U}_{ind_1}$ is applicable which results in $x : A\,\mathcal{U}\,B \Rightarrow B$ and since $x : A\,\mathcal{U}\,B$ is in the proof, we obtain $x : B$ by modus ponens. Let us informally explain this searching technique. Having derived $x : A$, and $x : \Box(B \Rightarrow A)$ we just linked the worlds where B is true following the semantics of $\mathcal{U}$ with the worlds where A is true: since B should become eventually true, there must be the world z such that $z : B$ and hence $z : A$. Due to $x : \Box(A \wedge \bigcirc A \Rightarrow A)$, $z - 1 : A$, hence $z - 2 : A$ etc.

4. If all the above fail, we apply the $\mathcal{U}_{ind_2}$ rule. Here, we express the semantical requirement to eventually satisfy B while two other formulae in the conclusion represent the fixpoint nature of $\mathcal{U}$. Since we have not found a specific form of C following $\mathcal{U}_{ind_1}$ rule, here as C we simply use a new proposition (earlier not occurring in the proof). The fact that C is a new proposition guarantees the soundness of the rule - we simply update a model where $A\,\mathcal{U}\,B$ was true deriving a new model by carefully making C true at those states in the original model where B was true first time after world x and at every state between this world and x.

With these modifications the new ND system is able to prove all axioms of PLTL and derive rules of axiomatics.

References

[1] A. Bolotov, V. Bocharov, A. Gorchakov, V. Makarov, and V. Shangin. *Let Computer Prove It.* Logic and Computer. Nauka, Moscow, 2004. (In Russian), Implementation of the proof search technique for classical propositional logic available on-line at `http://prover.philos.msu.ru`.

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

[3] 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.

[4] A. Bolotov, O. Grigoriev, and V. Shangin. Natural deduction calculus for computation tree logic. In *IEEE John Vincent Atanasoff Symposium on Modern Computing*, pages 175–183, 2006.

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

[6] E. A. Emerson. Temporal and Modal Logic. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science: Volume B, Formal Models and Semantics.*, pages 996–1072. Elsevier, 1990.

[7] Valentin Goranko. Temporal Logics Of Computations. lecture notes. 12th Summer School on Logic, Language and Information in Birmingham. 2000.

[8] A. Indrzejczak. A labelled natural deduction system for linear temporal logic. *Studia Logica*, 75(3):345–376, 2004.

[9] V. Makarov. Automatic theorem-proving in intuitionistic propositional logic. In *Modern Logic: Theory, History and Applications. Proceedings of the 5th Russian Conference*, StPetersburg, 1998. (In Russian).

[10] C. Renteria and E. Haeusler. Natural deduction for CTL. *Bulletin of the Section of Logic, Polish Acad. of Sci.*, 31(4): 231–240, 2002.

Representation Repair Plans in the Physics Domain

Alan Bundy

School of Informatics, University of Edinburgh, A.Bundy@ed.ac.uk

The automation of reasoning as deduction in logical theories is well established. Such logical theories are usually inherited from the literature or are built manually for a particular reasoning task. They are then regarded as fixed. We argue that they should be regarded as fluid. We are developing tools for the automated evolution of ontologies, in particular, their signatures, e.g., the splitting and merging of functions and the changing of their arities.

We are currently developing our techniques in the domain of physics [1; 3]. This is an excellent domain because many of its most seminal advances can be seen as signature evolution, i.e., changing the way that physicists view the world. These changes are often triggered by a contradiction between existing theory and experimental observation. These contradictions, their diagnosis and the resulting repairs have usually been well documented by historians of science, providing us with a rich vein of case studies for the development and evaluation of our techniques. The physics domain requires higher-order logic: both at the object-level, to describe things like planetary orbits and calculus, and at the meta-level, to describe the repair operations.

1 Repair Plans

We are developing a series of *repair plans* which operate simultaneously on a small set of small higher-order theories, e.g., one representing the current theory of physics, another representing a particular experimental set-up. Before the repair, these theories are individually consistent but collectively inconsistent. Afterwards the new theories are also collectively consistent. Each repair plan has a trigger formula and some actions: when the trigger is matched, the actions are performed. The actions modify the signatures and axioms of the old theories to produce new ones. The repair plans have been implemented in the GALILEO system (Guided Analysis of Logical Inconsistencies Leads to Evolved Ontologies) using λProlog [4] as our implementation language, because it provides a polymorphic, higher-order logic.

This combination of repair plans and multiple interacting logic theories helps to solve several tough problems in automated signature evolution.

- The overall context of the plan supplies the values of additional arguments and specifies which new function should replace which old one. Organising the theory as several interacting, small theories further guides the refinement, e.g., by enabling us to uniformly replace all the occurrences of an old function in one theory in one way, but all the occurrences in another theory in a different way.
- Grouping the operations into a predefined repair plan helps control search. This arises not only from inference, but also from repair choices. This solution is adopted from our work on *proof plans* [2].
- Having several theories helps us control inconsistency. A predictive theory and an observational one can be internally consistent, but inconsistent when combined. Since all sentences are theorems in an inconsistent theory, the triggers of all repair plans would be matched, creating a combinatorial explosion. This problem can be avoided when a trigger requires simultaneous matching across a small set of consistent ontologies.
- It is also enabling us to prove the minimality of our repair plans, i.e., to show that the repairs do not go beyond what is necessary to remove the inconsistency. We have extended the concept of *conservative extension* to signature evolution. We can now prove that the evolution of each separate theory is conservative in this extended sense. Of course, we do not want the evolution of the combined theory to be conservative, since we want to turn an inconsistent combined theory into a consistent one.

2 Some Repair Plans and their Evaluation

We have so far developed two repair plans, which we call *Where's my stuff?* (WMS) and *Inconstancy*. These roughly correspond to the refinement operations of splitting a function and adding an argument, respectively. We have found multiple examples of these repairs across the history of physics.

The WMS repair plan aims at resolving contradictions arising when the predicted value returned by a function does not match the observed value. This is modelled by having two theories, corresponding to the prediction and the observation, with different values for this function. To break the inconsistency, the conflicting function is split into three new functions: *visible*, *invisible* and *total*. The conflicting function becomes the total function in the predictive theory and the visible function in the observation theory[1]. The invisible function is defined as the difference between them, and this new definition is added to the predictive theory. The intuition behind this repair is that the discrepancy arose because the function was not being applied to the same *stuff* in the predictive and the observational theories — the invisible stuff was not observed.

WMS has been successfully applied to conflicts between predictions of and observations of the following functions: the temperature of freezing water; the energy of a bouncing ball; the graphs relating orbital velocity of stars to distance from the galactic centre in spiral galaxies; and the precession of the perihelion of Mercury. In these examples the role of the invisible stuff is played by: the latent heat of fusion, elastic potential energy, dark matter and an additional planet, respectively.

The Inconstancy repair plan is triggered when there is a conflict between the predicted independence and the observed dependence of a function on some parameter, i.e., the observed value of a function unexpectedly varies when it is predicted to remain constant. This generally requires several observational theories, each with different observed values of the function, as opposed to the one observational theory in the WMS plan. To effect the repair, the parameter causing the unexpected variation is first identified and a new definition for the conflicting function is created that includes this new parameter. The nature of the dependence is induced from the observations using curve-fitting techniques.

Inconstancy has been successfully applied to the following conflicts between predictions and various observations: the ratio of pressure and volume of a gas; the speed of light; and again the graphs relating orbital velocity of stars to distance from the galactic centre in spiral galaxies. The unexpected parameter of the function is the temperature of the gas, the distance from the light source, and the acceleration between the stars, respectively. The first of these repairs generalises Boyle's Law to the Ideal Gas Law, the second replaces Aristotle's concept of instantaneous light travel with a finite (but fast) light speed, and the third generalises the Gravitational Constant to Milgrom's MOND (MOdified Newtonian Dynamics).

Acknowledgements

The research reported in this paper was supported by EPSRC grant EP/E005713/1. It will soon be supported by EPSRC grant EP/G000700/1. I would like to thank Michael Chan, Lucas Dixon and Fiona McNeill for their feedback and for their contributions to the research referred to in this paper.

References

[1] A. Bundy. Where's my stuff? An ontology repair plan. In W. Ahrendt, P. Baumgartner, and H. de Nivelle, editors, *Proceedings of the Workshop on Disproving - Non-Theorems, Non-Validity, Non-Provability*, pages 2–12, Bremen, Germany, July 2007. http://www.cs.chalmers.se/∼ahrendt/CADE07-ws-disproving/.

[2] A. Bundy. A science of reasoning. In J.-L. Lassez and G. Plotkin, editors, *Computational Logic: Essays in Honor of Alan Robinson*, pages 178–198. MIT Press, 1991.

[3] A. Bundy and M. Chan. Towards ontology evolution in physics. In W. Hodges, editor, *Procs. Wollic 2008*. LNCS, Springer-Verlag, July 2008.

[4] D. Miller and G. Nadathur. An overview of λProlog. In R. Bowen, editor, *Proceedings of the Fifth International Logic Programming Conference/ Fifth Symposium on Logic Programming*. MIT Press, 1988.

[1] There are situations in which these roles are inverted.

Model Checking Agent Programming Languages*

Louise A. Dennis[⋆] *Rafael H. Bordini*[†] *Berndt Farwer*[†] *Michael Fisher*[⋆]
[⋆]*Department of Computer Science, University of Liverpool, UK* [†]*Department of Computer Science, Durham University, UK*

1 Introduction

The last decade has seen significant growth in both the amount and maturity of research being carried out in the area of *agent-based systems*. An agent can be seen as an *autonomous* computational entity, making its own decisions about what activities to pursue. *Rational agents* make such decisions in a rational and explainable way and, since agents are autonomous, understanding *why* an agent chooses a course of action is vital. Therefore, a key new aspect in the design and analysis of such systems is the need to consider not just what agents do but *why* they do it.

While program verification is well advanced, for example Java verification using Java PathFinder [6; 9], verification of agent-oriented programs poses new challenges that have not been adequately addressed, particularly in the context of practical model-checking tools. In agent verification, we have to verify not only what the agent does, but *why* it chose that course of action, *what* the agent believed that made it choose to act in this way, and what its intentions were in doing so.

Rather than providing an approach for *one* particular programming language, we here describe an architecture for a system allowing the verification of a wide range of agent-based programs, produced using various high-level agent-oriented programming languages. Our previous work [1; 2] has concentrated on model checking techniques for agent-based systems written in the logic-based agent-programming language AgentSpeak [8]. As described above, it is vital to verify not only the behaviour that the agent systems has, but to verify *why* the agents are undertaking certain courses of action. Thus, the temporal basis of model-checking captures the *dynamic* nature of agent computation, but is extended with *intensional modal operators* capturing the *informational* ('beliefs'), *motivational* ('desires') and *deliberative* ('intentions') aspects of a rational agent. Such pioneering work on *model checking* techniques for the verification of agent-based systems often based on standard model checkers such as Spin or JPF, has appeared, for example, in [1; 2; 7].

2 Architecture of the AIL

One of the problems with existing approaches to model checking agent programs is that one had to find ways of encoding beliefs, goals, etc., within the state of the JPF or Spin state machine. This is a complex task, and one that would need to be (at least partly) done again to allow model checking for other agent programming languages. As in other fields of Computer Science, with the emergence of various modelling formalisms (in particular here new agent programming languages), a need for a unifying framework arises.

We have investigated the key aspects underlying several BDI (Belief/Desire/Intention) programming languages [4]. Based on that, we have developed the Agent Infrastructure Layer (AIL) [5; 3], a collection of Java classes that: (i) enables implementation of interpreters for various agent languages, (ii) contains adaptable, clear semantics, and (iii) can be verified through AJPF, an extended version of the open source Java model checker JPF [9]. AJPF is a customisation of JPF that was optimised for AIL-based interpreters.

The AIL can be viewed as a platform on which agents programmed in different programming languages co-exist, and together with AJPF this provides uniform model checking techniques for various agent-oriented programming languages. This is further extended with the *MCAPL* [1] *interface* which allows programming languages that do not have their own AIL-based interpreters to be model checked against specifications written in the same property specification language. (However, these will not benefit from the efficiency improvements that the AJPF customisations provide.) Figure 1 provides a diagrammatic representation of the AIL architecture.

The AIL is not intended as a new language in its own right, but as an intermediate layer incorporating the main features of various existing (practical) agent languages. We have identified the key *operations* that many (BDI-)languages use and treat these operations as part of an *AIL toolkit*. The semantic rules in [4] are a part of this toolkit but any given language can use a selection of these rules in its own AIL-based interpreter and may choose to add its own custom rules built from the basic operations made available. These operations and rules have formal semantics and are implemented in Java.

*Work supported by EPSRC grants EP/D054788 (Durham) and EP/D052548 (Liverpool).

[1] *Model Checking Agent Programming Languages*.

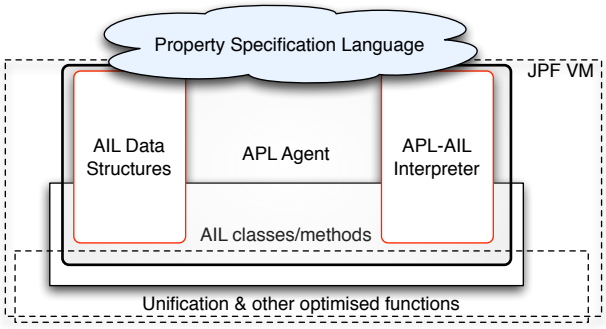


Figure 1: Overview of the AIL Architecture.

We assume that agents in any target agent programming language all possess a *reasoning cycle* consisting of several (≥ 1) stages. Each stage is a disjunction of rules that define how an agent's state may change during the execution of that stage. The combined rules of the stages of the reasoning cycle define the operational semantics of that language. The construction of an interpreter for a language involves the implementation of these rules (which in some cases might simply make reference to the pre-implemented rules) and a reasoning cycle. This means that the AIL can be viewed as a collection of Java classes/methods that provide the building blocks for custom programming of agent language interpreters, with the particular advantage of making model checking possible (and more efficient). In this way, we can implement, for example, an AgentSpeak interpreter following the AgentSpeak operational semantics but using the AIL operations rather than using Java from scratch.

Common to all language interpreters implemented using AIL methods are the AIL-agent data structures for beliefs, intentions, goals, etc., which are accessed by the model checker and on which the modalities of the property specification language are defined. The implicit data structures of a given BDI language need to be translated into the AIL's data structures. In particular, the initial state of an agent has to be translated into an AIL agent state.

