Joint Automated Reasoning Workshop and Deduktionstreffen

As part of the Vienna Summer of Logic – IJCAR

23-24 July 2014

Preface

For many years the British and the German automated reasoning communities have successfully run independent series of workshops for anybody working in the area of automated reasoning. Although open to the general public they addressed in the past primarily the British and the German communities, respectively. At the occasion of the Vienna Summer of Logic the two series have a joint event in Vienna as an IJCAR workshop. In the spirit of the two series there will be only informal proceedings with abstracts of the works presented. These are collected in this document. We have tried to maintain the informal open atmosphere of the two series and have welcomed in particular research students to present their work. We have solicited for all work related to automated reasoning and its applications with a particular interest in work-in-progress and the presentation of half-baked ideas.

As in the previous years, we have aimed to bring together researchers from all areas of automated reasoning in order to foster links among researchers from various disciplines; among theoreticians, implementers and users alike, and among international communities, this year not just the British and German communities.

Topics

Topics of interest include but are not limited to:

- Theorem proving in classical and non-classical logics
- Interactive theorem proving, logical frameworks, proof assistants, proof-planning
- Reasoning methods
  - Saturation-based, instantiation-based, tableau, SAT
  - Equational reasoning, unification
  - Constraint satisfaction
  - Decision procedures, SMT
  - Combining reasoning systems
  - Non-monotonic reasoning, commonsense reasoning,
  - Abduction, induction
  - Model checking, model generation, explanation
- Formal methods to specifying, deriving, transforming and verifying computer systems, requirements and software
- Logic-based knowledge representation and reasoning:
  - Ontology engineering and reasoning
  - Domain specific reasoning (spatial, temporal, epistemic,agents, etc)
- Logic and functional programming, deductive databases
- Implementation issues and empirical results, demos
- Practical experiences and applications of automated reasoning

For the Programme Committee:
Alexander Bolotov and Manfred Kerber

Programme Committee:

- Serge Autexier (DFKI)
- Bernhard Beckert (Karlsruhe Institute of Technology)
- Christoph Benzmüller (Freie Universität Berlin)
- Alexander Bolotov (University of Westminster) - chair
- Simon Colton (Department of Computing, Goldsmiths College, University of London)
- Louise Dennis (University of Liverpool)
- Clare Dixon (University of Liverpool)
- Jacques Fleuriot (University of Edinburgh)
- Ulrich Furbach (University of Koblenz)
- Jürgen Giesl (RWTH Aachen)
- Ullrich Hustadt (University of Liverpool)
- Dieter Hutter (DFKI GmbH)
- Reiner Hähnle (Technical University of Darmstadt)
- Mateja Jamnik (University of Cambridge)
- Manfred Kerber (University of Birmingham) - chair
- Ekaterina Komendantskaya (School of Computing, University of Dundee)
- Sebastian Rudolph (Technische Universitt Dresden)
- Renate A. Schmidt (University of Manchester)
- Viorica Sofronie-Stokkermans (MPI)
- Volker Sorge (University of Birmingham)

Table of Contents

Towards Usability Evaluation of Interactive Theorem Provers*

Bernhard Beckert Sarah Grebing Florian Böhl

Karlsruhe Institute of Technology (KIT)
{beckert, sarah.grebing, florian.boehl}@kit.edu

Abstract: The effectiveness of interactive theorem provers (ITPs) has increased in a way that the bottleneck in the interactive process shifted from effectiveness to efficiency. Proving large theorems still needs a lot of effort for the user interacting with the system. This issue is recognized by the ITP-communities and improvements are being developed. However, in contrast to properties like soundness or completeness, where rigorous methods are applied to provide evidence, the evidence for a better usability is lacking in many cases. Our contribution is the application of methods from the human-computer-interaction (HCI) field to ITPs. We report on the application of focus groups to evaluate the usability of Isabelle/HOL and the KeY system. We apply usability evaluation methods in order to a) detect usability issues in the interaction between ITPs and their users, and b) to analyze whether methods such as focus groups are applicable to the field of ITP.

1 Introduction

Motivation. The degree of automation of interactive theorem provers (ITPs) has increased to a point where complex theorems over large formalizations of real-world problems can be proven effectively. But even with a high degree of automation, user interaction is still required on different levels. On a global level, users have to find the right formalization and have to decompose the proof task by finding useful lemmas. On a local level, when automatic proof search for a lemma fails, they have to either direct proof search or understand why no proof can be constructed and fix the lemma or the underlying formalization. As the degree of automation increases, the number of interactions decreases. However, the remaining interactions get more complex as ITPs are applied to more complex problems. Thus, the time has come to shift the focus from the effectiveness of automated proof search to the efficiency of user interaction by increasing usability of ITPs and providing a better user experience. Soundness of ITPs can be formally proven and the effectiveness of automated proof search can, e.g., be measured with benchmarks. But, in the area of usability of ITPs, objective and reproducible experiments are rare and often replaced by anecdotal evidence or handwaving. Here, we report on two experiments applying the focus group method to two different ITPs: The tactical theorem prover Isabelle/HOL [11] and the interactive program verification system KeY [3]. The focus group method is a structured group discussion guided by a moderator. The main goal of our experiments was twofold: Firstly, on the "meta-level," we wanted to see if focus groups can be used to evaluate the usability of ITPs and what impact the specific characteristics of ITPs have on the setup and the results of focus groups. Secondly, on the concrete level, our aim was to compare the two ITP systems w.r.t. their usability.

*The work presented here is part of the project Usability of Software Verification Systems within the BMBF-funded programme Software Campus.

Related Work There have been some attempts to address the usability of theorem provers with structured usability evaluations, e.g., applying questionnaire based methods [12, 4, 2, 7], qualitative methods such as co-operative evaluations [6] or analyzing recordings of user errors [1].

2 Evaluation

Evaluation Method. Focus group discussions are a qualitative method to explore opinions of users about specific topics or products, e.g., in market research. In the field of human-computer interaction (HCI) they are used to explore user perspectives on software systems and their usability in an early stage of the usability engineering process [5, 10]. Based on their results, (prototypical) mechanisms for improving usability can be developed, which can then be evaluated with methods such as usability testing and user questionnaires to quantitatively measure increases in usability. While focus groups explore the subjective experience of users, they are designed to eliminate experimenter's bias and to provide more objective results. The number of participants required (five to ten participants) to get significant results is much smaller than for quantitative evaluations, which makes focus groups well-suited for the relatively small user base of ITPs. The duration of the discussion groups is around one to two hours and it is guided by a moderator who uses a script to structure the discussion. Focus groups have three phases: Recruiting participants, performing the discussion and post-processing. In the following we will briefly describe how we conducted the focus groups for Isabelle/HOL and the KeY system.

Participants. The participants, mostly Master or PhD students, were recruited using personal contacts. We ensured that each group included novice, intermediate, and expert users in different proportions. The KeY group had seven and the Isabelle group five participants.

Script for the discussions. The main questions and tasks in the script were the same for both discussions as we

wanted to compare the results. Adaptations of the questions and mock-ups to the specifics of the two systems were the main differences. The full scripts for our experiments are available at `formal.iti.kit.edu/grebing/SWC`. As warm-up task, we asked about typical application areas of the systems and about their strengths and weaknesses related to the proof process. In the main part of the discussion, we had two topics: (1) Support during the proof process and (2) Mechanisms for understanding proof states. For the cool-down task, we asked the participants to be creative and imagine their ideal interactive proof system.

Topic 1: Support during the proof process. In this topic we address the question "Does the tool give sufficient support during the proof process?". We divided the discussion for this topic into two parts, namely the global proof process (finding the right formalization and decomposing the proof task) and the local proof process (proving a single lemma or theorem). For each part, participants were asked to describe the typical proof process and discuss where the prover gives support and where support is missing. Time-consuming actions should be pointed out as well.

Topic 2: Mechanisms for understanding proof states. For the second topic, we initiated a more focused discussion by presenting mock-ups for mechanisms not yet built into the tools. This included (a) a mechanism for tracing formulas, terms, and variables that are generated during proof construction back to the original proof goal (for both tools), (b) a visual support for proof management that shows which lemmas contribute to a proof (for Isabelle), and (c) a mechanism for highlighting local changes between two adjacent nodes in the proof tree (for KeY). Thus, we used focus groups to get a first assessment of new features.

For all presented mechanisms we had the same line of action and questions. First, the participants were asked to describe what they think the mechanism does (i.e., the mechanism was not explained by the moderator). This was done to avoid bias introduced by the moderator and to see if the mechanism is intuitive. Then, the participants should express opinion on the usefulness of the mechanism. The planned duration for both groups was 2 hours. Due to lively discussions, the actual duration was 2.5 resp. 3 hours.