In addition to the AIL toolkit, we also provide a set of Java interfaces that we call the *MCAPL interface*. As mentioned earlier, this allows agents to be model checked using the same property specification language even if no AIL-based interpreter for that language has been developed. An implementation of the MCAPL interface must define the required operators of the property specification language. For instance, agents implementing the MCAPL agent interface must provide a method which succeeds when the agents believes that the given parameter (represented as a "formula") is true. In other words, the implementation of such a method effectively corresponds to the semantics for the *belief* modality in that specific language. The AIL implements these interfaces and so defines an AIL-specific semantics for the property specification language; supported languages that use the AIL must ensure that they do so in a way that is consistent with their own semantics of those modalities.

References

[1] R. H. Bordini, M. Fisher, W. Visser, and M. Wooldridge. Model Checking Rational Agents. *IEEE Intelligent Systems*, 19(5):46–52, 2004.

[2] R. H. Bordini, M. Fisher, W. Visser, and M. Wooldridge. Verifying Multi-Agent Programs by Model Checking. *J. Autonomous Agents and Multi-Agent Systems*, 12(2):239–256, 2006.

[3] R. H. Bordini, L. A. Dennis, B. Farwer, and M. Fisher. Automated Verification of Multi-Agent Programs. In A. Ireland and W. Visser, eds., *Proc. 23rd IEEE/ACM International Conference on Automated Software Engineering (ASE'08)*. 2008, To Appear.

[4] L. A. Dennis, B. Farwer, R. H. Bordini, M. Fisher, and M. Wooldridge. A Common Semantic Basis for BDI Languages. In *Proc. 7th Int. Workshop on Programming Multiagent Systems (ProMAS)*, 2007.

[5] L. A. Dennis, B. Farwer, R. H. Bordini, and M. Fisher. A flexible framework for verifying agent programs (short paper). In Padgham, Parkes, Muller, and Parsons, eds., *Proc. of the 7th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2008)*. pp. 1303–1306, ACM, 2008.

[6] `http://javapathfinder.sourceforge.net`.

[7] F. Raimondi and A. Lomuscio. Automatic Verification of Multi-agent Systems by Model Checking Ordered Binary Decision Diagrams. *J. Applied Logic*, 5(2):235–251, 2007.

[8] A. Rao. AgentSpeak(L): BDI Agents Speak Out in a Logical Computable Language. In *Proc. 7th European Workshop on Modelling Autonomous Agents in a Multi-Agent World (MAAMAW)*, volume 1038 of *LNCS*, pages 42–55. Springer, 1996.

[9] W. Visser, K. Havelund, G. P. Brat, S. Park, and F. Lerda. Model Checking Programs. *Automated Software Engineering*, 10(2):203–232, 2003.x

Resolution-Based Proof for a Temporal Logic of Robustness

Clare Dixon[⋆]
[⋆]*Department of Computer Science, University of Liverpool, Liverpool, L69 3BZ, UK*
clare@csc.liv.ac.uk

John C. M[c]*Cabe-Dansted*[†]
[†]*School of Computer Science and Software Engineering The University of Western Australia, Australia*
john@csse.uwa.edu.au

Abstract

RoCTL is a branching time temporal logic which has additional operators relating to obligation and robustness. We provide a satisfiability preserving translation from RoCTL to the branching-time temporal logic CTL thus obtaining a proof procedure for RoCTL.

1 Introduction

The RoCTL* logic [5] is an extension of CTL* introduced to represent issues relating to robustness and reliability in systems. It does this by explicitly representing success and failure relations in the underlying model structures and using these to define an *obligatory* operator and a *robustly* operator. The obligatory operator ($\mathbf{O}$) specifies how the systems *should* behave by quantifying over paths in which no failures occur. The robustly operator ($\blacktriangle$) specifies that something must be true on the current path and on all paths that deviate from the current path that have at most one more failure than the current path. This notation allows phrases such as "even with n additional failures" to be built up by chaining n simple unary operators together. One of the strengths of RoCTL* is its ability to express contrary-to-duty obligations [4] which can be difficult for other Deontic logics.

Unfortunately the only known decision procedure for RoCTL* involves a reduction to QCTL* [5] which is non-elementary to decide. We therefore study a CTL-like restriction, termed RoCTL, and propose a translation into a normal form for CTL thus allowing us to use a known CTL resolution based decision procedure [1] to carry out RoCTL proofs. RoCTL is still reasonably expressive. For example it can still be used to express certain contrary-to-duty obligations.

This work provides a resolution proof procedure for RoCTL, a sub-logic of RoCTL*, via a satisfiability preserving translation to a normal form for CTL. Resolution based decision procedures have been developed for the branching time temporal logic CTL [1]. These first translate formulae into a normal form and then apply a number of resolution rules.

2 Syntax and Semantics of RoCTL

Formulae are constructed from a set $PROP = \{p, q, r, \ldots\}$ of *primitive propositions*. The language of RoCTL contains **true** and **false** and the standard propositional connectives $\neg$ (not), $\vee$ (or), $\wedge$ (and) and $\Rightarrow$ (implies). For the temporal dimension we take the usual set of future-time temporal connectives $\bigcirc$ (*next*), $\Diamond$ (*sometime* or *eventually*), $\Box$ (*always*), $\mathcal{U}$ (*until*) and $\mathcal{W}$ (*unless* or *weak until*). Each of these must be paired with a path operator $\mathbf{A}$ (*all paths*), $\mathbf{E}$ (*some path*), $\mathbf{O}$ (*obligatory*), $\mathbf{P}$ (*permissible*), $\blacktriangle$ (*robust*) and $\triangle$ (*prone*).

The set of well-formed formulae of RoCTL, WFF, is defined as follows:

- **false**, **true** and any element of $PROP$ is in WFF;
- if A and B are in WFF and $\mathbf{H} \in \{\mathbf{A}, \mathbf{E}, \mathbf{O}, \mathbf{P}, \blacktriangle, \triangle\}$ then the following are in WFF: $\neg A$; $A \vee B$; $A \wedge B$; $A \Rightarrow B$; $\mathbf{H}\Diamond A$; $\mathbf{H}\Box A$; $\mathbf{H}\bigcirc A$; $\mathbf{H}(A\,\mathcal{U}\,B)$; $\mathbf{H}(A\,\mathcal{W}\,B)$.

A RoCTL structure, M, is a 4-tuple $(A, \xrightarrow{s}, \xrightarrow{f}, \alpha)$ such that A is a set of states; $\xrightarrow{s}$ is a serial, binary success relation; $\xrightarrow{f}$ is a binary failure relation; and α is a valuation (a map from A to the powerset of propositional variables). Let $\rightarrow$ be an abbreviation for $\xrightarrow{s} \cup \xrightarrow{f}$. A *fullpath* is an infinite sequence of states $\sigma = \langle w_0, w_1, w_2, \ldots\rangle$ such that for all $i \geqslant 0$ $(w_i, w_{i+1}) \in \rightarrow$. Let $\sigma_{\geqslant i}$ be the fullpath $w_i, w_{i+1}, \ldots$, let σ_i be w_i and $\sigma_{\leqslant i}$ be $w_0, \ldots w_i$.

Definition 1 *A fullpath is* failure free *if and only if for all $i \in \mathbf{N}$ we have $w_i \xrightarrow{s} w_{i+1}$. Let $SF(w)$ be the set of fullpaths in M starting at state w and $S(w)$ be the set of all failure free fullpaths in M starting with w.*

Definition 2 *For two fullpaths σ and π, π is an* i-deviation *from σ if and only if $\sigma_{\leqslant i} = \pi_{\leqslant i}$ and $\pi_{\geqslant i+1} \in S(\pi_{i+1})$. π is a deviation from σ if there exists a non-negative integer i such that, π is an i-deviation from σ. A function δ from a fullpath to a set of fullpaths is defined as π is a member of $\delta(\sigma)$ if and only if π is a deviation from σ where σ and π are fullpaths.*

The semantics of RoCTL formulae are defined on a fullpath $\sigma = \langle w_0, w_1, \ldots \rangle$ in a RoCTL structure M follows. Recall $\sigma_i = w_i$ so $\sigma_0 = w_0$.

$$
\begin{array}{rll}
M, \sigma \models \bigcirc \varphi & \text{iff} & M, \sigma_{\geqslant 1} \models \varphi \\
M, \sigma \models \Box \varphi & \text{iff} & \forall i \in \mathbf{N}, M, \sigma_{\geqslant i} \models \varphi \\
M, \sigma \models \Box \varphi & \text{iff} & \exists i \in \mathbf{N}, M, \sigma_{\geqslant i} \models \varphi \\
M, \sigma \models \varphi \mathcal{U} \psi & \text{iff} & \exists i \in \mathbf{N} \text{ s.t. } M, \sigma_{\geqslant i} \models \psi \text{ and } \forall j \in \mathbf{N} \text{ s.t. } j < i,\ M, \sigma_{\geqslant j} \models \varphi \\
M, \sigma \models \varphi \mathcal{W} \psi & \text{iff} & M, \sigma \models \Box \varphi \text{ or } M, \sigma \models \varphi \mathcal{U} \psi \\
M, \sigma \models \mathbf{A} \varphi & \text{iff} & \forall \pi \in SF(\sigma_0)\ M, \pi \models \varphi \\
M, \sigma \models \mathbf{O} \varphi & \text{iff} & \forall \pi \in S(\sigma_0)\ M, \pi \models \varphi \\
M, \sigma \models \blacktriangle \varphi & \text{iff} & \forall \pi \in \delta(\sigma)\ M, \pi \models \varphi \text{ and } M, \sigma \models \varphi
\end{array}
$$

The definitions for propositions, and Boolean operators are as we would expect from classical logic. The semantics of other operators can be derived via equivalent formulae where $\mathbf{E}\varphi \equiv \neg\mathbf{A}\neg\varphi$, $\mathbf{P}\varphi \equiv \neg\mathbf{O}\neg\varphi$ and $\triangle\varphi \equiv \neg\blacktriangle\neg\varphi$. We say that a RoCTL formula φ is satisfiable if and only if for some structure M and some path σ, $M, \sigma \models \varphi$.

In the following let a literal be a proposition or a negated proposition.

3 Translating RoCTL into SNF$_{CTL}$

Next we provide a translation from formulae in RoCTL into a normal form for CTL. We just give an example for the combination of $\blacktriangle\Box$. The full translation can be found in [2]. In the following τ_1 is the translation which takes F which is a disjunction of literals marking the current path(s), and an implication with a RoCTL formula on the right hand side. There are two cases where p is a proposition and where G is not a proposition. In the former t and t' are new propositions marking the new current path(s) which are used in the translation of the latter.

$$
\begin{array}{rl}
\tau_1(F, x \Rightarrow (\blacktriangle \Box p)) = & \tau_2(x \Rightarrow t) \wedge (t \Rightarrow \mathbf{A}\bigcirc(\neg F \vee t)) \wedge (t \Rightarrow \mathbf{A}\bigcirc t') \wedge \\
& (t' \Rightarrow \mathbf{A}\bigcirc(viol \vee t')) \wedge \tau_2(t \Rightarrow p) \wedge \tau_2(t' \Rightarrow p) \\
\tau_1(F, x \Rightarrow (\blacktriangle \Box G)) = & \tau_1(F, x \Rightarrow (\blacktriangle \Box y)) \wedge \tau_1(t \vee t', y \Rightarrow G)
\end{array}
$$

We show that the translation is satisfiability preserving and involves a linear increase in the length of the formula [2]. This result, along with the fact that the complexity of satisfiability of CTL (a fragment of RoCTL) is EXPTIME [3] and recent results showing that the complexity of a resolution calculus based on SNF$_{CTL}$ is EXPTIME [6] allow us to conclude that the complexity of satisfiability of RoCTL is in EXPTIME. The translation we provide means we can apply a known resolution calculus [1] to the resulting clauses to provide a practical proof method for RoCTL.

References

[1] A. Bolotov. *Clausal Resolution for Branching-Time Temporal Logic*. PhD thesis, Dept. of Computing and Mathematics, Manchester Metropolitan University, 2000.

[2] C. Dixon and J.C. M$^{\text{c}}$Cabe-Dansted. Resolution for a temporal logic of robustness (extended version). Technical Report ULCS-08-002, University of Liverpool, Department of Computer Science, 2008. `http://www.csc.liv.ac.uk/research/techreports/`.

[3] E. A. Emerson and J. Y. Halpern. Decision Procedures and Expressiveness in the Temporal Logic of Branching Time. *Journal of Computer and System Sciences*, 30(1):1–24, February 1985.

[4] J.W. Forrester. Gentle murder, or the adverbial samaritan. *The Journal of Philosophy*, 81(4):193–7, April 1984.

[5] T. French, J.C. M$^{\text{c}}$Cabe-Dansted, and M. Reynolds. Temporal Logic of Robustness. In B. Konev and F. Wolter, editors, *Proceedings of the 6th International Symposium of the Frontiers of Combining Systems*, volume 4720 of *Lecture Notes in Artificial Intelligence*, pages 193–205. Springer, 2007.

[6] L. Zhang, U. Hustadt, and C. Dixon. A Refined Resolution Calculus for CTL. Submitted, 2008.

Dynamic relational rippling for HOL

Lucas Dixon[⋆] *and Dominic Mulligan*[†]

[⋆] *Mathematical Reasoning Group, Univeristy of Edinburgh.*
l.dixon@ed.ac.uk

[†] *Dependable Systems Group, Heriot-Watt University, Edinburgh.*
dpm8@macs.hw.ac.uk

Introduction

Rippling is a heuristic for guiding inductive proofs. It uses annotations to derive a measure that guides rewriting of the step case in order to make the inductive hypothesis applicable. Rippling is traditionally used to prove statements that are expressed using a functional notation, for example, the distributivity of multiplication over addition $a * (b + c) = (a * b) + (a * c)$. Rippling uses the nested structure of an expression in order to calculate a measure that guides the proof. Unfortunately, this limits its applicability to functional expressions and in particular it fails to guide the proof for relational properties. For instance, consider the confluence proof in figure 1. Such relational representations are widely used throughout mathematics and computer science; instances can be found in proving properties of logic programs, electronic circuits, and programming languages.

We outline an extension to rippling that makes it applicable to relational problems with shared existential variables. We revisit rippling annotations and generalise them to give guidance for relational proofs while preserving the ability to guide functional proofs.

Background

The existing implementation of rippling in IsaPlanner uses embeddings between terms to calculate a rippling annotation [2]. The embeddings are homomophic mappings from a source term tree (typically the induction hypothesis) into a destination term (typically the goal). The parts of the destination term that are unmapped correspond to the differences between the induction hypothesis and the goal and are called *wave fronts*. For example, a goal $(p\ x\ n)$ embeds into $p\ x\ \boxed{(s\ \boxed{n})}$ Where the function s is shaded to indicate that it is in the wave front. Rippling uses the differences to guide rewriting: it tries to move or remove the differences such that the source and destination terms unify. This allows an inductive proof to be completed by applying the induction hypothesis.

Relational representations share values via existential variables and conjunction. For example, consider the functional expression $y = f\ (g\ x)$. When translated into relational form, this becomes $\exists g_x.\ (p_g\ x\ g_x) \wedge (p_f\ g_x\ y)$, where p_g and p_f are relations corresponding to the functions g and f respectively, and g_x is a shared existential variable. The use of conjunction to connect expressions introduces associative-commutative (AC) variants of relational problems. For instance, the above goal can also be written as $\exists g_x.\ (p_f\ g_x\ y) \wedge (p_g\ x\ g_x)$. AC symmetries, as well as the lack of nested functions, motivate specialised relational annotations that take these into account. The traditional rippling annotations fail to provide guidance for relational proofs.

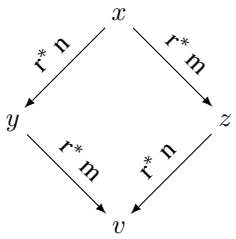
The only existing approach to relational rippling that we know of is that proposed in [1]. However, it is based on a non-standard proof system, is not fully implemented, and is not applicable to higher order logics. We address these problems by introducing a new rippling heuristic based on embeddings and naming goals and assumptions. This treats relational, functional and mixed expressions in a unified manner. An implementation is almost completed for the IsaPlanner system. This allows relational properties to be formalised in Isabelle/HOL and the proofs to be automated by IsaPlanner.

Annotations for Relational Rippling

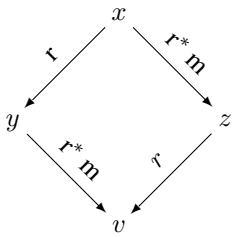
Our extension of rippling to relational domains involves two modifications to the traditional account:

- We use a representation of assumptions and goals that ignores AC permutations of conjunction. Each conjunct is given a unique name; goals in a proof state are named and their context is a set of assumption names.
- We extend the embedding algorithm to allow differently named variables to embed when there are relations that connect them. For example in the relational expression $(r\ a\ b) \wedge (r_2\ b\ c) \wedge (r_3\ c\ d)$, the variable b is related to d by the relations r_2 and r_3; these relations provide a path that connects the variables.

Confluence Theorem:
$\forall x, y, z.\ r^*\ n\ x\ y\ \wedge\ r^*\ m\ x\ z$
$\rightarrow \exists v.\ \ r^*\ m\ y\ v\ \wedge\ r^*\ n\ z\ v$



1 Sided Confluence Lemma:
$\forall x, y, z.\ r\ x\ y\ \wedge\ r^*\ m\ x\ z$
$\rightarrow \exists v.\ r^*\ m\ y\ v\ \wedge\ r\ z\ v$



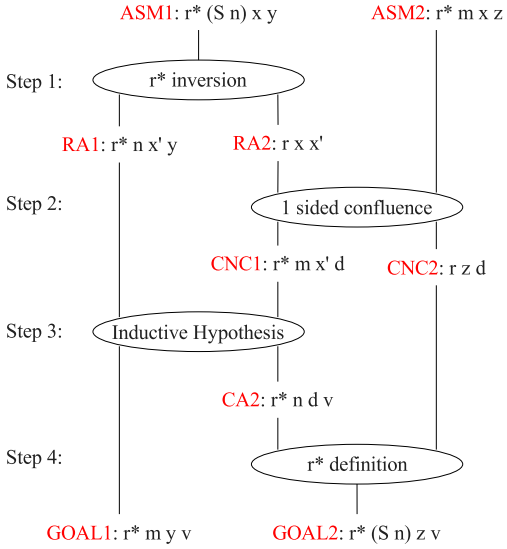


Figure 1: Left: Commuting diagrams for the confluence theorem and the single step confluence lemma. Right: presentation of the confluence proof.

As a consequence of allowing relational expressions, each argument of a relation can have relational as well as functional differences. Relational differences are computed simply by counting the (minimum) number of relations needed to connect the souce variable and the target variable. Thus embedding $(r_3\ a\ d)$ into $(r_3\ c\ d)$, given $(r\ a\ b) \wedge (r_2\ b\ c)$ results in a relational difference of 2 (from embedding a into c).

This approach to relational rippling introduces additional choices that need to be made in order to guide rippling. In particular, each argument position must prefer functional or relational differences with respect to other positions. This defines which kind of differences in which argument possitions are preferred in order to lead to application of the induction hypothesis. Because this ordering is problem specific, we use a dynamic table which instantiates an ordering as the proof proceeds. To illustrate this, we consider an abstract proof of confluence for a relation, r, given the one step diamond property $((r\ x\ y) \wedge (r\ x\ z) \rightarrow \exists v.\ \ (r\ y\ v) \wedge (r\ z\ v))$. The confluence theorem's statement, its proof, and the main lemma are shown in Figure 1. The successor function on n is written $S\ n$ and r^* is defined inductively by $(r\ x\ y) \rightarrow (r^*\ 0\ x\ y)$ and $(r\ v\ x) \wedge (r^*\ n\ x\ y) \rightarrow (r^*\ (S\ n)\ v\ y)$. The inversion rule corresponds to case analysis on this definition, for instance: $(r^*\ (S\ m)\ v\ y) \rightarrow \exists x.\ \ (r\ v\ x) \wedge (r^*\ m\ x\ y)$.

In the confluence proof, rippling first guides rewriting of the assumptions `ASM1` and `ASM2` towards the assumptions of the inductive hypothesis into which they embed. Initially, there is a funtional difference of 1 on the first argument of r^* in `ASM1`, the first step of rewriting changes this into a relational difference on the second argument. This creates the first value in our dymanic ordering, $r^*_{R2} < r^*_{F1}$, which states that a relational difference on the second argument is considered to be better than a functional difference on the first argument. Observe that the definition of r^* is not then imidiately applied; this would move the relational difference back to a functional one which would not preserve the partially instantiated order. The second step of rewriting removes this relational difference allowing the induction hypothesis to be applied. Finally, the single functional difference between `CA2` and `GOAL2` is removed by using the r^* definition. This illustrates that relational rules can be used in both directions without suffering loss of termination.

Conclusion

We have outlined a new approach to relational rippling. This can be used within the existing interactive provers without asserting new axioms and has been shown to successfully guide a confluence proof. A full implementation in IsaPlanner is ongoing work.

References

[1] Alan Bundy and Vincent Lombart. Relational rippling: A general approach. In *IJCAI*, pages 175–181, 1995.

[2] Lucas Dixon and Jacques D. Fleuriot. Higher order rippling in IsaPlanner. In *TPHOLs*, volume 3223 of *LNCS*, 2004.

Towards Verification by Symbolic Debugging

Holger Gast[⋆]

[⋆] *Wilhelm-Schickard-Insitut für Informatik*
University of Tübingen
Sand 13, 72076 Tübingen
gast@informatik.uni-tuebingen.de

Abstract

Semi-automatic or interactive software verification requires the programmer to understand the generated proof obligations by connecting them to the source code being verified. A major difficulty lies in the backward-style reasoning employed by most programming logics. We propose that Hoare-style verification with forward reasoning is feasible and useful. The verification process appears to the user as "symbolic debugging", in which the pre-condition of a Hoare triple captures the "current state" and evolves to reflect the side-effects encountered during execution. As a result, programmers can apply their operational understanding of the program to the solving the arising proof obligations.

1 Introduction

Many verification tools highlight the connection between the verification conditions and the source code to facilitate verification: Boogie ([2]) reports failed proofs as annotations; JACK ([4]) marks the control paths that lead to proof obligations; Jive ([7]) allows the user to apply verification rules interactively; KIV ([6]) and the Key tool ([1]) execute programs symbolically in dynamic logic. The problem that remains is that in proof obligations all approaches rely on backward-style reasoning, which goes against the operational understanding that programmers are familiar with (see [4]). Even symbolic execution of statement s in dynamic logic results in a proof obligation $P \vdash \mathcal{U}\, Q$ where $\mathcal{U}$ is an *update* that captures the effect of s. The update is then simplified by modifying the post-condition Q [3, §5].

We propose to apply a guideline for designing user interfaces to the design of verification environments: The interface should consistently implement a metaphor for procedures that the user is already familiar with. One obvious candidate here is interactive debugging: Programmers usually have extensive experience with stepping through the source code, and examining the resulting states in detail to locate the source of an unexpected result. The corresponding metaphor for verification environments is this: The precondition of a Hoare triple $\{P\}s\{Q\}$ captures the current state when statement s starts executing. The programmer should then be able to step through s and see how the precondition, and thus the "current state", evolves with each executed step. The verification environment becomes a symbolic debugger that allows the programmer to inspect the behaviour of the program for all possible possible inputs.

We have implemented this approach in Isabelle/HOL for a low-level, C-like language L_0. The technical details of the development are given in ([5]). The focus there is on automated reasoning about disjointness of memory regions for simplifying memory updates. The purpose of this presentation is to demonstrate by examples the usefulness of the "verification as symbolic debugging" metaphor. Furthermore, we describe the specialized tactics that enable forward-style reasoning about programs in Isabelle, which is geared towards backward-style, goal directed proving.

2 The Hoare Logic for Forward Reasoning

The Hoare logic for L_0 programs is designed to parallel the operational semantics of the language. A primary example is the rule for the assignment expression. Expressions can be evaluated as l- and r-values, as in C, using a standard big-step semantics [5, §3]. The Hoare rules are then lemmata about relations ⊨l { P } e { Q } and ⊨r { P } e { Q } defined to capture partial correctness as usual. The assignment rule, in Isabelle's syntax, is:

⟦ ⊨l { P0 } e1 { P1 };
  ∀a. (⊨r { λΓ M. P1 Γ M a } e2 { P2 a } ∧
    (∀Γ M v. P2 a Γ M v ⟶ M ▷ typed-block Γ a t ∧ tyval Γ v t))
⟧ ⟹ ⊨l { P0 } EAssign t e1 e2 { λΓ M a. ∃M' v. P2 a Γ (STORE-TYPED Γ a t M' M) v ∧ v = rd Γ a t M }

The context Γ maps local variables to their addresses, M and M' are memory states, a is an address, and v is a value. The assertions in the Hoare triples are (higher-order) predicates on these entities. The rule has an operational reading: If e1 takes a memory state described by P0 to a memory state (and result) described by P1, and e2 takes that state to a state described by P2, then after the assignment, P2 holds for the resulting memory state, except that reading from adress a

yields the value returned by e2, and a sequence of bytes has been modified, as represented by the STORE-TYPED operator ([5]). Since pointers are available, a side-condition checks that the accessed area is indeed allocated after e2 has been executed. Note that the assertions P0, P1, and P2 inspect the passed memory state using the function rd. The generated post-condition will therefore contain, after β-reduction, sub-expressions rd Γ a' t' (STORE-TYPED Γ a t M' M).

3 Forward Reasoning Tools

Most of Isabelle's tactics support backward-style, goal-oriented proving, while support for forward-style reasoning is very limited. To enable "verification as debugging", we have implemented the following specialized tactics.

Unfolding and Folding of Assertions A post-condition P generated by the forward application of Hoare rules has the form $\lambda\Gamma$ M. $\exists\, x_1 \ldots x_n$. P' where P' may contain any of the preceding bound variables. The user may manipulate assertions using weakening $\mathsf{P} \Rrightarrow Q$, where Q is a variable. Of the existing Isabelle tactics, only substitution and simplification can access subterms of P. We therefore *unfold* $\mathsf{P} \Rrightarrow Q$ to a proper goal $\bigwedge \Gamma\, M\, x_1 \ldots x_n.\mathsf{P}' \Longrightarrow Q$ on which all Isabelle tactics operate. In particular if P' is a conjunction, then all conjuncts will be available as separate assumptions in the goal. After the manipulation, the tactic `foldup` instantiates Q with a term $\lambda\Gamma$ M. $\exists\, x_1 \ldots x_n$. Q'.

Lightweight Separation The tactic `sep` ([5]) rewrites unfolded goals using conditional rules like the following:

"is-valid (typed-block Γ a t || typed-block Γ a' t') $\Longrightarrow$ rd Γ a' t' (STORE-TYPED Γ a t M' M) = rd Γa a' t' M"

The rule asserts that the memory access by rd is independent of the memory update represented by STORE-TYPED, if the accessed and modified address ranges do not overlap. `sep` uses assertions about the memory layout to prove this condition and drop memory update operator.

Naming Old Values Some accessor/modifier pairs will, of course, fail to simplify. If we have, for instance, an assertion about the content of a variable `i` and write to that variable, then the assertion contains a term rdv Γ "i" (STORE-VAR Γ "i" M' M). The tactic `name_old_vals` replaces such terms with existentially quantified variables, in this case `old_i`.

Factoring of Disjunctions The rules for `if` statements and the short-circuit logical or operator generate post-conditions of the form $\lambda\Gamma M.\exists x_1 \ldots x_n.P_1\Gamma M \vee \exists y_1 \ldots y_m.P_2\Gamma M$, where P_1 and P_2 describe the post-state of the respective branches. Since both P_1 and P_2 have evolved from a common predecessor P_0, those conjuncts R of P_0 that are not affected by the state updates in the branches will be duplicated. Also those existentially quantified variables $x_1 \ldots x_i$ that were already present in P_0 wil occur as $y_1 \ldots y_i$ as well. A tactic `factor_disj` replaces the above term with the simpler $\lambda\Gamma M.\exists x_1 \ldots x_i.R \wedge (x_{i+1} \ldots x_n.P_1'\Gamma M \vee \exists y_{i+1} \ldots y_m.P_2'\Gamma M)$.

References

[1] Wolfgang Ahrendt, Bernhard Beckert, Reiner Hähnle, Philipp Rümmer, and Peter H. Schmitt. Verifying object-oriented programs with KeY: A tutorial. In *5th International Symposium on Formal Methods for Components and Objects, Amsterdam, The Netherlands*, volume 4709 of *LNCS*, pages 70–101. Springer, 2007.

[2] Mike Barnett, Bor-Yuh Evan Chang, Robert DeLine, Bart Jacobs, and K. Rustan M. Leino. Boogie: A modular reusable verifier for object-oriented programs. In *Fourth International Symposium on Formal Methods for Components and Objects*, volume 4111 of *LNCS*. Springer, 2006.

[3] Bernhard Beckert. A dynamic logic for the formal verification of Java Card programs. In *JavaCard '00: Revised Papers from the First International Workshop on Java on Smart Cards: Programming and Security*, pages 6–24, London, UK, 2001. Springer-Verlag. ISBN 3-540-42167-X.