Analysis and Results We transcribed the recorded discussion to use Qualitative Content Analysis [9] to analyze and structure the results to draw conclusions from the evaluation. We gained insight into strengths and weaknesses of the two systems, which mostly can be generalized for ITPs, e.g., missing comprehension about the automatic strategies of the tools. Also technical issues, which are annoying for the user and in their opinion compromising for the efficiency, were mentioned, e.g., unstable loading mechanisms or a slow user interface. These point out where the systems could be improved in particular. By showing mock-ups of improvements we gained lively feedback and opinions about the presented mechanisms, allowing us to improve our mechanisms and prototypically implement them in the future. The full details of the presented mechanisms, the evaluation and the results will be presented in the poster.

3 Conclusion and Future Work

Our experiments show, that focus groups can be used to get insight into ITPs and explore where to improve the systems in order to ease the interactive verification process for the user. The first results already show, that focus groups can help to determine which mechanisms might or might not satisfy the needs of the users and how to improve the presented mechanisms. A full analysis and interpretation of the recorded and transcribed material is currently being done. This will result in a detailed report on desirable features for interactive theorem provers. The mechanisms that attracted interest during the discussions need to be further developed and prototypically implemented. By using usability testing we ensure that the mechanisms suit the user's needs and evaluate the impact on the usability of the systems. We will apply the User Experience Questionnaire method [8] to assess the usability of the KeY system quantitatively. Here, we will determine, whether such general-purpose questionnaires are helpful for evaluating the usability of ITPs, or whether more adaptable solutions are needed.

References

[1] J. S. Aitken and T. F. Melham. An analysis of errors in interactive proof attempts. *Interacting with Computers*, 12(6):565–586, 2000.

[2] B. Beckert and S. Grebing. Evaluating the usability of interactive verification system. In *Proceedings of COMPARE 2012*, CEUR Workshop Proceedings 873, 2012.

[3] B. Beckert, R. Hähnle, and P. H. Schmitt, editors. *Verification of Object-Oriented Software: The KeY Approach*. LNCS 4334. Springer-Verlag, 2007.

[4] J. Cheney. Project report – theorem prover usability. Technical report, 2011.

[5] X. Ferré, N. J. Juzgado, H. Windl, and L. L. Constantine. Usability basics for software developers. *IEEE Software*, 18(1):22–29, 2001.

[6] M. Jackson, A. Ireland, and G. Reid. Interactive proof critics. *Formal Aspects of Computing*, 11(3):302–325, 1999.

[7] G. Kadoda, R. Stone, and D. Diaper. Desirable features of educational theorem provers: A Cognitive Dimensions viewpoint. In *Proceedings of the 11th Annual Workshop of the Psychology of Programming Interest Group*, 1996.

[8] B. Laugwitz, T. Held, and M. Schrepp. Construction and evaluation of a user experience questionnaire. In A. Holzinger, editor, *HCI and Usability for Education and Work*, LNCS 5298, pages 63–76. Springer, 2008.

[9] P. Mayring. *Einführung in die qualitative Sozialforschung – Eine Anleitung zu qualitativem Denken (Introduction to qualitative social research)*. Psychologie Verl. Union, 1996.

[10] J. Nielsen. *Usability Engineering*. Morgan Kaufmann, 1993.

[11] T. Nipkow, L. C. Paulson, and M. Wenzel. *Isabelle/HOL: A Proof Assistant for Higher-Order Logic*. LNCS 2283. Springer, 2002.

[12] V. Vujosevic and G. Eleftherakis. Improving formal methods' tools usability. In *2nd South-East European Workshop on Formal Methods*. South-East European Research Centre (SEERC), 2006.

Combined Reasoning with Sets and Aggregation Functions

Markus Bender

Universität Koblenz-Landau
Institut für Informatik
Universitätsstrasse 1
56070 Koblenz
mbender@uni-koblenz.de

Abstract: We developed a method that allows to check the satisfiability of a formula in the combined theories of sets and the bridging functions card, sum, avg, min, max by using a prover for linear arithmetic. Since abstractions of certain verification tasks lie in this fragment, this method can be used in program verification.

1 Introduction

In [2, 1], Kuncak et al. give a sound, complete and terminating method to check the satisfiability of formulae in the theory of Boolean algebras and Presburger arithmetic (BAPA), which is equivalent to the combined theory of sets and cardinalities. They reduce the given problem to a problem in pure Presburger arithmetic and then use a prover for Presburger arithmetic. We extended this method, such that in addition to considering BAPA, we can deal with additional bridging functions between the theory of sets and the theory of Presburger arithmetic, namely the functions sum that calculates the sum of all elements of a set, avg that calculates the average of all elements of a set and min and max that calculate the minimal and maximal element of a set, respectively.

This variety of additional bridging functions allows us to deal with a broader range of verification tasks.

2 Preliminaries

As we need the concepts of atomic sets and atomic decomposition, these are introduced as follows:

For n given sets $\mathsf{A}_1, \ldots, \mathsf{A}_n$, there are 2^n *atomic sets* $\mathcal{S}_0, \ldots, \mathcal{S}_{2^n-1}$ that are defined as follows:

$$\mathcal{S}_i = \bigcap_{j=0}^{n} \mathsf{A}_j^{d(j,i)}, \text{ for } 0 \leq i < 2^n.$$

where $d(i,j)$ is the j-th binary digit of i and A^0 is defined as A and A^1 is defined as the complement of A, $\overline{\mathsf{A}}$.

Due to the construction of the atomic sets, all $\mathcal{S}_i$ are disjoint.

We define that an atomic set $\mathcal{S}_i = \bigcap_{j=0}^{n} \mathsf{A}_j^{d(j,i)}$, for $0 \leq i < 2^n$ is *contained* in a set A if and only if there is a j such that $\mathsf{A}_j^{d(j,i)} = \mathsf{A}$.

Each given set A_i and its complement $\overline{\mathsf{A}_i}$ can now be represented uniquely as *atomic decomposition*, which is the union of all atomic sets that are contained in the set.

As the developed approach relies on the method developed by Kuncak et al. [3], we give a short introduction to the latter.

3 Boolean Algebra and Presburger Arithmetic

The method of Kuncak et al. [2, 1] follows these 5 steps: a) replace equations of sets $\mathsf{A}_1 \approx \mathsf{A}_2$ with subset relations in both directions $\mathsf{A}_1 \subseteq \mathsf{A}_2 \wedge \mathsf{A}_2 \subseteq \mathsf{A}_1$, b) replace every subset relation between two sets $\mathsf{A}_1 \subseteq \mathsf{A}_2$ with a relation on the cardinality $\mathsf{card}(\mathsf{A}_1 \cap \overline{\mathsf{A}_2}) \approx 0$, c) create all atomic decompositions for the sets appearing in the given formula and represent all set expressions by equivalent unions of atomic sets, d) replace cardinality of unions with sum of cardinalities of atomic sets, e) purify by introducing new constants of sort natural numbers for cardinalities of atomic sets. After these steps, a prover for Presburger arithmetic can be used to check the satisfiability of the derived formula. With this method a formula in the combined theory BAPA is reduced to a formula in Presburger arithmetic in a sound and complete way. This works nicely as the only attributes of the sets that needs to be considered is their size, i.e specific elements are not of interest. We extended this method to deal with the additional bridging functions.

4 Combined Reasoning with Sets and Aggregation Functions

In contrast to the method of Kuncak et al. [2, 1], we are considering not only the size of the sets but their elements as well. This makes the method applicable for checking the satisfiability of a formula in the combined theories of sets of numbers and the aggregation functions sum, avg, min, max more complex.

In our approach, the bridging functions min and max have to be considered together, and the handling of avg relies on the function sum, i.e. we have three different theories, called BAPA_M (for min and max), BAPA_A (for avg) and BAPA_S (for sum), or combinations thereof.

The method involves the following parts, which are explained afterwards: a) enrichment of the given formula with axioms, b) transformation of the formula to pure linear arithmetic, c) computation of a model of this formula and construction of a new formula in pure linear arithmetic from the model, and finally d) computing a model of this formula. If we have both models, we can construct a model of the original formula.

In the enrichment step, a certain set of axioms for each of the theories is added to the given formula, to model some properties of the involved theories, resulting in the so called *enriched problem*. In the transformation step, we use an extended version of the transformation of Kuncak et al. with additional steps for treating each of the aggregation functions. The result of this step is called *transformed enriched problem*. We then need a prover for linear arithmetic for checking the satisfiability of the transformed enriched problem. In Kuncak et al.'s approach, this is a prover for Presburger arithmetic, as the value for the cardinality of a set is a natural number. As we are considering not only the size of the sets, but their elements as well, and as those can be elements of $\mathbb{N}, \mathbb{Z}, \mathbb{Q}, \mathbb{R}$, we need a prover that can deal with these domains and combinations thereof. The choice of the universe affects the completeness of the method. This is discussed in a later paragraph.