[4] Lilian Burdy, Antoine Requet, and Jean-Louis Lanet. Java applet correctness: A developer-oriented approach. In Keijiro Araki, Stefania Gnesi, and Dino Mandrioli, editors, *FME*, volume 2805 of *Lecture Notes in Computer Science*, pages 422–439. Springer, 2003. ISBN 3-540-40828-2.

[5] Holger Gast. Lightweight separation. In *Theorem Proving in Higher Order Logics 21st International Conference, TPHOLs 2008*. Springer, 2008. (to appear).

[6] Dominik Haneberg, Simon Bumler, Michael Balser, Holger Grandy, Frank Ortmeier, Wolfgang Reif, Gerhard Schellhorn, Jonathan Schmitt, and Kurt Stenzel. The user interface of the KIV verification system - a system description. In *Proceedings of the User Interfaces for Theorem Provers Workshop (UITP 2005)*, 2005.

[7] J. Meyer and A. Poetzsch-Heffter. An architecture for interactive program provers. In S. Graf and M. Schwartzbach, editors, *TACAS '00, Tools and Algorithms for the Construction and Analysis of Software*, volume 276 of *LNCS*, pages 63–77, 2000.

Proving Functional Properties in Separation Logic

Andrew Ireland, Ewen Maclean and Ianthe Hind {air,eahm2,isah1}@macs.hw.ac.uk

1 Motivation

In the past cumbersome reachability proofs have hindered automatically verifying relatively simple array programs. In Separation logic [4], reasoning about arrays and pointers can be done without the need for these explicit reachability side conditions.

The CORE project aims to exploit the reasoning power of Separation Logic to prove properties of imperative programs which manipulate the heap via pointers [1]. In particular we are interested in proving the functional properties of programs, as well as their safety properties. One of the more challenging problems which we hope to address is the generation of loop invariants via existing and novel proof planning techniques [7; 6; 8].

2 Separation Logic

The central syntactic elements which define Separation Logic are all characterised with respect to a semantics about heap structures. *emp* represents the empty heap, $*$ represents separating conjunction, $-\!\!*$ represents separating Implication, and $X \mapsto E$ represents a singleton pointer which represents a single cell in the heap.

These notions are combined with standard Higher Order Logic notions of implication to allow for reasoning both about resources – storage on the heap – and purely functional elements, which describe properties of the values of cells in the heap. Importantly Separation Logic extends Hoare logic so that verification conditions which are generated via weakest precondition analysis describe heap manipulations using the extra logical syntax itemised above.

3 Example

A canonical example for separation logic is the verification of *In-place List Reversal* - a procedure shown in figure 1, where $\alpha 0$ is the initial value of α. The preconditions are written in braces in the code. The details of the list predicate used, and of applying proof planning techniques to such a proof are described in detail in [9].

```
{(∃ α. list(α, i, nil) ∧ α0 = α) }
j := nil;
{(∃ α, β. list(α, i, nil) * list(β, j, nil) ∧ α0 = app(rev(β), α))}
while (i != nil) do
   k := i.2;
   i.2 := j;
   j := i;
   i := k;
od
{(∃ β. list(β, j, nil) ∧ rev(α0) = β) }
```

Figure 1: In-place List Reversal

4 Functional vs. Spatial

There is existing successful work on verifying the correctness of pointer programs with respect to safety [1; 2]. This means that the values of the heap cells are ignored and purely the shape of the heap is analysed. A correct loop invariant for the program shown in figure 1 is shown below, annotated with the spatial and functional parts.

$$(\exists \alpha, \beta.\ \underbrace{list(\alpha, i, nil) * list(\beta, j, nil)}_{\textbf{spatial}} \wedge \underbrace{\alpha 0 = app(rev(\beta), \alpha)}_{\textbf{functional}})$$

[1] This work is funded by EPSRC grant EP/F037597

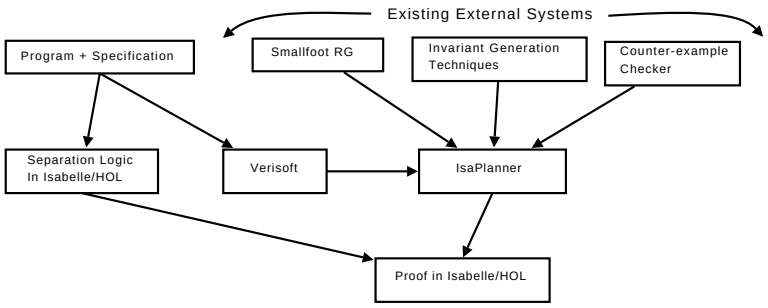


Figure 2: Proposed Architecture for the CORE Project

Our work is predominantly interested with extending existing shape analysis work to verify the functional correctness of programs that manipulate pointers. In the list reversal example, the memory safety proof would be concerned with showing that no unallocated memory is accessed, and that the amount of memory used is constrained to that allocated in the precondition. We are interested in showing not only that the memory usage is safe, but that the resulting program reverses the list described in the precondition.

5 Work Achieved

So far we have started a deep embedding of Separation Logic in Isabelle/HOL in which we can perform proof. Above this lies IsaPlanner [3], which is a sophisticated planning system which has been used to successfully automate inductive proofs, and allows for controlled instantiation of meta-variables, for example when discovering missing lemmas.

The overall architecture, is shown in figure 2. We plan to extend the Verisoft system [10] to generate verification conditions in the Separation Logic expressions defined via the deep embedding. IsaPlanner is then used to automate the resulting entailment proofs with the help of external systems such as *Smallfoot RG* [2].

We have used IsaPlanner so far to solve problems where loop invariants are represented by higher-order meta-variables, and synthesised using a search over a limited set of function symbols, and checked using a counter-example checker. This technique has successfully synthesised loop invariants for simple array programs, but we hope that it will scale up to more complicated problems, and incorporate separation logic. We also plan to extend rippling techniques [5] in order to aid automation and proof analysis.

References

[1] J. Berdine, C. Calcagno, and P. O'Hearn. Smallfoot: Modular automatic assertion checking with separation logic. In *FMCO*, LNCS Vol 4111, Springer, 2005.

[2] C. Calcagno, M. Parkinson, and V. Vafeiadis. Modular safety checking for fine-grained concurrency. In *To appear in the Proceedings of SAS 2007*, 2007.

[3] L. Dixon and J. D. Fleuriot. IsaPlanner: A prototype proof planner in Isabelle. In *Proceedings of CADE'03*, LNCS Vol. 2741, 2003.

[4] J. C. Reynolds. Separation logic: A logic for shared mutable data structures. In *Logic in Computer Science*. IEEE Computer Society, 2002.

[5] A. Bundy, D. Basin,D. Hutter and A. Ireland. Rippling: Meta-level Guidance for Mathematical Reasoning. Cambridge University Press, 2005

[6] A. Ireland and J. Stark. Proof Planning for Strategy Development. in *Annals of Mathematics and Artificial Intelligence*, Vol. 29, pages 65-97, Kluwer, 2001

[7] A. Bundy. The use of explicit plans to guide inductive proofs. In R. Lusk and R. Overbeek, editors, *9th International Conference on Automated Deduction*, Springer, 1988.

[8] A. Ireland, B.J. Ellis, A. Cook, R. Chapman and J. Barnes. An Integrated Approach to Hight Integrity Software Verification. Journal of Automated Reasoning, Vol. 36, pages 379-410, Kluwer 2006

[9] A. Ireland. A Cooperative Approach to Loop Invariant Discovery for Pointer Programs. in *Proceedings of 1st International Workshop on Invariant Generation*, 2007

[10] N. Schirmer. A Verification Environment for Sequential Imperative Programs in Isabelle/HOL. In G. Klein, editor, *Proc. NICTA Workshop on OS Verification 2004*, 2004.

Normalizations in Propositional Logic*

Manfred Kerber
School of Computer Science
University of Birmingham
Birmingham B15 2TT, England
http://www.cs.bham.ac.uk/~mmk

Introduction

Even simple propositional logic allows for infinitely many formulae. When normalized to a clause normal form, there are only finitely many normal forms (without tautologies). Any such clause can be represented as a string over the alphabet $\{0, 1, \#\}$: Assumed we have n boolean variables $\mathtt{X}_1, \mathtt{X}_2, \ldots, \mathtt{X}_n$ then we can represent a (non-tautological) clause C by a string $c_1 c_2 \cdots c_n$ as follows: for every i c_i is 0 if $\mathtt{X}_i$ occurs negatively in C, 1 if it occurs positively, and # if it does not occur at all. For instance, for $n = 6$, the string $0\,\#\,1\,1\,1\,\#$ represents $\neg\mathtt{X}_1 \vee \mathtt{X}_3 \vee \mathtt{X}_4 \vee \mathtt{X}_5$ and $1\,\#\,\#\,1\,0\,\#$ represents $\mathtt{X}_1 \vee \mathtt{X}_4 \vee \neg\mathtt{X}_5$. This representation is used in the field of classifier systems, in which problems can be represented – using this string representation of clauses – in different ways resulting in different complexities (see, for instance, [4]).

With this string representation of clauses we see that there are 3^n different clauses altogether. Since the clause sets which contain the empty clause are all unsatisfiable anyway and can be recognized as such computationally very cheaply, we can take all these clause sets out of our consideration. That is, we get $3^n - 1$ different (non-empty) clauses with n boolean variables. Hence there are all in all $f(n) = 2^{3^n-1}$ different clause sets (without tautologies and without the empty clause). Although finite, this set grows doubly exponentially in the number of boolean variables. Normalization beyond clause normal form must be applied or a study becomes infeasible even for very small n.

n	$f(n)$
0	1
1	4
2	256
3	67108864
4	1208925819614629174706176
5	70673882591135373183331900029716740633099355875024758324864248051704791 04

For $n = 0$ there is only the one possibility to remain silent. For $n = 1$, we have the clauses $\mathtt{X}_1$, $\neg\mathtt{X}_1$, and the four clause sets: $\emptyset$, $\{\mathtt{X}_1\}$, $\{\neg\mathtt{X}_1\}$, and $\{\mathtt{X}_1, \neg\mathtt{X}_1\}$. For $n = 3$ there are already more than sixty million different clause sets.

Reductions

Since the names of the boolean variables do not matter, we can identify clause sets which can be transformed into each other by swapping variable names, likewise those which can be transformed into each other by flipping the sign of some variables consistently. Such symmetry transformations have been studied in the area of constraint satisfaction problems, see in particular the work by Frisch et al. [2; 3], where permutations and changes of polarity of boolean variables play a major role. Traditionally subsumed clauses and clauses containing pure literals can be removed from a clause set (see for instance traditional theorem proving textbooks such as [1]).

Although it is theoretically possible to follow a naive approach and to first generate the whole class of all clause sets and then reduce it, this is not feasible for doubly exponential problems even for small n. The set of all clauses can be effectively generated for $n \leq 10$ (or modestly bigger n; for $n = 10$ there are $3^n - 1 = 59049$ different clauses). Even for $n = 3$ there are, however, more than 60 million different clause sets. For any bigger n it is certainly infeasible to compute the full class of all clause sets. For this reason it is necessary to keep the set on construction as small as possible, that is, we want to apply the reductions on construction as much as possible. But may we? Or will we change the result this way? We call the reductions which may be applied during construction *stable*. The properties `permutation`, `flip`, and `subsumed` are stable, and `pure` is not.

*I would like to thank Riccardo Poli for stimulating discussions which inspired this work. A full version of this contribution can be found in the Proceedings of AISC/Calculemus/MKM 2008, Springer, LNAI 5144, p. 494–503.

For $n = 3$, the total class with a size of 67108864 elements can be reduced by the application of subsumption to 15935 elements. This can be reduced further by the combined application of permutation/negation reduction to 522 elements. Finally purity reduction reduces the set to 410 elements.

Summarizing we get the following numbers of different clause sets after the cumulative application of reductions:

n	no red.	subsum	subsum+neg/perm	subsum+neg/perm+purity
1	4	4	3	2
2	256	47	14	8
3	67108864	15935	522	410

E.g., for $n = 2$ the 8 elements after all reductions are:

```
NIL
("00" "11")                ("#1" "#0")
("00" "01" "10")           ("#0" "01" "11")       ("#0" "0#" "11")
("00" "01" "10" "11")      ("#0" "#1" "0#" "1#")
```

These eight cases are clearly non-isomorphic since they can be classified by the number of clauses contained (0, 2, 3, or 4), and within these classes by the total number of # symbols contained in the clause sets. It is under current investigation to find a similar classification for the 410 elements for $n = 3$. If we had a general classification principle, this could be used in order to come up with an effective mechanism to create the reduced class of clause sets for $n = 4$ without having to go back to the powerset construction. Note, however, that it cannot be expected that the growth in number of the reduced clause set classes is in a better complexity class compared to the full class. That is, they are expected to be doubly exponential as well. However, the much reduced numbers make them more accessible to complete investigation for small n, in particular for $n = 4$. Because of the special situation of two-clauses, the case $n = 3$ is the first really interesting case and it would be good to have at least the case $n = 4$ at hand as well to come up with generalizable conjectures.

Let us add the empty clause to the class of reduced clause sets for $n = 3$. Of the 411 different cases, 207 are not satisfiable and 204 are satisfiable. The non-satisfiable ones can be ordered as a partial order with respect to a relation of clause sets in which the empty clause comes lowest. A clause set is immediately below another one if the first can be generated from the second by a single application of the binary resolution rule.

In order to get from the top most by binary resolution to the empty clause it is necessary to apply binary resolution at least 7 times and at most 18 times. That means that any heuristic to reduce proof search can at best reduce the search from 18 steps to 7 steps for this particular example. A long term goal of this work to theorem proving is to better understand the impact of theorem proving heuristics on the class of *all* problems rather than on a set of challenge problems. This way this work may contribute to a better understanding of heuristics and help to speed up theorem provers. The hope would be to generalize from cases with few propositional logic variables (e.g., $n = 2$, $n = 3$, and $n = 4$) to the arbitrary propositional logic case and to lift results to first order logic.

Another application of this work is in the search of similar expressions. The work can be used in order to detect structurally equivalent formulae. This is relevant when different formulations of the same problem are given; in this case these formulations are typically not logically equivalent. This becomes more relevant for logics which are more powerful than the ones studied here.

References

[1] Chin-Liang Chang and Richard Char-Tung Lee. *Symbolic Logic and Mechanical Theorem Proving*. Academic Press, New York, USA, 1973.

[2] Alan M Frisch, Chris Jefferson, Bernadette Martinez Hernandez, and Ian Miguel. The rules of constraint modelling. In *Proceedings of the 19th IJCAI*, pages 109–116, 2005. `http://www.cs.york.ac.uk/aig/constraints/AutoModel/conjure-ijcai05.pdf`.

[3] Alan M Frisch, Chris Jefferson, Bernadette Martinez Hernandez, and Ian Miguel. Symmetry in the generation of constraint models. In *Proceedings of the International Symmetry Conference*, 2007. `http://www.cs.york.ac.uk/aig/constraints/AutoModel/ISC07Conjure.pdf`.

[4] Tim Kovacs and Manfred Kerber. A study of structural and parametric learning in XCS. *Evolutionary Computation Journal*, **14**(1):1–19, Spring 2006.

Fair Monodic Temporal Logic Proving

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 $\mathbb{N}$, that is, the most commonly used model of time). 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.

Every monodic temporal formula can be 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. Initial and universal clauses are ordinary first-order clauses, containing no temporal operators. *Step* clauses in the clausal form of monodic temporal formulae are of the form $p \Rightarrow \bigcirc q$, where p and q are propositions, or of the form $P(x) \Rightarrow \bigcirc Q(x)$, where P and Q are unary predicate symbols and x a variable. During a derivation more general *step* clauses can be derived, which are of the form $C \Rightarrow \bigcirc D$, where C is a *conjunction* of propositions, atoms of the form $P(x)$ and ground formulae of the form $P(c)$, where P is a unary predicate symbol and c is a constant such that c occurs in the input formula, D is a *disjunction* of arbitrary literals, such that C and D have at most one free variable in common. *Eventuality* clauses are of the form $\Diamond L(x)$, where $L(x)$ is a literal having at most one free variable.

2 Fine-Grained Resolution

In [2] the monodic fine-grained temporal resolution calculus was introduced for monodic FOTL, which consists of the *eventuality resolution rule*:

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

where $\forall x(\mathcal{A}_i(x) \Rightarrow \bigcirc\, \mathcal{B}_i(x))$ are complex combinations of step clauses such that for all $i \in \{1, \ldots, n\}$, 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; and the following five rules of *fine-grained step resolution*:

1. *First-order resolution between two universal clauses and factoring on a universal clause.* The result is a universal clause.
2. *First-order resolution between an initial and a universal clause, between two initial clauses, and factoring on an initial clause*. The result is again an initial clause.
3. *Fine-grained (restricted) step resolution*.
$$\frac{C_1 \Rightarrow \bigcirc(D_1 \vee L) \quad C_2 \Rightarrow \bigcirc(D_2 \vee \neg M)}{(C_1 \wedge C_2)\sigma \Rightarrow \bigcirc(D_1 \vee D_2)\sigma} \qquad \frac{C_1 \Rightarrow \bigcirc(D_1 \vee L) \quad D_2 \vee \neg M}{C_1\sigma \Rightarrow \bigcirc(D_1 \vee D_2)\sigma}$$
4. *(Step) factoring*.
$$\frac{C_1 \Rightarrow \bigcirc(D_1 \vee L \vee M)}{C_1\sigma \Rightarrow \bigcirc(D_1 \vee L)\sigma} \qquad \frac{(C \wedge A \wedge A) \Rightarrow \bigcirc D}{(C \wedge A) \Rightarrow \bigcirc D}$$
5. *Clause conversion*.
A step clause of the form $C \Rightarrow \bigcirc\mathbf{false}$ is rewritten into the *universal clause* $\neg C$.

The five rules of fine-grained step resolution are close enough to classical first-order resolution to use ordered first-order resolution as an implementation basis. In addition, for the realisation of the eventuality resolution rule a special resolution-based algorithm, called loop search algorithm, is used to find *loop formulae* for eventualities, which are combinations of left-hand sides of step clauses such that the right-hand sides of those step clauses imply that an eventuality can never be fulfilled from a certain point of time onwards; for details, see e.g. [2].

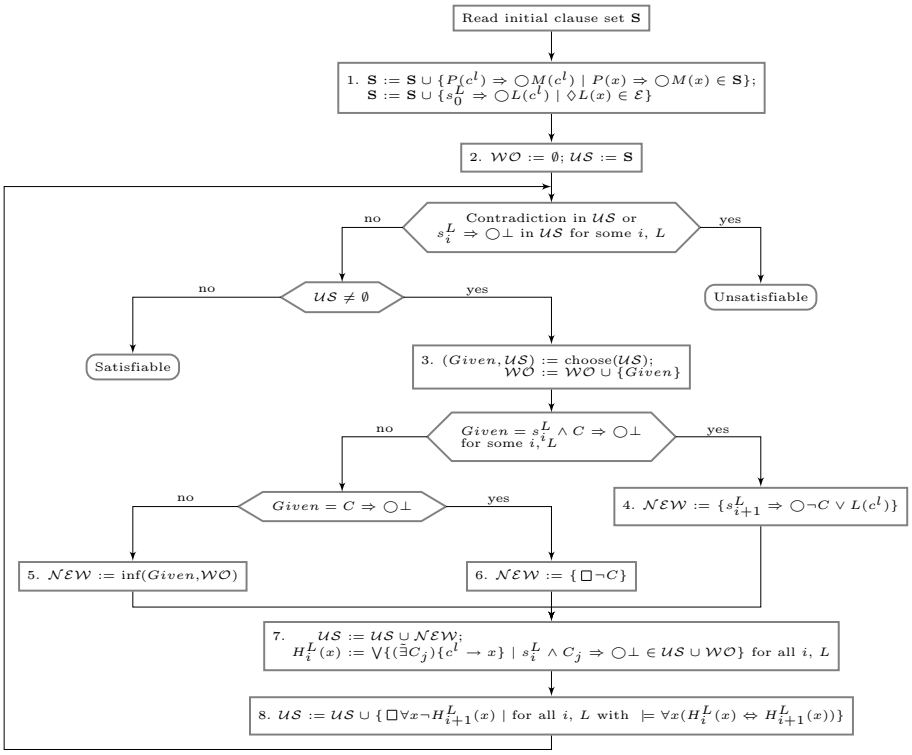


Figure 1: Architecture of a fair monodic temporal logic prover

To find loop formulae, the loop search algorithm constructs a sequence of sets containing universal and step clauses, which are then saturated under the first four rules of fine-grained step resolution. For each attempt to apply the eventuality resolution rule an instance of the loop search algorithm needs to be executed.

In the monodic first-order temporal logic prover TeMP [1], which is based on the implementation approach described above, a regular first-order prover is used as a black box which saturates sets containing the first-order translations of temporal clauses. The termination of these saturation processes is not guaranteed and partial results cannot be used until a saturation has finished. We devise a new architecture in this document which tries to overcome these limitations.

3 The Problem of Fairness

As the loop search algorithm requires sets of clauses to be saturated in order to find loop formulae, it is possible in practice that a theorem prover remains stuck in an infinite saturation during loop search. Moreover, it might happen that a partial loop formula, which is essential for a refutation, has been discovered by the algorithm but due to an infinite saturation this loop is never considered for the saturation with universal clauses.

One possible remedy for this problem would consist of performing the saturations for the different loop searches in parallel. One drawback of that approach would be that the loop searches would have to be restarted as soon as a new universal clause has been derived. Hence, we believe that such a parallel saturation procedure will not be that effective in practice. Instead, we propose a more sequential approach, shown in Figure 1, in which the loop search process is integrated into the general saturation.

We are currently trying to extend the new architecture with redundancy elimination methods such as tautology deletion and the removal of subsumed clauses.

References

[1] U. Hustadt, B. Konev, A. Riazanov, and A. Voronkov. TeMP: A temporal monodic prover. In D. A. Basin and M. Rusinowitch, editors, *IJCAR'04*, volume 3097 of *LNCS*, pages 326–330. Springer, 2004.

[2] 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.

Ontology Evolution in Law: (Extended Abstract)

Andrew Priddle-Higson[⋆]*, Alan Bundy*[⋆]*, Fiona McNeill*[⋆]

[⋆]*School of Informatics,*
University of Edinburgh
a.g.priddle-higson@sms.ed.ac.uk a.bundy,f.j.mcneill@ed.ac.uk

Burkhard Schafer[†]

[†]*Joseph Bell Centre for Forensic Statistics and Legal Reasoning,*
University of Edinburgh
b.schafer@ed.ac.uk

Abstract

We are exploring ontology evolution in the legal domain, by modeling the interpretation of open-textured statutory concepts in a legal case. This requires representing, and reasoning about, the mappings between a statutory context, a real-world context, and the various legal, social, and ethical contexts involved in a legal case. We are using contextual logics to formalise the process of legal theory construction.

The aim of our research is to explore computational models of ontology evolution in the legal domain[1]. We define ontology evolution as the changes that occur to a conceptualisation of a domain held by an agent, or a group of agents, through the interaction between the agent and the domain. The questions that our research is trying to address include: what changes can occur, why do these changes occur, and how do these changes occur? We are using the legal domain to explore ontology evolution since there are many well-documented, historical cases which can be used as case studies.

There are many forms of ontology evolution that occur in the law, with different ontologies being changed, for example: the change to the structure of the legal system when new courts are created, the changes to our commonsense understanding of the world which affect the law, changes to our values, or changes through the introduction of new laws. We are looking at the changes to the meaning of statutory concepts that occur as a result of their interpretation in specific legal cases. In particular, we are studying cases where the interaction between different legal systems can influence the interpretation of a statutory concept. For example, the interpretation of an enactment of a European Directive in one country might be influenced by the interpretation of an enactment in another European country.

An example of this form of ontology evolution exists in the case of *James Buchanan and Co Ltd v. Babco Forwarding and Shipping (UK) Ltd* (Buchanan v. Babco). The case involved the theft of a shipment of whisky in the U.K. whilst it was being transported from Scotland to Iran. Since the whisky was stolen in the U.K., the distillers (Buchanan) were liable for excise duty of £30,000 on it, the whisky was being sold for £7000. Had the whisky left the U.K., as planned, then no excise duty would have been levied. The distillers wanted compensation for the excise duty and the cost of the whisky from the carrier (Babco), since the whisky was only stolen due to the negligence of the carrier's lorry driver. The relevant Act of Parliament allowed compensation for the "current market price" of the goods, and for "other charges incurred in respect of carriage". The question in the legal case was whether either "current market price" or "other charges incurred in respect of carriage" included the excise duty.

The basic problem in statutory interpretation is that legal concepts are open-textured[2], there are borderline cases about which we are uncertain about whether the concept applies or not. In Buchanan v. Babco "current market price" does not obviously include excise duty since there are two notions of market: the domestic market (where the price would include excise duty) and the international market (where the price does not include excise duty). The problem of the open-texture of legal concepts is recognised within the AI and Law community [4] as a challenge to computational modeling of legal reasoning.

We view the problem of statutory interpretation in terms of the "bridge rules" between different contexts in a contextual logic [1]. This allows us to reason about how the interpretation of statutory concepts is affected by the various contexts in which they are interpreted. The importance of context to understanding how legal rules are used within a legal system is well known to lawyers [5].

Following work in AI and Law [4], we model legal reasoning as a process of theory construction. A lawyer must create a theory which relates the statutory context to the real-world context (the facts of the case) bearing in mind the legal, social

[1] there is also work on ontology evolution in physics in our research group [2]

[2] The classic example is "No vehicles in the park", the concept of vehicle is open-textured since it isn't clear whether it would include "vehicles" like ambulances or skateboards

and ethical contexts within which this interpretation occurs. This theory must also satisfy the goal of the lawyer, winning the case for their client.

For example, in Buchanan v. Babco there was an international context to the case. The relevant Act of Parliament was the U.K. enactment of an international convention. This led to debate about how (or even if) the French language version of the convention could be used to disambiguate the U.K. Act, and whether Foreign case law was relevant to the U.K. courts. So the problem of whether $current\text{-}market\text{-}price(whisky) = 37000$ or $current\text{-}market\text{-}price(whisky) = 7000$ might depend upon arguments about the interpretation of the French version of the convention.

The kinds of ontological changes to legal concepts that we are interested in include the change to the concept of "current market price" in Buchanan v. Babco. In this case, the lawyers recognised that there was a conflict between our commonsense intuition that there is a unique market price for a good and the existence of two distinct markets (domestic and international), with distinct prices, in the real-world situation. The judges refined the meaning of the statutory concept of "current market price" to be dependent upon the destination of the goods, and "other charges incurred in respect of carriage" to include the excise duty. These changes could be modeled in a similiar manner to ontology evolution in physics [2] and dynamic ontology repair [3].

Clearly, there are many problems to be solved in this work. Firstly, the reasoning and representation framework for contextual theory construction needs to be formalised. We have been looking at argumentation frameworks and abductive reasoning as a way of characterising the process of theory construction. Secondly, there is a significant commonsense-knowledge problem with creating the relevant background knowledge to model a legal case. Ontologies which capture the legal, social and ethical contexts of the law do not exist, it will be necessary to create these ontologies to explore specific cases. This presents a difficulty for the applicability and evaluation of the research, but the problem of ontology evolution is a significant and necessary research challenge which demands this exploration.

References

[1] Bouquet P. Benerecetti, M. and C. Ghidini. Contextual reasoning distilled. *Journal of Experimental Artificial Intelligence*, 12(3):279–305, 2000.

[2] A. Bundy and M. Chan. Towards ontology evolution in physics. In W. Hodges, editor, *Procs. Wollic 2008*. LNCS, Springer-Verlag, July 2008.

[3] F. McNeill and A. Bundy. Dynamic, automatic, first-order ontology repair by diagnosis of failed plan execution. *International Journal on Semantic Web and Information Systems*, 3(3):1–35, 2007.

[4] E.L. Rissland, K.D. Ashley, and RP Loui. AI and law: a fruitful synergy. *Artificial Intelligence*, 150(1-2):1–15, 2003.

[5] W. Twining and D. Miers. *How to Do Things With Rules (3rd Edition)*. Butterworths, 1991.

A Heuristic for Tableau Reasoning Based on Tree Decompositions of Knowledge Bases

Hilverd Reker⋆

⋆*School of Computer Science, The University of Manchester*
Oxford Rd, Manchester M13 9PL, UK
rekerh@cs.man.ac.uk

Abstract

Common-sense knowledge bases (KBs) typically possess certain structural characteristics which randomly generated ones do not. It has been conjectured that we can exploit these characteristics for improving the efficiency of automated reasoning about such KBs. This has been done for first-order resolution using *tree decompositions* of KBs, and for DPLL-based SAT solving as well. Here we propose a heuristic for *semantic tableaux* based on tree decompositions.

1 Introduction

Figure 1(i) shows an example of a common-sense knowledge base (KB) Γ, describing a few university-related concepts in the description logic $\mathcal{ALC}$. Observe that the first three statements of Γ are student-oriented, while the remaining ones are course-oriented. This suggests a "natural" decomposition of Γ into the two modules depicted in Fig. 1(ii).