With a model of the enriched formula, we have values for the size of each of the involved sets and an assignment for the different instances of the involved aggregation functions, but no assignment of the elements of the sets.

To be able to generate assignments for the elements of the sets with the help of a prover for linear arithmetic, we use the values from the constructed model to build a new formula, which we call *set constraint*. A template on how to use the values from the models is used to incorporate the properties of sets and the involved aggregation function. There is one distinct template for each of the theories $\mathsf{BAPA_S}$, $\mathsf{BAPA_A}$ and $\mathsf{BAPA_M}$.

A model of the set constraint defines assignments for the sets, i.e. defines which elements are in which of the sets. Together with the assignments gathered from the model of the transformed enriched problem, we can construct a model for the original formula.

$\mathsf{BAPA_S}$. The method for dealing with the function sum is proven to be sound, i.e. a model of the enriched transformed model and a model of the set constraint can always be used to construct a model of the original problem. Completeness is proven for the case that the model of the transformed enriched problem has the property that for all individuals of the domain c there exist infinitely many individuals a and b in the domain such that $a \neq b \wedge a + b = c$.

If this property is not fulfilled, then the method is not able to generate a model of the set constraint an therefore no model of the original formula can be constructed.

$\mathsf{BAPA_A}$. As avg relies on sum the same considerations concerning completeness and soundness apply.

$\mathsf{BAPA_M}$. Informal considerations lead to the result that the method for dealing with min and max is sound. If a dense ordering on the domain of the model of the transformed enriched problem exists, the method is complete. If the considered model does not have this property, the effect is equivalent to the one for the appropriate case for sum.

Termination of the method is obvious.

Combination of the three theories $\mathsf{BAPA_S}$, $\mathsf{BAPA_A}$ and $\mathsf{BAPA_M}$ are possible as well. For generating the transformed enriched problem, a combination of the transformation methods is used, and the set constraint is a conjunction of the involved set constraints, i.e. the template for constructing the set constraint in the combined case is a union of the templates used for the involved theories.

5 Conclusion and Ongoing Work

We have developed a way for checking satisfiability of formulae in $\mathsf{BAPA_S}$, $\mathsf{BAPA_A}$, $\mathsf{BAPA_M}$, or combinations thereof. This method simplifies certain verification task and only relies on provers for arithmetic, which are well established and reliable tools.

As a next step we will formalize the proofs of soundness, completeness and termination for the methods for reasoning in $\mathsf{BAPA_M}$.

Based on the snippet presented in the appendix of [2], we are implementing a system that allows to use the presented methods and a prover for linear arithmetic for checking satisfiability of formulae in the presented theories.

To increase the flexibility of the developed approach, we will consider an extension that works in the following way: Instead of applying the aggregation functions on the elements of the sets, we change the method in such a way that a function f can be supplied so that the aggregation functions do not consider an element e but $f(e)$. This allows to reason about properties of elements of a set.

With this extension, we will have a verification tool with a variety of use cases in verification with data structures.

Acknowledgements

We would like to thank Viorica Sofronie-Stokkermans and Matthias Horbach for fruitful discussions.

References

[1] Viktor Kuncak, Huu Hai Nguyen, and Martin C. Rinard. An algorithm for deciding BAPA: Boolean algebra with presburger arithmetic. In Robert Nieuwenhuis, editor, *CADE-20, Proceedings*, volume 3632 of *LNCS*, pages 260–277. Springer, 2005.

[2] Viktor Kuncak and Martin Rinard. The first-order theory of sets with cardinality constraints is decidable. Technical Report 958, MIT CSAIL, July 2004.

[3] Viktor Kuncak and Martin C. Rinard. Towards efficient satisfiability checking for boolean algebra with presburger arithmetic. In Frank Pfenning, editor, *CADE-21, Proceedings*, volume 4603 of *LNCS*, pages 215–230. Springer, 2007.

Reasoning about Auctions*

Marco Caminati[1] Manfred Kerber[1] Christoph Lange[1,2] Colin Rowat[3]

[1] Computer Science, University of Birmingham, UK
[2] Fraunhofer IAIS and University of Bonn, Germany
[3] Economics, University of Birmingham, UK

Project homepage: `http://cs.bham.ac.uk/research/projects/formare/`

Abstract: In the ForMaRE project formal mathematical reasoning is applied to economics. After an initial exploratory phase, it focused on auction theory and has produced, as its first results, formalized theorems and certified executable code.

1 Introduction

An auction mechanism is mathematically represented through a pair of functions (a, p): the first describes how some given goods at stake are allocated among the bidders (also called participants or agents), while the second specifies how much each bidder pays following this allocation. Each possible output of this pair of functions is referred to as an *outcome* of the auction. Both functions take the same argument, which is another function, commonly called a *bid vector* $\boldsymbol{b}$; it describes how much each bidder values the possible outcomes of the auction. This valuation is usually expressed through money.

In this setting, some common questions are the study of the quantitative and qualitative properties of a given auction mechanism (e.g., whether it maximizes some relevant quantity, such as revenue, or whether it is efficient, that is, whether it allocates the item to the bidder who values it most), and the study of the algorithms running it (in particular, their correctness).

In the following three sections we will see three important cases of auctions for which we have proved theorems and extracted verified code using the Isabelle/HOL proof assistant.

2 Single-good static auctions

In the simplest case there is exactly one indivisible good auctioned in a single round of bidding. As a consequence, the formal details are simple: the allocation function a takes only two possible values, 1 and 0, depending on whether a participant receives the good or not, respectively. Its argument, the bid vector, is a map from the set of participants to the non-negative reals (or naturals, according to the kind of numbers chosen to represent money). The payoff for a participant is given by a simple algebraic expression (where v is the actual valuation of the good according to that participant): $v * a(\boldsymbol{b}) - p(\boldsymbol{b})$. An important auction is Vickrey's auction in which the good is allocated to the highest bidder, who pays the second-highest price. In this situation Vickrey's theorem holds, one of the most important theorems in auction theory: no participant can be better off upon bidding anything different from her actual valuation of the good; this holds independently of how the other participants behave. We formalized Vickrey's theorem in several proof assistants to get an idea of their suitability for auctions [2].

The subsequent result of ForMaRE was to extend this theorem, using Isabelle/HOL, in two ways: proving the inverse of this result, and characterizing the class of all the mechanisms (called generalized Vickrey) enjoying this truthful bidding properties [3, Theorem 2].

Finally, a further important result was formalized in Isabelle: that it is impossible to achieve budget balancing for the generalized Vickrey mechanism, [3, Theorem 3]. This means that there will always be some acceptable bid vector giving an outcome for which the sum of the total payments will be non-zero. Such a result is another standard of auction theory; however, our proof is original and has the distinctive feature of not relying on the specific form of the generalized Vickrey mechanism, thereby establishing an algebraic property of a wide class of mechanisms.

3 Multiple-good static auctions

An important generalization of the single-good case (§2) is that of a set of several goods at stake, but with the important proviso that participants do not bid on each item independently, but rather bid on sub*sets* of items. That is, they bid on combinations of items, e.g., allowing them to express interest for multiple items which are worth to them only when they are together (a left shoe *and* its right shoe); or allowing participants to express the same preference between distinct subsets of goods of which they need only one.

In this setting, there is a mechanism enjoying properties similar to those enjoyed by the Vickrey mechanism (§2). It is called nVCG (n goods following the mechanism of Vickrey, Clarke, and Groves). Determining the outcome (a, p) of such a second price, combinatorial auction is much more complex (indeed, NP-complete) [1, Chapter 12], since for each possible allocation, the total revenue has to be computed in order to find the maximum. This gives the value of a; then, a similar computation has to be performed to determine p.

In ForMaRE we have extracted executable Scala code for this from the Isabelle formalization. This allows to provide certified soundness properties for the code. In particular, the prices are non-negative, the outcome of the auction al-

*This work has been supported by EPSRC grant EP/J007498/1.

locates each good exactly once, and for each possible bid vector there is exactly one outcome (up to tie-breaking).

Producing the formalization of theorems and automatically generated Scala code has been the fundamental accomplishment of the project. One of the main goals is now to better integrate these two efforts. Indeed, at the moment, some formalized theorems only apply to the simplest among the settings for which we can generate Scala code. Notably, while we can extract code to run any combinatorial VCG auction, the proofs for the Vickrey characterization and budget imbalance theorems are currently restricted to the special case of single-good auctions.

Generalizing those results from the single-good case to the combinatorial case is not trivial, because the allocation function gets to yield values more complex than the numbers 0,1. Correspondingly, the definition of payoff has to be modified into $v\left(a\left(\boldsymbol{b}\right)\right) - p\left(\boldsymbol{b}\right)$, and the simple second price rule has to be reformulated.