$\mathsf{student} \sqsubseteq \mathsf{undergrad} \sqcup \mathsf{postgrad}$
$\mathsf{postgrad} \equiv \mathsf{student} \sqcap \mathsf{hasBSc}$
$\mathsf{student} \sqsubseteq \exists\, \mathsf{enrolledIn.course}$
$\mathsf{course} \sqsubseteq \exists\, \mathsf{taughtBy.person}$
$\mathsf{cs\text{-}course} \sqsubseteq \mathsf{course}$
$\mathsf{math\text{-}course} \sqsubseteq \mathsf{course}$.

(i) A common-sense KB Γ.

$\mathsf{student} \sqsubseteq \mathsf{undergrad} \sqcup \mathsf{postgrad}$
$\mathsf{postgrad} \equiv \mathsf{student} \sqcap \mathsf{hasBSc}$
$\mathsf{student} \sqsubseteq \exists \mathsf{enrolledIn.course}$

$\{\mathsf{student}, \mathsf{undergrad}, \mathsf{postgrad}, \mathsf{hasBSc}, \mathsf{enrolledIn}, \mathsf{course}\}$

$\mathsf{course} \sqsubseteq \exists \mathsf{taughtBy.person}$
$\mathsf{cs\text{-}course} \sqsubseteq \mathsf{course}$
$\mathsf{math\text{-}course} \sqsubseteq \mathsf{course}$

$\{\mathsf{course}, \mathsf{taughtBy}, \mathsf{person}, \mathsf{cs\text{-}course}, \mathsf{math\text{-}course}\}$

(ii) A decomposition of Γ.

Figure 1: Decomposing a common-sense knowledge base.

Below each module its *signature* is shown, which is the set of concept and role names occurring in it. Only one symbol, the concept name course, is shared by the two modules. Such decompositions can be performed automatically: in particular, *tree decomposition* [3] algorithms can be used to split up a large KB into a tree of modules, based on the signatures of the formulae in the KB. Tree decompositions are a well-studied concept in algorithmic graph theory. Many problems that are intractable for an arbitrary graph G can be solved in polynomial or even linear time, given a tree decomposition of G whose so-called *treewidth* is bounded by some fixed constant.

In [1; 4] tree decompositions are used to speed up resolution-based first-order logic reasoning. A related approach is taken in [2] for DPLL-based SAT solving. This suggests that, perhaps, *semantic tableaux* can somehow benefit from tree decompositions as well. In §2 we summarise the basic ideas of Amir et al.'s work. Then §3 addresses an adaptation of their method to tableaux.

2 Partition-Based Reasoning

Given a formula φ, let $\mathit{sig}(\varphi)$ denote the signature of φ. Knowledge bases are sets of formulae and will be denoted by Γ, Δ. We write $\mathit{sig}(\Delta)$ to mean $\bigcup_{\varphi \in \Delta} \mathit{sig}(\varphi)$. For our purposes, a *tree decomposition* is an undirected tree $T = \langle V, E \rangle$ in which every node is a KB (i.e. module), such that the following holds. For every pair $\Delta, \Delta' \in V$, if $P \in \mathit{sig}(\Delta)$ and $P \in \mathit{sig}(\Delta')$, then $P \in \mathit{sig}(\Gamma)$ for every node Γ on the (unique) path between Δ and Δ' in T. That is, if a symbol occurs in two nodes of the tree, it also occurs in all the nodes on the path between those nodes. A *tree decomposition of* Γ is a tree decomposition $T = \langle V, E \rangle$ such that $\bigcup V = \Gamma$.

Amir et al.'s approach is the following. Starting from a KB Γ, we first (somehow) compute a suitable tree decomposition $T = \langle V, E \rangle$ of Γ. We turn T into a directed tree by fixing a particular node as the *root* and directing all edges towards it. The reasoning algorithm then allows for two kinds of operations on this tree. First, resolution inferences may be performed locally in modules Δ, adding derived formulae to Δ. Second, formulae φ may be *propagated* from a node Δ to its parent Δ', but only if $\mathit{sig}(\varphi) \subseteq \mathit{sig}(\Delta')$.

This basic algorithm is shown to be refutation-complete, and Amir et al. claim that certain variations on it perform well in practise. Note that a tree decomposition of a KB essentially clusters together formulae with similar signatures into modules. Amir et al. have argued that the algorithm (only) works well if the decomposition consists of many, small modules, that are "loosely coupled" in terms of their signatures (i.e. share only a small number of symbols). Such kinds of decompositions can typically be obtained for common-sense KBs, as opposed to randomly generated ones.

3 Adaptation to Semantic Tableaux

We will use tree decompositions as *heuristics* to influence two kinds of decisions: which formula to operate on next, and which disjunct to explore first when branching. Given a (rooted, directed) tree decomposition $T = \langle V, E \rangle$, we begin by assigning to all nodes Δ an index $\mathit{idx}(\Delta)$ in an top-down, breadth-first, increasing manner: cf. Fig. 2. Such an indexing causes T to satisfy the following property for all $\Delta \in V$:

$$\begin{array}{l}\text{If } P \in \mathit{sig}(\Delta) \text{ and } P \notin \mathit{sig}(\mathit{parent}(\Delta)),\\ \text{then } P \notin \mathit{sig}(\Gamma) \text{ for all } \Gamma \in V \text{ with } \mathit{idx}(\Gamma) < \mathit{idx}(\Delta)\ .\end{array} \qquad (*)$$

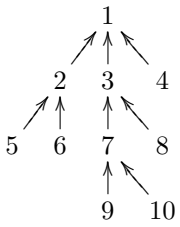


Figure 2: Indexing the nodes of the tree decomposition.

Our heuristics, given next, attempt to exploit this property.

1. When selecting the next formula to be operated on, try to choose one with an index that is as high as possible. That is, we start processing formulae at the leaves of the tree decomposition, and then gradually work our way up.

2. When branching on a disjunction $\varphi_1 \vee \varphi_2$ from a node Δ, first explore that disjunct φ_i $(i = 1, 2)$ for which the size of $\mathit{sig}(\varphi_i) \cap \mathit{sig}(\mathit{parent}(\Delta))$ is smallest.

The intuition behind this is the following. Suppose we are building a tableau and are about to add a disjunct φ from a module Δ below the current branch. It follows from $(*)$ that any literal L in φ involving a symbol $P \in \mathit{sig}(\varphi)$ that is not in $\mathit{sig}(\mathit{parent}(\Delta))$ will either cause a clash soon, or not at all. The former may happen if L clashes with a literal already on the current branch, or if another formula in Δ causes a clash later on. The latter follows from the fact that once we start processing only modules with an index lower than $\mathit{idx}(\Delta)$, no formulae in those modules will contain P anymore.

While computing a tree decomposition is expensive, we can use the same decomposition for query answering over a static KB. These heuristics have been implemented as part of an $\mathcal{ALC}$ reasoner, with encouraging initial results [5]. However, the experiments are preliminary, as it remains to be investigated how the proposed heuristics interact with, and can be combined with, the many existing tableau reasoner optimisations.

References

[1] E. Amir and S. McIlraith. Solving satisfiability using decomposition and the most constrained subproblem. Proc. Workshop on Theory and Applications of Satisfiability Testing, 2001.

[2] P. Bjesse, J. Kukula, R. Damiano, T. Stanion, and Y. Zhu. Guiding sat diagnosis with tree decompositions. SAT 2003, volume 2919 of LNCS, 2004.

[3] H. L. Bodlaender. A tourist guide through treewidth. Technical Report RUU-CS-92-12, Department of Information and Computing Sciences, Utrecht University, 1992.

[4] B. MacCartney, S. McIlraith, E. Amir, and T. Uribe. Practical partition-based theorem proving for large knowledge bases. In *Proceedings of the Nineteenth International Conference on Artificial Intelligence (IJCAI-03)*, pages 89–98, Acapulco, Mexico, August 2003.

[5] Hilverd Reker. A Tree Decomposition Approach to Automated Reasoning. Master's thesis, Rijksuniversiteit Groningen, the Netherlands, August 2007. `http://www.cs.man.ac.uk/~rekerh/msc-thesis/`.

On the Correspondence between Hypersequent and Labelled Calculi for Intermediate Logics

Robert Rothenberg[⋆]

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

Abstract

We show a correspondence between hypersequent and labelled sequent calculi. In particular, we show that for some intermediate logics, structural rules from a hypersequent calculus correspond with labelling rules from a labelled calculus. These labelling rules reflect the Kripke semantics of the underlying logics. We note that this correspondence may be useful for automated reasoning, as it allows one to separate the formalism used as a basis for an underlying implementation from the formalism that the user works in.

1 Introduction

Hypersequent calculi were first introduced independently by [14] and [1] as an extension of sequent calculi suitable for some non-classical logics. This formalism allows for the development of *analytic* calculi for a wide-range of logics—that is, cut-free calculi with the same set of logical constants as the logics they are intended to be proof-systems for. In recent years, hypersequent calculi have been used to develop the proof theory of several intermediate [2; 3; 4] and fuzzy logics [7; 11].

Labelled sequent calculi, apparently first introduced by [10], are another extension of sequent calculi where formulae are annotated with labels which correspond with some *semantic* interpretation, and the sequents may include other kinds of formulae that indicate the relationship between labels (e.g., accessibility relations in a Kripke frame). Labelled formalisms—discussed in general by [8]—are noted for allowing various non-classical logics to be embedded in a classical system—in particular cut-free and contraction-free systems, such as [13; 12].

Our work shows that these formalisms are in some cases essentially the same: we can think of labels as names for components, and accessibility relations between labels as indicators of how components are related in terms of proof (or counter-model) search. This seems to be a part of the folklore of hypersequents, but aside from the relationship between formalisms for the modal logic $\mathbf{S5}$ discussed in [2], [6] and [15], we are unaware of any work which discusses this correspondence.

What makes this correspondence of interest for automated reasoning is that it allows one to separate the underlying formalism used in an implementation from the formalism that the user works with—making the latter a choice for the user.

2 Correspondence for Intermediate Logics

The correspondence between hypersequent and simply labelled sequent calculi (calculi with no accessibility relations) is straightforward to show—the only complication being formalising a notion of *equivalence module permutation of labels*. Extending this correspondence to relational calculi such as $\mathbf{G3I}^*$ [5; 12] is work in progress. We are interested in devising automated methods of translating between rules of a simply labelled sequent calculus and relational sequent calculi, and in particular ways which allow us to demonstrate the admissibility of rules corresponding to properties of intuitionistic Kripke frames such as monotonicity, reflexivity and transitivity. Initial work seems to indicate a connection between such rules and the presence of structural rules such as contraction in the simply labelled sequent calculus.

We note the following correspondence between structural hypersequent rules that can be added to a hyperextension of a calculus for intuitionistic logic ($\mathbf{Int}$) and labelling rules from $\mathbf{G3I}^*$ for various intermediate logics is shown in Table 1.

3 Future Work

We plan to extend the correspondence between hypersequents and labelled sequents to other labelled formalisms, such as connection-based systems [16], or tree sequents [9]. Existing work connecting hypersequents and display sequents [17] may also be considered.

Logic	Distinguishing Axiom	Hypersequent Rule	Labelling Rule
LQ	$\neg\neg A \vee \neg A$	$\dfrac{\mathcal{H} \mid \Gamma_1, \Gamma_2 \Rightarrow \Delta}{\mathcal{H} \mid \Gamma_1 \Rightarrow \Delta \mid \Gamma_2 \Rightarrow \Delta}$	$\dfrac{x \leq w, y \leq w, z \leq x, z \leq y, \Sigma, \Gamma \Rightarrow \Delta}{z \leq x, z \leq y, \Sigma, \Gamma \Rightarrow \Delta}$
LC	$(A \supset B) \vee (B \supset A)$	$\dfrac{\mathcal{H} \mid \Gamma_1, \Gamma_2 \Rightarrow \Delta_1 \quad \mathcal{H} \mid \Gamma_1, \Gamma_2 \Rightarrow \Delta_2}{\mathcal{H} \mid \Gamma_1 \Rightarrow \Delta_1 \mid \Gamma_1 \Rightarrow \Delta_2}$	$\dfrac{x \leq y, \Sigma, \Gamma \Rightarrow \Delta \quad y \leq x, \Sigma, \Gamma \Rightarrow \Delta}{\Sigma, \Gamma \Rightarrow \Delta}$
Cl	$A \vee \neg A$	$\dfrac{\mathcal{H} \mid \Gamma_1, \Gamma_2 \Rightarrow \Delta_1, \Delta_2}{\mathcal{H} \mid \Gamma_1 \Rightarrow \Delta_1 \mid \Gamma_2 \Rightarrow \Delta_2}$	$\dfrac{x \leq y, y \leq x, \Sigma, \Gamma \Rightarrow \Delta}{\Sigma, \Gamma \Rightarrow \Delta}$

Table 1: Some extensions of **Int** and their corresponding hypersequent and labelling rules.

Acknowledgements

We are grateful to Roy Dyckhoff and Sara Negri for providing us with a copy of their paper [5], and to Greg Restall for discussions about work in [15] which inspired some of this work.

References

[1] A. Avron. A constructive analysis of RM. *J. of Symb. Log.*, 52(4):939–951, 1987.

[2] A. Avron. The method of hypersequents in the proof theory of propositional non-classical logics. In W. Hodges, M. Hyland, C. Steinhorn, and J. Truss, editors, *Logic: Foundations to Applications*. Oxford Science Pub., 1996.

[3] A. Ciabattoni and M. Ferrari. Hypertableau and path-hypertableau calculi for some families of intermediate logics. In *TABLEAUX 2000, Proceedings*, volume 1847 of *LNCS*, pages 160–175, 2000.

[4] A. Ciabattoni and M. Ferrari. Hypersequent Calculi for some Intermediate Logics with Bounded Kripke Models. *J. Log. Comput.*, 11(2):283–294, 2001.

[5] R. Dyckhoff and S. Negri. Proof analysis in intermediate propositional logics. Unpublished Manuscript, 2005.

[6] M. Fitting. Introduction. In M. D'Agostino, D.M. Gabbay, R. Hähnle, and J. Posegga, editors, *Handbook of Tableau Methods*, pages 1–45. Kluwer, 1999.

[7] D. Gabbay, G. Metcalfe, and N. Olivetti. Hypersequents and fuzzy logic. *Revista de la Real Academia de Ciencias*, 98(1):113–126, 2004.

[8] D.M. Gabbay. *Labelled Deductive Systems*, volume 1. Clarendon P., Oxford, 1996. ISBN 0-19-853833-2.

[9] R. Ishigaki and K. Kikuchi. Tree-sequent methods for subintuitionistic predicate logics. In *TABLEAUX 2007*, volume 4548 of *LNCS*, pages 149–164. Springer, 2007.