4 Dynamic auctions

A possible goal in designing a mechanism is to allow participants to refine their valuations about the goods at stake and in general the information they have about the possible outcomes. One common way to achieve this is to run dynamic auctions: the goods are not allocated in a single round, but multiple bidding rounds are run until some condition is met (e.g., nobody raises her bid any longer). This inevitably increases the risk of setting up an ill-specified auction (e.g., by specifying one that can never stop). Hence, formal methods get even more useful.

There are some problems for the adoption of formal methods in this setting:

1. A dynamic auction inherently requires repeating an input/output phase from participants, which typically happens through hardware and software the designer has no control on (e.g., on a remote machine).

2. The functional Isabelle/HOL formalization we used to generate Scala code has no notion of time. Hence, to actually run an auction, we wrap the Isabelle-generated code in a manually written Scala snippet providing access to the hardware clock of the machine.

For the last issue, the idea is to make the manually written wrapper as thin as possible. A `while` loop is all one needs to get access to the clock: both the termination condition and the code executed in each round can and should then be generated by Isabelle. Besides the skeleton of the `while` loop, the remaining manually written code should only concern input/output (point 1. above): we tested such a minimal wrapper loop around the Isabelle-generated code. It is eight lines of code, in the file `Dyna.scala` available in the GitHub repository linked from our homepage. After this loop, the last bid vector is passed to a second stage determining the outcome according to a and p, for which we can use the existing code of static auctions (see § 3).

5 Conclusion

The ForMaRE project has been working on three themes: theorems for single-good static auctions, verified code for multiple-good combinatorial static auctions, and verified code for dynamic auctions. These themes all present potential for further work, suggesting three dimensions along which to expand the project, building on the formalization already created, and using it as a guidance. As we mentioned in section 3, combining the first two themes, we are studying the extensions of theorems from section 2 to the combinatorial auctions of section 3. But this will only be a first step, because there are much more complex auctions of theoretical and practical relevance: there are interesting properties of a mechanism which depend on the information and beliefs of the single participant, and not merely on the specification of the mechanism itself. For example, a possible desirable goal in designing an auction would be that each participant submits a bid close to her perception of the value of the good, rather than a strategic lie to influence the outcome; or even to allow each participant to form or refine such a perception. To do that, the theory must be enriched to more expressively model the participants themselves, rather than only the mechanism. This has been accomplished in a subfield of game theory, mechanism design (also known as reverse game theory), by introducing the so-called type profile t^i of a given participant i. It models the participant's information, beliefs and preferences, so that in general the *payoff* u^i of the participant is a function of both the outcome *and* the profile. Our current machinery lacks the notion of profile, and introducing it will move our whole approach beyond the generalization to combinatorial auctions, granting us the possibility to formalize and investigate the vast range of results of reverse game theory.

The theme of dynamic auctions is of practical relevance and presents challenging possible evolutions: for example, how to regulate and implement the possibility for any participant (including the current winner) to withdraw her bid, and how to handle the feedback given to each participant after each bidding round. This latter point will need more complex interfacing between the dynamic stage and the existing winner determination code.

References

[1] Peter Cramton, Yoav Shoham, and Richard Steinberg, editors. *Combinatorial auctions*. MIT Press, 2006.

[2] Christoph Lange et al. A qualitative comparison of the suitability of four theorem provers for basic auction theory. In *Intelligent Computer Mathematics: MKM, Calculemus, DML, and Systems and Projects 2013*, pages 200–215. Springer-Verlag, 2013.

[3] Eric Maskin. The unity of auction theory: Milgrom's master class. *Journal of Economic Literature*, 42(4):1102–1115, December 2004.

Automating Regression Verification

Dennis Felsing Sarah Grebing Vladimir Klebanov Mattias Ulbrich

Karlsruhe Institute of Technology, Germany

Abstract: Regression verification is an approach to prevent regressions in software development using formal verification. The goal is to prove that two versions of a program behave equally or differ in a specified way. We worked on an approach for regression verification, extending Strichman and Godlin's work by relational equivalence and two ways of using counterexamples.

1 Introduction

Preventing unwanted behaviour, commonly known as *regressions*, is a major concern during software development. Currently the main quality assurance measure during development is *regression testing*. Regression testing uses a manually crafted test suite to check the behaviour of new versions of a program.

For example, consider the following two functions in ANSI C in Figure 1, which both calculate the greatest common divisor of two positive numbers:

```
int gcd1(int a, int b) {
  if (b == 0) {
    return a;
  } else {
    a = a % b;
    return gcd1(b,a);
  }
}
```

```
int gcd2(int x, int y) {
  int z = x;
  if (y > 0) {
    z = gcd2(y, z % y);
  }
  return z;
}
```

Figure 1: Example functions calculating the GCD

To test such a function multiple test cases would have to be written to cover the entire function behaviour. Writing these regression tests requires such an amount of manual work that typically more than 50% of the development time is spent on designing test cases [5]. Still, there is no guarantee of finding all introduced bugs.

Another approach to this problem is *formal verification*: The functions *gcd1* and *gcd2* can individually be proved correct with respect to a formal specification of the greatest common divisor, which would imply their equivalence. This requires the software engineer to provide said formal specification. Additionally it is often necessary to manually guide the proof.

Regression verification offers the best of both worlds. As in formal verification full coverage is achieved and no test cases are required. As in regression testing no formal specification of function behaviour is required. Instead of comparing the two programs to a common formal specification, regression verification compares them to each other. The old program version serves as specification of the correct behaviour of the new one. Note that the "correctness" that regression verification proves is different from that shown using formal verification. In formal verification there is a degree of freedom for the program behaviour, which allows to introduce certain bugs even when a bug is present.

In regression verification the use of an old program version as specification assures that the exact behaviour is preserved, so no new bugs can be introduced at all.

Regression verification is limited to proving functional relations, such as equivalence, between program versions. Regression testing on the other hand can also be employed to test for nonfunctional requirements, such as performance.

A number of approaches for regression verification have been developed. [1, 2, 4, 6]

2 Overapproximation using uninterpreted functions

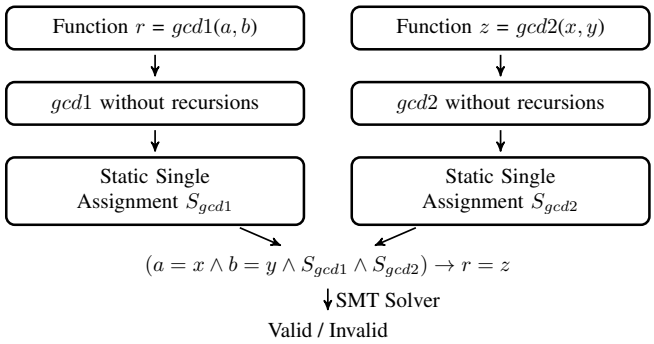

Figure 2: Regression verification approach by Strichman and Godlin

An initial approach for regression verification has been developed by Strichman and Godlin and is illustrated in Figure 2 [3]:

Proving equivalence of *gcd1* and *gcd2* is difficult since the programs call themselves recursively, potentially an unbounded number of times. Strichman and Godlin propose to replace recursive calls by the same placeholder in both programs, a so called *uninterpreted function* U.

Afterwards the programs can be converted into logical formulae S_{gcd_1}, S_{gcd_2}, which incorporate U. These formulae model the behaviour of *gcd1* and *gcd2* respectively, relating function outputs to inputs. Hence, S_{gcd_1} and S_{gcd_2} imply the equality of respective output values, assuming equality of inputs, in the following way:

$$(a = x \wedge b = y \wedge S_{gcd1} \wedge S_{gcd2}) \rightarrow r = z \qquad (1)$$

Strichman and Godlin use the bounded model checker CBMC to prove this formula for C programs on bitvectors.

We implemented this approach in the tool *simplRV* (**Sim**ple **P**rogramming **L**anguage **R**egression **V**erification), which is capable of performing regression verification on unbounded integer and array functions in a simple imperative programming language featuring recursions as well as loops, but no global variables. simplRV outputs an SMT formula, which is passed to state-of-the-art SMT solvers like Z3 and Eldarica. We developed and implemented the following extensions to the existing approach within *simplRV*:

Total equivalence between the functions to be compared is not always desired. Consider our example in Figure 1: Equality fails for negative numbers, but one can imagine that these functions are only called with positive numbers. In this case we require *conditional equivalence* for nonnegative inputs:

$$(a \geq 0 \wedge a = x \wedge b = y \wedge S_{gcd1} \wedge S_{gcd2}) \rightarrow r = z \quad (2)$$

Another common example for conditional equivalence are bug fixes in the program. Once a bug has been fixed, an equivalence proof is still desirable to prevent the introduction of new bugs. But simple equivalence of all outputs for all inputs would not be correct in this case. Instead the case of the bug fix has to be excluded using a precondition.