[10] S. Kanger. *Provability in Logic*, volume 1 of *Stockholm Studies in Philosophy*. Almquist & Wiksell, Stockholm, 1957.

[11] G. Metcalfe and F. Montagna. Substructural fuzzy logics. *J. Symb. Log.*, 72(3):834–864, 2007.

[12] S. Negri. Proof analysis in non-classical logics. In *Logic Colloquium 2005*, pages 107–128. Cambridge U.P., 2007.

[13] S. Negri. Proof analysis in modal logic. *J. Phil. Logic*, 34(5–6):507–544, 2005.

[14] G. Pottinger. Uniform cut-free formulations of T, S4 and S5 (abstract). *J. Symb. Logic*, 48(3):900, 1983.

[15] G. Restall. Proofnets for S5: sequents and circuits for modal logic. In *Logic Colloquium 2005*. Cambridge U.P., 2007.

[16] A. Waaler. Connections in nonclassical logics. In J.A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, volume II, pages 1487–1578. Elsevier, 2001.

[17] H. Wansing. Translation of hypersequents into display sequents. *Logic Journal of the IGPL*, 6(5):719–733, 1998.

Reasoning on Abstract Matrix Structures

Alan P. Sexton and Volker Sorge
School of Computer Science, University of Birmingham
A.P.Sexton|V.Sorge@cs.bham.ac.uk, www.cs.bham.ac.uk/~aps|~vxs

Matrices are often presented with symbolic dimensions and with a mixture of terms and ellipsis symbols to describe their structure. Such a representation effectively determines entire classes of matrices with respect to their common structural properties. When reasoning with such abstract matrices, one is often only interested in the interaction of structural properties of matrix classes under arithmetic operations, while ignoring the actual content of the matrices involved. For example, when showing that the product of two upper triangular matrices is an upper triangular matrix, one shows this property by considering the interaction of the zero and non-zero regions, while ignoring the actual composition of the latter. While this type of reasoning is everyday mathematical practice there is little automated support for this.

We presenthave developed an algebraic encoding of abstract matrices as linear combinations of basis functions that enables us to define addition and multiplication algorithms. Since recovering structural properties from the results of these operations is non trivial we have designed a rewrite system that normalises the results to a form from which the structural properties can be directly read. We can thus regain the resulting abstract matrix, which provides an effective procedure for reasoning about structural properties of classes of abstract matrices under arithmetic operations. The work is based on [1; 2].

Basis Function

Abstract matrices consist of three types of regions: single terms, single lines or ellipses, and closed convex polygons. These can all be represented via intersections of half planes that corresponds to the four different possible ellipses in the abstract matrix: vertical, horizontal, diagonal and anti-diagonal, the latter two at $\pm 45^\circ$ angles. Each half plane constrains the indices of the region. In order to capture the idea of half plane constraints algebraically, we define a basis function $\sigma(x,y) = \begin{cases} 1 & \text{if } x \leqslant y \\ 0 & \text{otherwise} \end{cases}$ for $x, y \in \mathbb{N}$. We often write $\sigma_{x,y}$ as shorthand and can define the complement of $\sigma_{x,y}$ to be $\overline{\sigma_{x,y}} = \sigma_{y,x-1}$. As example consider:

$$M = \begin{bmatrix} a_{11} \cdots a_{1n} & & \\ \ddots \ \vdots & \mathbf{0} & \\ a_{nn} & & \\ & b_{11} \cdots b_{1m} & \\ \mathbf{0} & \ddots \ \vdots & \\ & b_{mm} & \end{bmatrix} \begin{array}{l} = [\sigma_{1,i}\sigma_{j,n}\sigma_{i,j}a_{ij} + \sigma_{n+1,i}\sigma_{j,n+m}\sigma_{i,j}b_{i-n,j-n}]_{i,j}^{m+n,m+n} \\ = [\quad \Gamma_1 T_1 \qquad + \qquad \Gamma_2 T_2 \quad]_{i,j}^{m+n,m+n} \\ = [\quad M_{ij} \quad]_{i,j}^{m+n,m+n} \end{array} \tag{1}$$

Here $[M_{ij}]_{i,j}^{m+n,m+n}$ denotes a matrix of entries indexed by variables i, j each ranging from 1 to $m+n$. We write regions as $\Gamma_i T_i$, where Γ_i is a product of σ terms and T_i is a product of matrix entries. If $\Gamma = \sigma_1\sigma_2 \ldots \sigma_n$, then its complement is defined as $\overline{\Gamma} = \overline{\sigma_1\sigma_2 \ldots \sigma_n} = \overline{\sigma_1} + \overline{\sigma_2}\sigma_1 + \overline{\sigma_3}\sigma_1\sigma_2 + \cdots + \overline{\sigma_n}\sigma_1\sigma_2 \ldots \sigma_n$

Abstract Matrix Addition

The reason why we can not analyse the sum of two abstract matrices immediately is that, while the regions of an individual matrix are disjoint, pairs of regions drawn from different matrices are not so in general. We overcome this by decomposing the regions of each matrix into compatible disjoint regions during the addition.

$$A = \left[\left(\Gamma_1^A T_1^A + \cdots + \Gamma_1^A T_R^A\right)\right]_{i,j}^{m,n} \qquad B = \left[\left(\Gamma_1^B T_1^B + \cdots + \Gamma_S^B T_S^B\right)\right]_{i,j}^{m,n} \tag{2}$$

where Γ_l^A,Γ_l^B indicates that the respective Γ term originates from matrix A or B, respectively. We define the abstract matrix addition algorithm ADD as

$$\mathsf{ADD}(A,B) := \left[\begin{array}{l} \Gamma_1^A\Gamma_1^B(T_1^A + T_1^B) + \Gamma_1^A\Gamma_2^B(T_1^A + T_2^B) + \cdots + \Gamma_R^A\Gamma_S^B(T_R^A + T_S^B) + \\ \Gamma_1^A\overline{\Gamma_1^B} \cdots \overline{\Gamma_S^B}T_1^A + \Gamma_2^A\overline{\Gamma_1^B} \cdots \overline{\Gamma_S^B}T_2^A + \cdots + \Gamma_R^A\overline{\Gamma_1^B} \cdots \overline{\Gamma_S^B}T_R^A + \\ \Gamma_1^B\overline{\Gamma_1^A} \cdots \overline{\Gamma_R^A}T_1^B + \Gamma_2^B\overline{\Gamma_1^A} \cdots \overline{\Gamma_R^A}T_2^B + \cdots + \Gamma_S^B\overline{\Gamma_1^A} \cdots \overline{\Gamma_R^A}T_S^B \end{array}\right]_{i,j}^{m,n} \tag{3}$$

Abstract Matrix Multiplication

Our multiplication algorithm is based on the standard dot product of rows with columns. However, this is carried out pairwise on regions, and the resulting terms, for any particular cell of the product matrix, may be from intersecting or overlapping regions. To end up with a disjoint sum of regions, we need to deal with these overlaps in a manner similar to the case of addition, via calculating intersections and differences of regions. Letting A, B be as in (2), we define the abstract matrix multiplication algorithm MULT as:

$$\mathsf{MULT}(A,B) := \left[\sum_{k=1}^{p} A_{i,k}B_{k,j}\right]_{i,j}^{m,n} = \left[\sum_{k=1}^{p}\left(\Gamma_1^A\Gamma_1^B T_1^A T_1^B + \Gamma_1^A\Gamma_2^B T_1^A T_2^B + \cdots + \Gamma_R^A\Gamma_S^B T_R^A T_S^B\right)\right]_{i,j}^{m,n} \tag{4}$$

$$= \left[\sum_{k=1}^{p}\left(\widehat{\sum_{I\subseteq\mathcal{N}}}\left(\widehat{\prod_{q\in I}}\Gamma_q\right)\left(\widehat{\prod_{q\in\mathcal{N}\setminus I}}\overline{\Gamma_q}\right)\left(\widehat{\sum_{q\in I}}T_q\right)\right)\right]_{i,j}^{m,n} = \left[\sum_{k=1}^{p}(\Gamma_1 T_1) + \sum_{k=1}^{p}(\Gamma_2 T_2) + \cdots + \sum_{k=1}^{p}(\Gamma_N T_N)\right]_{i,j}^{m,n} \tag{5}$$

Observe that in the formula above the sum symbol Σ is part of the term syntax, i.e., the actual terms in the matrix contain a syntactic summation. $\widehat{\sum}$ and $\widehat{\prod}$ on the other hand meta-syntactic summation and product operators, which we use for the simplification of (4).

In (4) the distinction between terms originating from A and B is no longer necessary, because the σ terms involving the k index variable no longer describe half-plane constraints. The resulting Γ_i regions are still convex but also contain overlap. To factor these overlapping regions into disjoint ones, we need to apply some elementary set manipulations. If there are N potentially overlapping regions, then these can be decomposed into $2^N - 1$ disjoint component regions. Each such component region is formed by constructing a region R as the intersection of some subset $\mathcal{R}$ of the regions, and removing any parts of R that are in any of the other regions, i.e., intersecting R with the intersection of the complements of the remaining non-$\mathcal{R}$ regions. Thus a single component region will have the form $\Gamma_{s_1}\ldots\Gamma_{s_m}\overline{\Gamma_{s_{m+1}}\ldots\Gamma_{s_N}}$, where S is an indexing of $\{1,\ldots,N\}$. This process is described by the formula on the left hand side of (5), while the result on the right hand side is the fully expanded version.

Normalisation

After addition and multiplication algorithm the resulting abstract matrices are generally no longer in a form from which we can easily infer the actual form and approximate content of the single regions. We therefore define a rewrite system that ensure to produce a desired *regular form* of an abstract matrix. We thereby define the *regular form* of $[M]_{i,j}^{m,n} = [\Gamma_1 T_1 + \cdots + \Gamma_k T_k]_{i,j}^{n,m}$ by requiring that the following holds:

(i) $\Gamma_l\Gamma_{l'} = 0$ for each l, l' in $1,\ldots,k$ with $l \neq l'$. ***(Disjointness)***

(ii) $\Gamma_l = \sigma_1\cdots\sigma_p$, for each l in $1,\ldots,k$ and $p \geq 0$. ***(Convexity)***

(iii) for each T_l, l in $1,\ldots,k$ there does not exist a $\sigma \in T_l$ such that $\sigma T_l = T_l$. ***(Γ-Structured)***

The rewrite system consists of four sets of rules that are applied in three different stages:

Stage 1: 2 Transitive Closure rules complete the partial order in a given Γ expression.

Stage 2: 2 Factoring rules pull single σs that are independent from a summation variable in front of a sum expression.

Stage 3: 2 Reduction rules reduce a Γ expression to a minimal form while retaining the maximum structural information on the overall partial order by removing all implied σ expressions.

Finally, 7 contraction rules can be applied during each stage to reduce the complexity of the entire operation, but always have to be exhaustively applied as well, before resuming the rewriting of the particular stage. We can then show for the combined system $\mathcal{R}$ =(ADD,MULT,NORM) that it terminates, is confluent and correct.

References

[1] A. P. Sexton and V. Sorge. Abstract matrices in symbolic computation. In *Proc. of ISSAC 2006*, p. 318–325. ACM Press, 2006.

[2] A. P. Sexton, V. Sorge, S. M. Watt. Abstract matrix arithmetic. *ORCCA Technical Report TR-08-01*, University of Western Ontario, 2008.

Towards Meta-Level Theory Formation

Pedro Torres and Simon Colton*
Department of Computing, Imperial College London, UK
180 Queen's Gate, Imperial College London, SW7 2AZ.
ptorres,sgc@doc.ic.ac.uk

Introduction

HR [1] is an Automated Theory Formation (ATF) system which has been described as a Descriptive Inductive Logic Programming system [4]. Given a set of initial concepts, each provided with a definition and a set of examples, HR forms a theory about them. HR uses a set of production rules to form new concepts and forms conjectures according to the examples it finds for new concepts.

We address here the question of whether a theory formation system can gain from analysing interesting established theories, before it attempts actual theory formation. The approach taken here is to improve the theory formation abilities of a theory formation system by allowing a prior learning stage where the system is allowed to consult existing well-established theories — so called *inspiring theories*. The aim of this project is therefore to test the hypothesis that theory formation can be improved by introducing a prior learning stage where inspiring theories can be consulted. We call this learning stage the *meta-learning stage* since the system is auto-configuring itself to perform a different learning task.

In a study involving algebraic structures [2], we extended our theory formation approach with a generic first-order production rule. By writing explicit definitions of first-order production rules, the user could add to the set of production rules used by HR. We extend this approach here, and we investigate how a search for sets of production rules could be carried out. If such a search can be efficiently implemented, this opens up the possibility of automatically generating production rule based theory formation systems tailored to a specific domain. Such tailoring will be enabled by the user presenting core background theory and interesting concepts derived from them, with our system able to generate production rule sets that produce the interesting concepts from the core ones, and hopefully additional interesting material.

First-order representation of production rules

Let $\{\tau_i\}_{i=1}^N$ be a collection of sets, which we will call *types*. We define a *concept* to be a relation between any n types. We call n the *arity* of the concept. We will consider here concepts of the form $\{(x_1, \ldots, x_n) \in \tau_1 \times \cdots \times \tau_n | \ \varphi\}$, where φ is some first-order logic formula for which the only free variables are $x_1, \ldots, x_n$ and any other variables are bounded. The above concept will be written as $[x_1, \ldots, x_n] : \varphi(x_1, \ldots, x_n)$. We write $\mathcal{C}$ for the set of all concepts.

A *production rule* is a function $\pi : \mathcal{D} \rightarrow \mathcal{C}$, where $\mathcal{D} \subset \mathcal{C}^q$, for some q, called the *arity* of π. A production rule can be defined by specifying the transformation it operates on the logical formulas defining the input concepts. Suppose we have $x \in \tau_1$, $y \in \tau_2$ and $z \in \tau_3$. We can define, for example, π as a function carrying out the mapping $([x, y] : \varphi(x, y), [y, z] : \varphi(y, z)) \mapsto [x, z] : \exists_y.(\varphi(x, y) \wedge \varphi(y, z))$.