We implemented *relational equivalance* (denoted as "$\simeq$"), which is a superset of conditional equivalence, so that the user can specify relations between the inputs and outputs of functions. By default equality is used as the relation.

Our tool makes use of *counterexamples*, which are returned by the SMT solver on a failed proof. The functions are automatically tested using these counterexamples, and if their outputs differ, the programs are not equivalent and the user is informed about this with an actual counterexample.

Spurious counterexamples can be returned by the SMT solver because we overapproximate the functions using an uninterpreted function. We use the information won from spurious counterexamples as additional constraints on the uninterpreted function U and rerun the proof. This successfully handles the cases where a finite number of function values serve as the non-recursive base case of the function.

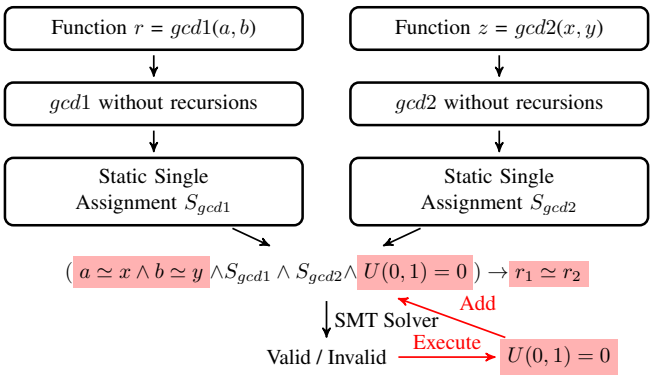

Figure 3: Extended regression verification approach

A summary of our extensions to the initial approach is given in Figure 3.

Using a collection of examples from various sources including compiler optimizations, refactorings and other publications we evaluated our approach and found it to work well for a wide range of examples.

Utilizing the information of spurious counterexamples can lead to an endless loop of new spurious counterexamples. These so called *edge cases* occur when only one of multiple parameters has a base case. Proving equivalence of functions of this kind is a limitation of the approach just described. A more intricate view on the problem can help, which is ongoing research.

3 Conclusion and future work

We have extended the reach of regression verification. This enables us to prove a greater class of functions and to still prove equivalence for the relevant cases when a bug has been fixed in the program.

So far only integer programs have been considered. Extending our approaches to other constructs like heaps and objects will improve comparability to other regression verification approaches and enable more realistic use cases.

References

[1] José Almeida, Manuel Barbosa, Jorge Sousa Pinto, and Bárbara Vieira. Verifying cryptographic software correctness with respect to reference implementations. In María Alpuente, Byron Cook, and Christophe Joubert, editors, *Formal Methods for Industrial Critical Systems*, volume 5825 of *LNCS*, pages 37–52. Springer Berlin / Heidelberg, 2009.

[2] Gilles Barthe, Juan Manuel Crespo, and César Kunz. Relational verification using product programs. In Michael Butler and Wolfram Schulte, editors, *FM 2011: Formal Methods - 17th International Symposium on Formal Methods, Limerick, Ireland, June 20-24, 2011. Proceedings*, volume 6664 of *Lecture Notes in Computer Science*, pages 200–214. Springer, 2011.

[3] Benny Godlin and Ofer Strichman. Regression verification. In *Design Automation Conference, 2009. DAC'09. 46th ACM/IEEE*, pages 466–471. IEEE, 2009.

[4] C. Hawblitzel, M. Kawaguchi, S. K. Lahiri, and H. Rebêlo. Mutual summaries: Unifying program comparison techniques. In *Proceedings, First International Workshop on Intermediate Verification Languages (BOOGIE)*, 2011.

[5] Glenford J. Myers and Corey Sandler. *The Art of Software Testing*. John Wiley & Sons, 2004.

[6] Sven Verdoolaege, Martin Palkovic, Maurice Bruynooghe, Gerda Janssens, and Francky Catthoor. Experience with widening based equivalence checking in realistic multimedia systems. *J. Electronic Testing*, 26(2):279–292, 2010.

Automated Reasoning in Deontic Logic*

Ulrich Furbach[1] Claudia Schon[1] Frieder Stolzenburg[2]

[1] Universität Koblenz-Landau, {uli,schon}@uni-koblenz.de
[2] Harz University of Applied Sciences, fstolzenburg@hs-harz.de

1 Introduction

Deontic logic is a very well researched branch of mathematical logic and philosophy. Various kinds of deontic logics are discussed for different applications like argumentation theory, legal reasoning and actions in multi-agent systems ([6]). Recently there also is growing interest in modelling human reasoning and testing the models with psychological findings. Deontic logic is an obvious tool to this end, because norms and licenses in human societies can be described easily. In [5] there is a discussion of some of these problems including a solution with the help of deontic logic. This paper concentrates on automated reasoning in deontic logic. We show that deontic logic can be translated into the description logic $\mathcal{ALC}$, for which the first oder reasoning system Hyper offers a decision procedure.

2 Deontic Logic as Modal Logic KD

In this section we briefly describe how standard deontic logic can be seen as modal logic K together with the seriality axiom D: $\Box\Phi \rightarrow \Diamond\Phi$. The $\Box$-operator is interpreted as 'it is obligatory that' and the $\Diamond$ as 'it is permitted that'. Assuming a Kripke-semantics, we have that there are different worlds, in which not all of the norms have to hold, but due to the seriality axiom, there always exists a world in which the norms hold, hence an ideal world.

To demonstrate the use of deontic logic, we consider the well-known problem of *contrary-to-duty obligations* introduced in [4] and the formalization as a normative system $\mathcal{N}'$ given in [10] (where s stands for *steals* and p for *punished*):

a') $\Box\neg s$

b') s

c') $s \rightarrow \Box p$

d') $\Box(\neg s \rightarrow \neg p)$

As shown in [10], the normative system a') - d') is inconsistent. This example is used as a running example throughout this paper.

3 Translating Deontic Logic into Description Logic

Hyper [11] is a theorem prover for first order logic with equality. It is the implementation of the E-hypertableau calculus [1] which extends the hypertableau calculus with equality handling based on superposition. Hyper has been successfully used in various AI-related applications, like intelligent interactive books or natural language query answering.

Recently the E-hypertableau calculus and its implementation have been extended to deal with knowledge bases given in the description logic $\mathcal{SHIQ}$ [2]. There is a strong connection between modal logic and description logic. As shown in [9], the description logic $\mathcal{ALC}$ is a notational variant of the modal logic $\mathcal{K}_n$. Therefore any formula given in the modal logic $\mathcal{K}_n$ can be translated as usual into an $\mathcal{ALC}$ concept and vice versa.

In addition to that, we have to translate the seriality axiom into description logic. In [7] it is shown, that the seriality axiom can be translated into the following TBox:

$$\mathcal{T} = \{\top \sqsubseteq \exists R.\top\}$$

with R the atomic role introduced by the translation described above.

4 Hypertableau for Deontic Logic

In this paper we used the deontic operator only in a few conditional formulae. In the philosophical literature deontic logic is also used to formulate entire normative systems (e.g. [10]). In practice such normative systems can be rather complex. This makes it difficult for the creator of a normative system to see if a normative system is consistent. We will show that it is helpful to be able to check consistency of normative systems automatically and use the Hyper theorem prover to check the consistency of a given normative system very much the same way as we used it in the previous sections.

The translation of the normative system $\mathcal{N}'$ introduced above into $\mathcal{ALC}$, called $\phi(\mathcal{N}')$ henceforth, is shown in Table 1. Checking the consistency of the normative system $\mathcal{N}'$ corresponds to checking the consistency of $\phi(\mathcal{N}')$ w.r.t. the TBox $\mathcal{T} = \{\top \sqsubseteq \exists R.\top\}$, where $\phi(\mathcal{N}')$ is the conjunction of the concepts given in the right column of Table 1. We transform $\phi(\mathcal{N}')$ into DL-clauses, which is the input language of Hyper. We will not give the result of this transformation here, but refer to [8] for details. Hyper constructs a hypertableau for the resulting set of DL-clauses. This hypertableau is closed and therefore we can conclude, that the set of DL-clauses is unsatisfiable. This tells us, that the normative system $\mathcal{N}'$ formalized above is inconsistent.

*Work supported by DFG grants FU 263/15-1 and STO 421/5-1 'Ratiolog'

Deontic Logic	$\mathcal{ALC}$
$\Box\Phi \rightarrow \Diamond\Phi$	$\top \sqsubseteq \exists R.\top$
$\Box\neg s$	$\forall R.\neg S$
s	S
$s \rightarrow \Box p$	$\neg S \sqcup \forall R.P$
$\Box(\neg s \rightarrow \neg p)$	$\forall R.(S \sqcup \neg P)$

Table 1: Translation of the normative system $\mathcal{N}'$ into $\mathcal{ALC}$.

5 An Example from Multi-agent Systems

In multi-agent systems there is a relatively new area of research, namely the formalization of 'robot ethics'. It aims at defining formal rules for the behavior of agents and to prove certain properties. As an example consider Asimov's laws, which aim at regulating the relation between robots and humans. In [3] the authors depict a small example of two surgery robots obeying ethical codes concerning their work. These codes are expressed by means of deontic logic, very much the same as we defined the normative systems in this paper.

We will give a formalization in standard deontic logic of this example together with a description of the proof tasks for Hyper. In our example, the robots $ag1$ and $ag2$ have two possible actions: $ag1$ can terminate a person's life support and $ag1$ can delay the delivery of pain medication. We consider two ethical codes O and $O^\star$

- $O \rightarrow \Box\neg action(ag2, delay)$, which means that "If ethical code O holds, then robot $ag2$ takes care, that delivery of pain medication is not delayed."
- $O^\star \rightarrow O \wedge O^\star \rightarrow \Box\neg action(ag1, term)$, which means that "If ethical code $O^\star$ holds, then code O holds, and robot $ag1$ takes care, that life support is not terminated."

Further we give a slightly modified version of the evaluation of the robot's actions given in [3], where $(+!!)$ describes the most and $(-!!)$ the least desired outcome:

$$action(ag1, term) \wedge action(ag2, delay) \rightarrow (-!!)$$
$$action(ag1, term) \wedge \neg action(ag2, delay) \rightarrow (-!)$$
$$\neg action(ag1, term) \wedge action(ag2, delay) \rightarrow (-)$$
$$\neg action(ag1, term) \wedge \neg action(ag2, delay) \rightarrow (+!!)$$

Further we add formulae stating that the formulae for the evaluation of the robot's actions hold in all reachable worlds. A possible query would be to ask, if the most desirable outcome $(+!!)$ will come to pass, if ethical code $O^\star$ is operative. This query can be translated into a satisfiability test: If $O^\star \wedge \Diamond\neg(+!!)$ is unsatisfiable, then ethical code $O^\star$ ensures outcome $(+!!)$. We have been able to solve this task successfully by translating the above formulae and the query into DL-clauses and use Hyper to test the satisfiability of the resulting set of DL-clauses.

References

[1] Peter Baumgartner, Ulrich Furbach, and Björn Pelzer. Hyper tableaux with equality. In Frank Pfennig, editor, *Automated Deduction - CADE 21, 21st International Conference on Automated Deduction, Bremen, Germany, July 17-20, 2007, Proceedings*, volume 4603 of *Lecture Notes in Computer Science*, 2007.

[2] Markus Bender, Björn Pelzer, and Claudia Schon. System description: E-KRHyper 1.4 - extensions for unique names and description logic. In Maria Paola Bonacina, editor, *CADE-24*, LNCS, 2013.

[3] Selmer Bringsjord, Konstantine Arkoudas, and Paul Bello. Toward a general logicist methodology for engineering ethically correct robots. *IEEE Intelligent Systems*, 21(4):38–44, 2006.

[4] R. M. Chisolm. Contrary-to-duty imperatives and deontic logic. *Analysis*, 23:33–36, 1963.

[5] Ulrich Furbach and Claudia Schon. Deontic logic for human reasoning. *CoRR*, abs/1404.6974, 2014.

[6] John F. Horty. *Agency and Deontic Logic*. Oxford University Press, Oxford, 2001.

[7] Szymon Klarman and Víctor Gutiérrez-Basulto. Description logics of context. *Journal of Logic and Computation*, 2013.

[8] Boris Motik, Rob Shearer, and Ian Horrocks. Optimized Reasoning in Description Logics using Hypertableaux. In Frank Pfenning, editor, *CADE-21*, volume 4603 of *LNAI*. Springer, 2007.

[9] Klaus Schild. A correspondence theory for terminological logics: Preliminary report. In *In Proc. of IJCAI-91*, pages 466–471, 1991.

[10] Frank von Kutschera. *Einführung in die Logik der Normen, Werte und Entscheidungen*. Alber, 1973.

[11] Christoph Wernhard and Björn Pelzer. System description: E-KRHyper. In Frank Pfennig, editor, *Automated Deduction - CADE 21, 21st International Conference on Automated Deduction, Bremen, Germany, July 17-20, 2007, Proceedings*, volume 4603 of *Lecture Notes in Computer Science*, 2007.

Modular Verification of Interconnected Families of Uniform Linear Hybrid Automata

Matthias Horbach Viorica Sofronie-Stokkermans

University Koblenz-Landau, Koblenz, Germany
and Max-Planck-Institut für Informatik, Saarbrücken, Germany

Abstract: We provide a mathematical model for unbounded parallel compositions of structurally similar linear hybrid automata, whose topology and flows are described using parametrized formulas. We study how the analysis of safety properties for the overall system can be performed hierarchically, using quantifier elimination to derive conditions on the parameters of individual components.

1 Introduction

We study possibilities of using hierarchical reasoning, quantifier elimination and model generation for the analysis and verification of families of structurally similar parametric hybrid systems. We illustrate our method using a system of water tanks connected in a simple linear topology.

2 Linear Hybrid Automata

A *hybrid automaton* (*HA* for short) is a tuple $S = (X, Q, \mathsf{flow}, \mathsf{inv}, \mathsf{init}, E, \mathsf{guard}, \mathsf{jump})$. X is a set of variables, Q a set of control modes and E a finite multiset of transitions between modes. For each $q \in Q$, (i) a predicate flow_q over the variables in X and their first derivatives specifies the continuous dynamics, (ii) a predicate inv_q over X defines invariant conditions, and (iii) a predicate init_q over X definines the initial states for control mode q. For each $e \in E$, (i) a predicate guard_e over X resticts states in which the transition can be taken, and (ii) a predicate jump_e over $X \cup X'$ (a copy of X whose elements are "primed") states how variables are reset during the transition.

A hybrid automaton S is *linear* if it satisfies the following two requirements [3]:

1. Linearity: For every control mode $q \in Q$, the flow condition, the invariant condition, and the initial condition are convex linear predicates, i.e. finite conjunctions of strict or non-strict linear inequalities For every control switch, the guard and reset conditions are convex linear predicates. In addition, as in [1, 2], we assume that the flow conditions are conjunctions of *non-strict* linear inequalities.

2. Flow independence: The flow condition of every mode is a predicate over the variables in $\dot{X}$ only (and does not contain any variables from X). This requirement ensures that the possible flows are independent from the values of the variables, and only depend on the control mode.

Parametric Linear Hybrid Automata. We consider *parametric* linear hybrid automata (PLHA) (cf. also [1, 2]), defined as linear hybrid automata for which a set $\Sigma_P = P_c \cup P_f$ of parameters is specified (consisting of parametric constants P_c and parametric functions P_f) with the difference that for every control mode $q \in Q$ and every mode switch e:

(1) the linear constraints in the invariant conditions Inv_q, initial conditions Init_q, and guard conditions guard_e are of the form: $g \leq \sum_{i=1}^{n} a_i x_i \leq f$,

(2) the inequalities in the flow conditions flow_q are of the form: $\sum_{i=1}^{n} b_i \dot{x}_i \leq b$,

(3) the linear constraints in jump_e are of the form $\sum_{i=1}^{n} b_i x_i + c_i x_i' \leq d$,

where the coefficients a_i, b_i, c_i and the bounds b, d are either numerical constants or parametric constants in P_c; and g and f are (i) constants or parametric constants in P_c, or (ii) parameteric functions in P_f satisfying the convexity (for g) resp. concavity condition (for f), or concrete functions with these convexity/concavity properties such that $\forall t(g(t) \leq f(t))$. The flow independence conditions hold as in the case of linear hybrid automata.

Uniform Parametric Linear Hybrid Automata. The fact that we consider PLHAs allows us to considerably simplify the description of such hybrid automata: We regard then as *uniform parametric hybrid automata* (UPLHA), i.e. systems in which modes have a uniform, but parametric, description. The differences between various modes are expressed not by different shapes of the predicates, but only by different properties of the parameters. For UPLHAs we have the following types of state change:

(Jump) The change of the control location from mode q to mode q' is simply due to an update of the values of the parameters which control the flow. The values of data variables are updated according to the jump conditions.

(Flow) For fixed values of the parameters the state can change due to the evolution in a given control mode over an interval of time: the values of data variables change in a continuous manner according to the flow rules of the current control location for the given values of the parameters.

We have proven that every parametric LHA can be represented as a uniform parametric LHA.