From an initial set of background concepts $\{C_1, \ldots, C_k\}$, by recursively applying production rules with appropriate domains, we can define new concepts, e.g. $C = \pi_6(\pi_4(\pi_1(C_1, C_2), \pi_2(C_2, C_3)), \pi_5(\pi_3(C_4)))$. Given a set of initial concepts $\mathcal{C}$ and a set of production rules Π, we recursively define $\Pi^*(\mathcal{C})$ as the set of all concepts which: (i) are in $\mathcal{C}$ or (ii) can be written as $\pi(C_1, \ldots, C_n)$ where π is in Π and every C_i is in $\Pi^*(\mathcal{C})$. The set $\Pi^*(\mathcal{C})$ can be intuitively described as the set of all concepts that can be produced from the concepts in $\mathcal{C}$ by using the production rules in Π. If we fix $\mathcal{C}$, the set Π defines exactly which concepts are in the search space. The essential difference between language biases and sets of production rules is that the former usually let the user directly restrict the syntactical form of all the concepts formed, whereas the latter restrict the stepwise syntactical construction of new concepts from old ones.

One way of making conjectures in a production rule based system is to compare the concepts produced. Suppose we have two concepts of the same arity and types for which all the available examples which satisfy one also satisfy the other and vice versa. In these circumstances, it is possible to conjecture that they are equivalent. Similar situations may lead to implication and non-existence conjectures. The association rules that such a system may come up with are therefore all of the form $\left[\forall_{\vec{x}}.C_1(\vec{x}) \rightarrow C_2(\vec{x})\right]$, $\left[\forall_{\vec{x}}.C_1(\vec{x}) \leftrightarrow C_2(\vec{x})\right]$ or $\left[\forall_{\vec{x}}.\neg C_1(\vec{x})\right]$ where $\vec{x}$ is some set of variables and C_1 and C_2 are concepts in $\Pi^*(\mathcal{C})$.

It is possible to define a conjecture as a 0-arity concept. In this light, conjecture making techniques are simply particular production rules of arity 2 (they take two concepts) which yield concepts of arity 0 (i.e. conjectures). By using these

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

definitions, concepts and conjectures can be treated at the same level. Furthermore, we can define an order relation on the space of sets of production rules. Given a set of initial concepts, $\mathcal{C}$, and two sets of production rules, Π_1 and Π_2, we say that $(\Pi_1 \leq_{\mathcal{C}} \Pi_2)$ iff $\Pi_1^*(\mathcal{C}) \subseteq \Pi_2^*(\mathcal{C})$. In this case, Π_2 is said to be *richer* than Π_1.

Searching for production rules

The main idea we explore is that, instead of searching for concepts with a fixed set of production rules, we can start by analysing examples of previously formed (interesting) theories in a chosen domain and build a set of production rules tailored to that particular domain.

In order to perform the search for sets of production rules, we have defined three high-level algorithms: (i) the EXPANDER algorithm for computing $\Pi^*(\mathcal{C})$, up to some defined depth, from user-defined $\mathcal{C}$ and Π; (ii) the PARROT algorithm which takes a large set of formed concepts, $\mathcal{C}$, all ultimately constructed from some elementary concepts $\mathcal{C}_0$, and defines a set of production rules, Π_0, such that $\Pi_0^*(\mathcal{C}_0) \supseteq \mathcal{C}$; (iii) the ENRICHER algorithm, which takes a set of production rules and iteratively enriches this set.

EXPANDER is a simple algorithm which considers all legal sequential applications of production rules up to some prescribed depth. In the PARROT algorithm, each concept, C, in the user-given set of concepts, $\mathcal{C}$, can be expressed as a first order formula making use of some old concepts $\mathcal{C}_0(C)$. It is therefore possible to craft a production rule which takes the concepts $\mathcal{C}_0(C)$ and produces C. If this is done for all the concepts in $\mathcal{C}$, we get a set of production rules, Π_0, which can generate all of the concepts in $\mathcal{C}$ from the elementary concepts they are based on. Since we have put concepts and conjectures on the same level, the above reasoning means that, if the user includes conjectures (or, possibly, theorems) in the set of concepts $\mathcal{C}$, the production rule set given by PARROT will also be able to make those conjectures. If the set of concepts, $\mathcal{C}$, is very restricted, the production rule set output by PARROT may not be able to generate much more than $\mathcal{C}$ itself. If this is the case, we will want to enrich that set.

A simple way to enrich a set of production rules, Π, is to remove a production rule π and add two separate production rules π_1 and π_2 such that, for every C in the domain of π, we have $\pi(C) = \pi_1(\pi_2(C))$. In this example we have considered rules of arity 1 but this *decomposition* idea can be generalised to higher arities. Given a set of production rules, there is a considerable number of possible decompositions enriching that set. To facilitate the analysis of these decompositions we have defined a set of so called *primitive* production rules which are closely based on the syntax of first order logic. By using the primitive production rules, we can write different composite production rules using a common vocabulary and search for matching subparts.

The ENRICHER algorithm is given a set of production rules and performs a best-first search through all decompositions. It uses EXPANDER to compute the concepts which each particular decomposition adds and calculates a score for each decomposition based on measures defined by the user. For example, using this algorithm, the user can ask for an enrichment which favours concepts with higher applicability (i.e. higher proportion of which satisfy the concept).

In the presentation poster, we will give details of the three theoretical algorithms mentioned above and toy examples of their application. This will include the complete definition of the primitive production rules, which play an important role in the ENRICHER algorithm, and the analysis of some of their properties.

References

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

[2] P. Torres and S. Colton. Using model generation in automated concept formation. In *Proceedings of the Automated Reasoning Workshop*, 2006.

[3] P. Torres and S. Colton. Towards meta-level descriptive ILP. In *Proceedings of the 16th International Conference on Inductive Logic Programming*, 2006.

[4] S. Colton and S. Muggleton. Mathematical applications of Inductive Logic Programming. *Machine Learning (online first)* 2005.

A Refined Resolution Calculus for CTL

Lan Zhang *Ullrich Hustadt* *Clare Dixon* *

Department of Computer Science, University of Liverpool
Liverpool, L69 3BX, UK
{lan,ullrich,clare}@csc.liv.ac.uk

1 Introduction

Computational Tree Logic (CTL) is a branching-time temporal logic whose underlying model of time is a choice of possibilities branching into future. Here we present an improved, sound and complete clausal resolution calculus $\mathsf{R}_{\mathrm{CTL}}$ [4] based on an earlier calculus by Bolotov [2]. The calculus $\mathsf{R}_{\mathrm{CTL}}$ involves transformation to a normal form and the application of *step* and *eventuality* resolution rules dealing with constraints on next states and on future states, respectively. Compared with the earlier calculus [2], we have fewer eventuality resolution rules, which are the most costly rules of the calculus. Moreover, our calculus $\mathsf{R}_{\mathrm{CTL}}$ is designed in order to allow the use of classical first-order resolution techniques to implement the core rules of the calculus.

2 Normal form for CTL and clausal resolution calculus $\mathsf{R}_{\mathrm{CTL}}$

The calculus $\mathsf{R}_{\mathrm{CTL}}$ operates on formulae in a clausal normal form called Separated Normal Form for CTL, denoted by $\mathrm{SNF}_{\mathrm{CTL}}$. The language of $\mathrm{SNF}_{\mathrm{CTL}}$ 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)	$\mathbf{A}\Box(\bigwedge_{i=1}^{n} l_i \Rightarrow \mathbf{E}\bigcirc \bigvee_{j=1}^{k} m_{j\langle ind\rangle})$	($\mathbf{E}$ step clause)
$\mathbf{A}\Box(\mathbf{true} \Rightarrow \bigvee_{j=1}^{k} m_j)$	(global clause)	$\mathbf{A}\Box(\bigwedge_{i=1}^{n} l_i \Rightarrow \mathbf{A}\Diamond l)$	($\mathbf{A}$ sometime clause)
$\mathbf{A}\Box(\bigwedge_{i=1}^{n} l_i \Rightarrow \mathbf{A}\bigcirc \bigvee_{j=1}^{k} m_j)$	($\mathbf{A}$ step clause)	$\mathbf{A}\Box(\bigwedge_{i=1}^{n} l_i \Rightarrow \mathbf{E}\Diamond l_{\langle LC(ind)\rangle})$	($\mathbf{E}$ 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. 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 equisatisfiable set of $\mathrm{SNF}_{\mathrm{CTL}}$ clauses, a complete description of which can be found in [4]. The transformation rules are similar to those in [2], but modified to allow for global clauses.

The resolution calculus consists of two types of resolution rules, *step* resolution rules (SRES1 to SRES8) and *eventuality* resolution rules (ERES1 and ERES2). Compared with [2], we have four more step resolution rules due to the introduction of global clauses in $\mathrm{SNF}_{\mathrm{CTL}}$. 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.

$$\textbf{SRES6}\quad \frac{\mathbf{true} \Rightarrow C \vee l,\ Q \Rightarrow \mathbf{A}\bigcirc(D \vee \neg l)}{Q \Rightarrow \mathbf{A}\bigcirc(C \vee D)}$$

$$\textbf{SRES7}\quad \frac{\mathbf{true} \Rightarrow C \vee l,\ Q \Rightarrow \mathbf{E}\bigcirc(D \vee \neg l)_{\langle ind\rangle}}{Q \Rightarrow \mathbf{E}\bigcirc(C \vee D)_{\langle ind\rangle}}$$

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

where $\overline{P} \Rightarrow \mathbf{E}\bigcirc\mathbf{E}\Box l$ is not a single $\mathrm{SNF}_{\mathrm{CTL}}$ clause but represents a set of $\mathrm{SNF}_{\mathrm{CTL}}$ clauses which together imply $\overline{P} \Rightarrow \mathbf{E}\bigcirc\mathbf{E}\Box l$.

3 Implementation of the calculus $\mathsf{R}_{\mathrm{CTL}}$

To implement the calculus $\mathsf{R}_{\mathrm{CTL}}$ and the associated decision procedure for CTL we intend to adopt an approach analogous to that used in [3] to implement a resolution calculus for PLTL. A formal description of the approach and related proofs

*The first and second author are supported by EPSRC grant EP/D060451/1.

are presented in detail in [4]. The main idea is to represent each initial, global, and step clause as a first-order clause and to utilise ordered first-order resolution to emulate step resolution rules.

With every propositional atom p we uniquely associate a unary predicate symbol Q_p. Besides these predicate symbols we assume that our first-order vocabulary includes a countably infinite set of variables x, y, ..., a constant $\mathsf{0}$, a binary function symbol app, and for every $ind \in Ind$ a constant s_{ind}.

Let M be a CTL tree model consisting of an infinite set S of states and labelled edges connecting states. The first-order atom $Q_p(x)$ represents that p holds at the state $x \in S$, while $Q_p(\mathsf{0})$ represents that p holds at the root state $s_0 \in S$. Informally speaking, the first-order term $\mathsf{app}(y, x)$ represents an arbitrary successor state of the state x, while $\mathsf{app}(s_{ind}, x)$ represents the successor state $s_i \in S$ of the state x such that the edge from x to s_i is labelled with ind. Then the first-order atoms $Q_p(\mathsf{app}(y, x))$ and $Q_p(\mathsf{app}(s_{ind}, x))$ represent that p holds at states $\mathsf{app}(y, x)$ and $\mathsf{app}(s_{ind}, x)$, respectively. Finally, for a clause $C = (\neg)p_0 \vee \ldots \vee (\neg)p_n$, let $\lceil C \rceil(t)$, where t is a term, denote the first-order clause $(\neg)Q_{p_0}(t) \vee \ldots \vee (\neg)Q_{p_n}(t)$.

We are then able to represent every initial, global, $\mathbf{A}$ step and $\mathbf{E}$ step clause by a first-order clause: an initial clause $\mathbf{start} \Rightarrow C$ becomes $\lceil C \rceil(\mathsf{0})$; a global clause $\mathbf{true} \Rightarrow C$ becomes $\lceil C \rceil(x)$; an $\mathbf{A}$ step clause $P \Rightarrow \mathbf{A}\bigcirc C$ becomes $\lceil \neg P \rceil(x) \vee \lceil C \rceil(\mathsf{app}(y, x))$; an $\mathbf{E}$ step clause $p \Rightarrow \mathbf{E}\bigcirc q_{\langle ind \rangle}$ becomes $\lceil \neg P \rceil(x) \vee \lceil C \rceil(\mathsf{app}(s_{ind}, x))$. This then allows for all our step resolution rules to be emulated using ordered first-order resolution [1]. For example, an application of SRES6

$$\frac{\mathbf{true} \Rightarrow C \vee l,\ P \Rightarrow \mathbf{A}\bigcirc(D \vee \neg l)}{P \Rightarrow \mathbf{A}\bigcirc(C \vee D)}$$

can be emulated by the following inference using ordered resolution

$$\frac{\lceil C \rceil(x) \vee Q_l(x),\ \lceil \neg P \rceil(y) \vee \lceil D \rceil(\mathsf{app}(z, y)) \vee \neg Q_l(\mathsf{app}(z, y))}{\lceil \neg P \rceil(y) \vee \lceil C \rceil(\mathsf{app}(z, y)) \vee \lceil D \rceil(\mathsf{app}(z, y))}$$

where an ordering is used to restrict the application of resolution to literals $Q_l(x)$ and $\neg Q_l(\mathsf{app}(z, y))$. The remaining step resolution rules are emulated in the same way.

To implement the eventuality resolution rules in [4], we will need to augment a first-order theorem prover with an implementation of the CTL loop search algorithm described in detail in [4]. The computationally most costly part of the CTL loop search algorithm is again based on step resolution and we can take advantage of our implementation of step resolution via ordered first-order resolution.

4 Conclusions

Compared to the definition of $\mathrm{SNF}_{\mathrm{CTL}}$ in [2], we use an additional type of clauses, namely *global clauses*. One obvious advantage of allowing global clauses is that fewer clauses will be needed to represent a CTL formula in $\mathrm{SNF}_{\mathrm{CTL}}$ and fewer clauses will be generated by resolution. Second, global clauses also inevitably occur as a result of inferences in an implementation of step resolution. Thus, removing global clauses via a rewriting procedure as done in [2] would require additional implementation effort in this approach. In addition, the main disadvantage of the introduction of global clauses, namely the need for the additional step resolution rules that allow resolution with on global clauses, disappears, because all step resolution rules map onto the rules for ordered first-order resolution. Further, we also show that two eventuality resolution rules in [2] are redundant. Considering that the eventuality resolution rules are computationally very expensive, we gain a significant improvement here.

We prove the calculus $\mathsf{R}_{\mathrm{CTL}}$ is sound and complete in [4]. We also prove that a decision procedure based on $\mathsf{R}_{\mathrm{CTL}}$ is of the order EXPTIME.

Finally, we provide a new technique to implement step resolution rules and the algorithm for eventuality resolution rules with ordered first-order resolution. This makes our calculus useful from both a theoretical and a practical perspective.

References

[1] L. Bachmair and H. Ganzinger. Resolution theorem proving. In *Handbook of Automated Reasoning*, volume 1, chapter 2, pages 19–99. Elsevier, 2001.

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

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

[4] 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.