Example 1 *Consider a controller that tries to keep the water level in a tank within a safe region, i.e. below a critical level* $\mathsf{l}_{\mathsf{overflow}}$*, by opening and closing an outlet valve (Figure 1, left). It is described by an LHA with two modes: For the mode in which the valve is closed, the flow is*

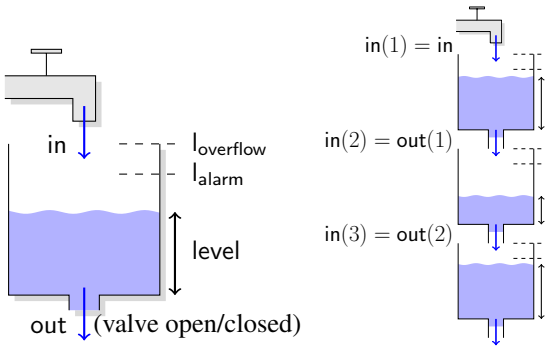

Figure 1: A single water tank and a system of tanks

$\dot{\mathsf{level}} = \mathsf{in}$, *for the mode in which the valve is open we have* $\dot{\mathsf{level}} = \mathsf{in} - \mathsf{out}$. *The difference between the two modes is expressed only in the fact that different constants are used in the differential equations which describe the flows in the two modes: the coefficient of* out *is either 1 or 0.*

The difference between the mode in which the valve is open and the mode in which the valve is closed is in the value of the constant in the flow description: $\dot{\mathsf{level}} = c$. *For the mode in which the valve is closed,* $c = \mathsf{in}$, *for the mode in which the valve is open,* $c = \mathsf{in} - \mathsf{out}$. *A jump from one mode to the other switches the value of* c *from* in *to* $\mathsf{in} - \mathsf{out}$ *and vice-versa.*

3 Systems of Uniform Hybrid Automata

We consider systems of hybrid automata of the form $\{S_i \mid i \in I\}$, where I is an index set (with an underlying structure modeling neighborhood and/or communication) and each S_i is a (uniform) linear hybrid automaton. We assume that all systems S_i have a similar description, using indexed variants of the same continuous variables (cf. [4]). We consider "global" safety properties of the form $\forall i \Psi(i)$. An example of a verification task is to show that the formula is invariant under jumps and flows. We showed that, in this setting, many verification tasks can be decomposed modularly to the verification of a bounded number of components. Moreover, we can synthesize requirements for the parameters that guarantee safety. Because of space liminations, we illustrate the ideas on an example.

Example 2 ([4]) *Consider a family of* n *water tanks with a uniform description (e.g. as in Figure 1, right), each modeled by the hybrid automaton* S_i. *Assume that every* S_i *has one continuous variable* level_i *(representing the water level in* S_i*), and that the input and output in mode* q *are described by parameters* in_i *and* out_i. *Every* S_i *has only one mode, in which the water level evolves according to rule* $\dot{\mathsf{level}}_i = \mathsf{in}_i - \mathsf{out}_i$. *We write* $\mathsf{level}(i,t), \mathsf{in}(i)$, *and* $\mathsf{out}(i)$ *instead of* $\mathsf{level}_i(t), \mathsf{in}_i$, *and* out_i, *respectively.*
Assume that the water tanks are interconnected in such a way that the input of system S_{i+1} *is the output of system* S_i. *A global constraint describing the communication of the systems is therefore:*

$$\forall i(2 \leq i \leq n-1 \rightarrow (\mathsf{in}(i) = \mathsf{out}(i-1)) \wedge \mathsf{in}(1) = \mathsf{in}.$$

An example of a "global" update describing the evolution of the systems S_i *during a flow in interval* $[t_0, t_1]$*:*

$$\forall i(\mathsf{level}(i,t_1) = \mathsf{level}(i,t_0) + (\mathsf{in}(i) - \mathsf{out}(i))(t_1 - t_0)).$$

Let $\Psi(t) = \forall i(\mathsf{level}(i,t) \leq \mathsf{l}_{\mathsf{overflow}})$ *be the safety property of our system that we are interested in. Assume that* $\forall i(\mathsf{in}(i) \geq 0 \wedge \mathsf{out}(i) \geq 0)$. *We generate a formula which guarantees that* Ψ *is an invariant: We start with the following formula (for simplicity of presentation we already replaced* $\mathsf{in}(i)$ *with* $\mathsf{out}(i-1)$*):*

$$\begin{aligned}\exists t_0, t_1 \quad & t_0 < t_1 \wedge \forall i(\mathsf{level}(i,t_0) \leq \mathsf{l}_{\mathsf{overflow}}) \\ & \wedge \exists j(\mathsf{level}(j,t_1) > \mathsf{l}_{\mathsf{overflow}}) \\ & \wedge \forall i((i{=}1 \wedge \mathsf{level}(1,t_1){=}\mathsf{level}(1,t_0) + (\mathsf{in} - \mathsf{out}(1))(t_1 - t_0)) \\ & \quad \vee (i > 1 \wedge \mathsf{level}(i,t_1) = \mathsf{level}(i,t_0) \\ & \qquad + (\mathsf{out}(i{-}1) - \mathsf{out}(1))(t_1 - t_0))).\end{aligned}$$

After Skolemization, quantifier elimination and some simplification, we obtain

$$\begin{aligned}\forall i(\quad & (i = 1 \rightarrow (\mathsf{in} - \mathsf{out}(i_0)){\leq}0) \\ & \wedge (i > 1 \rightarrow (\mathsf{out}(i{-}1) - \mathsf{out}(i)){\leq}0)).\end{aligned}$$

This means that Ψ *is an invariant for the family of systems if, and only if, this condition is satisfied.*

4 Conclusion

We presented a way of representing systems of similar hybrid automata which allows us to reduce the problem of checking invariance of safety conditions under jumps and flows to checking satisfiability of ground formulae w.r.t. background theories. We illustrated the types of problems which can be solved for a system $\{S_i \mid i \in I\}$ of parametric linear hybrid automata, where I is a list (here modeled by the set of natural numbers). Our method can also be applied to more complex topologies (e.g. systems of water tanks where I has a tree structure). Similar results can be obtained if updates are caused by changes in the topology (insertion or deletion of water tanks in the system in our example). In the future, we want to make out approach fully automatic and analyze its complexity.

Acknowledgements: This work was partly supported by the German Research Council (DFG) as part of the Transregional Collaborative Research Center "Automatic Verification and Analysis of Complex Systems" (SFB/TR 14 AVACS). See `www.avacs.org` for more information.

References

[1] W. Damm, C. Ihlemann, V. Sofronie-Stokkermans. Decidability and complexity for the verification of reasonable linear hybrid automata. *Proc. HSCC 2011*, 73–82, ACM, 2011.

[2] W. Damm, C. Ihlemann, V. Sofronie-Stokkermans. PTIME Parametric Verification of Safety Properties for Reasonable Linear Hybrid Automata. *Mathematics in Computer Science* 5(4): 469–497, 2011.

[3] T.A. Henzinger, P.W. Kopke, A. Puri, P. Varaiya. What's decidable about hybrid automata? *Journal of Computer and System Sciences* 57(1): 94–124, 1998.

[4] V. Sofronie-Stokkermans. Hierarchical Reasoning and Model Generation for the Verification of Parametric Hybrid Systems. *Proc. CADE 2013*, 360–376, Springer, 2013.

(AI) Planning to Reconfigure your Robot?

Mark Judge

Department of Automatic Control and Systems Engineering, University of Sheffield, Sheffield
m.judge@sheffield.ac.uk

Abstract: Current and future robotics and autonomous system applications will be used in highly complex environments. In such situations, automatic reconfiguration, due either to changing requirements or equipment failure, is highly desirable. By using a model of autonomy based around the Robot Operating System (ROS), and building on previous work, it is possible to use Artificial Intelligence (AI) Planning to facilitate the automatic reconfiguration. In this way, standard AI Planning machinery may be used with a suitable mathematical model of a given system. This paper reviews the background to this concept and details the initial steps taken towards the combination of AI Planning technology and a physical system, with the aim being to have the planner provide possible validated system reconfigurations which can then be implemented on the hardware.

1 Introduction

Early robotics systems [7] were built around three fundamental primitives: Sense, Plan, Act (SPA). However, since it became clear that planning was too complex to be carried out on-board in real time, AI planning has become a research area in its own right.

As a consequence of the divergence of planning and robotics, AI planning has grown, matured, and now employs a wide range of techniques, many of which are applied to a broad spectrum of complex problem domains [2].

Hence, originally, planning was intended as a mechanism for controlling the operation of a robot, but may now often be considered a general solving method, applicable to problems with a known current (*initial*) state and a desired *goal* state.

It is in this general problem solving sense in which AI planning technology is used here in order to provide a solution "plan" for the reconfiguration "problem" of an autonomous (robot) system. To facilitate the description of the autonomous system(s) as a planning domain, the Robot Operating System (ROS) [8] was used, with a mathematical model developed to describe the ROS system, with this in turn allowing autonomous reconfiguration.

A more detailed discussion of autonomous systems control and the formal model used for reconfiguration is provided in our foundational paper [1]. Here, in this current paper, we summarise that previous work, focusing more on the planning aspect and emphasising the use of an AI Planning logic system for the reconfiguration process.

2 Background

In this section, we highlight the need for the reconfiguration of autonomous and robotic systems, introduce ROS, and briefly describe one form of AI planning.

Autonomous control and robotics systems operate in highly complex environments, much more complex than the environments in which traditional control systems operated. The reconfiguration of such systems is required in order to accommodate either changes in the environment or changes in a given system's hardware [5], perhaps due to a fault or a better subsystem component becoming available. Such reconfiguration demands considerable resources from system engineers. Hence, automatic reconfiguration is desirable not only to minimise resource use, but also to reduce or remove errors and speed up redevelopment and redeployment.

ROS is an open source robot operating system, originally developed for use on specific large-scale service robot and mobile manipulator platforms. The designers' stated aims in developing ROS were that their system would be peer-to-peer, free, open source, thin, tools based, and multi-lingual (C++, Python, LISP, Octave).

A typical ROS system comprises a number of processes, optionally distributed over multiple hardware systems, and consists of *nodes*, *messages*, *topics*, and *services*. Nodes (software modules) carry out the computation and communicate via messages, doing this by publishing to a particular topic. In addition to this "broadcast" type model, services are used for synchronous transactions, which are defined by a named string and pairs of messages of a strict type.

Classical planning, also known as STRIPS planning [3], requires that the *state space* be finite and fully observable. It is assumed that only specified actions can change a state and that they do so instantaneously, with the resulting state being predictable. A STRIPS planning problem, P = (O,I,G), where O is a set of operators, I is a conjunction of fact literals describing the initial state, and G another conjuction of facts describing a partially specified goal state.

AI planning problems can be described using the Planning Domain Definition Language (PDDL) [6]. In order to find solutions to such planning problems, a dedicated planning system must be constructed. This is generally a time consuming process. Hence, for this work, one of the most well known, and best performing, planning systems, called Fast Forward (FF) [4], was used to solve the problem instances once these were formulated.

3 AI Planning for System Reconfiguration

One way of mathematically describing [1] a ROS system is by making use of a tri-partite graph (Figure 1).

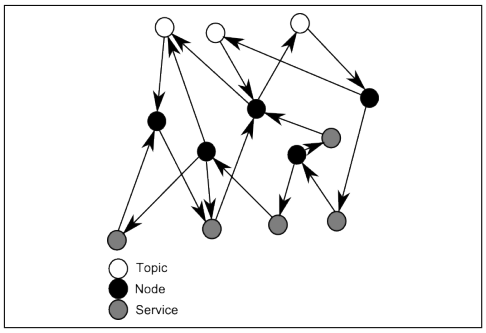

Figure 1: Example ROS graph.

In Figure 1, the three vertex types are considered different, with all inter-node communication occurring via a service or topic. Hence, the edges represent data flow, with a given topic or service requiring a minimum of one inbound edge from a ROS node.

To use AI planning machinery to reconfigure a ROS system of the type shown in Figure 1, we model the system using a PDDL domain description. This contains the action schema, which will be instantiated with the data contained in an associated problem instance description file.

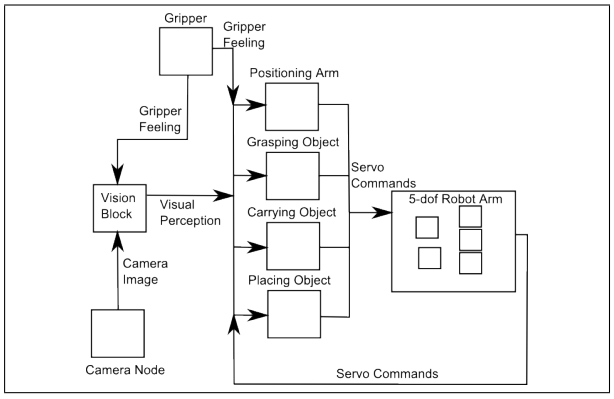

Figure 2: Example robot arm system diagram.

Taking as an example a modular robotic arm system combined with visual perception equipment, it is possible to set as a goal certain (lifting and moving) tasks, with the planning system providing the actual, physical (re)configuration detail to guarantee that a given task can be completed.

Considering Figure 2, the prototype planning system was set up to model two robot arms, each with 5 degrees of freedom (DOF), one with DC motor control, the other with servo control capability. The system is required to perform two different tasks. The first is Disk Loading, the second, Object Repositioning. The first task can only be performed with the servo control system in place, whilst the second can be carried out under either configuration.

In the planning action schema, each modelled ROS node supplies (or requires) certain services required (or supplied) by other nodes. This aligns well with the notion of preconditions and effects [3] in AI planning.

```
(:action dc_place_object
 :parameters
  (?dcservo1 - dcservo
   ?grippernode1 - grippernode
   ?visualperceptnode1 - visualperceptnode
   ?positionarmnode1 - positionarmnode
    ?dcplacingobjectnode1 - dcplacingobjectnode)
 :precondition
  (and (dcservo_available ?dcservo1 ?grippernode1)
       (gripper_sensing_available ?grippernode1)
      (visual_percept_available ?grippernode1
       ?visualperceptnode1 ?positionarmnode1)
      (no_dc_placing_object_available ?dcplacingobjectnode1)
      (visual_percept_not_assigned ?visualperceptnode1))
 :effect
  (and (dc_placing_object_available ?dcplacingobjectnode1)
       (not(visual_percept_not_assigned ?visualperceptnode1))
      (not(no dc placing object available?dcplacingobjectnode1))))
```

Figure 3: Example prototype action.

Part of the action description for the prototype is shown above in Figure 3. This shows the idea of one node (action) relying on the services provided by another, in the sense that the preconditions of the current action are satisfied by the effects of other, previous actions. Thus, a *plan* is a (re)configuration of the system.

Manual analysis of the ROS system used in the example gives two possible configurations for the Disk Loading task, and four for the Object Repositioning task. Running the planner on the PDDL encoded model gives the correct sets of solutions for both. A sample is shown in Figure 4.

```
0: DIGITAL_SERVO_BASE_TO_SHOULDER DIGITALBASENODE1 DIGITALSHOULDERNODE1 DISCLOAD1
1: DIGITAL_SERVO_SHOULDER_TO_ELBOW DIGITALBASENODE1 DIGITALSHOULDERNODE1 DIGITALELBOWNODE1
2: DIGITAL_SERVO_ELBOW_TO_WRIST DIGITALSHOULDERNODE1 DIGITALELBOWNODE1 DIGITALWRISTNODE1
3: DIGITAL_SERVO_WRIST_TO_GRIPPER DIGITALELBOWNODE1 DIGITALWRISTNODE1 DIGITALGRIPPERNODE1
4: SETUP_CAMERA_IMAGE CAMERANODE2
5: DIGITAL_SERVO_AVAILABLE DIGITALWRISTNODE1 DIGITALGRIPPERNODE1 DIGITALSERVO1
6: SETUP_GRIPPER_SENSING GRIPPERNODE2
7: VISUAL_PERCEPTION_READY VISUALPERCEPTNODE2 GRIPPERNODE2 CAMERANODE2 POSITIONARMNODE2
8: DIGITAL_DISC_LOADING DIGITALSERVO1 GRIPPERNODE2 VISUALPERCEPTNODE2 POSITIONARMNODE2 DIGITAL DISCLOADINGNODE1
```

Figure 4: Example Disk Loading configuration.

4 Conclusion

This paper summarises[1] a foundation for the development of autonomously reconfigurable robot systems. Results show, by defining a system in ROS and modelling this in PDDL, it is possible to attain automatic reconfiguration.

References

[1] J.M. Aitken, S.M. Veres, and M. Judge. Adaptation of System Configuration under ROS. In *Procs of 19th IFAC World Congress.*, 2014.

[2] A. Coles et al. A survey of the 7th ipc. *AI Magazine*, 33(1):1–8, 2012.

[3] R. E. Fikes and N. J. Nilsson. Strips. In *Readings in Planning*, pages 88–97. Kaufmann, San Mateo, CA, 1990.

[4] J. Hoffmann and B. Nebel. The FF planning system. *Journal of Artificial Intelligence Research*, 14:253–302, 2001.

[5] S. Karapinar, D Altan, and S. Talay. A robust planning framework for cognitive robots. In *Proceedings of AAAI Workshops (online)*, 2012.

[6] Craig Knoblock. Pddl. *AIPS98*, 78(4):1–27, 1998.

[7] N. J. Nilsson. *Principles of artificial intelligence*. Morgan Kaufmann, San Francisco, CA, USA, 1980.

[8] M. Quigley et al. Ros: an open-source robot operating system. In *ICRA Workshop on Open Source Software*, 2009.

[1] Thanks to J. Aitken and S. Veres, for access to their parts of [1] from which this paper draws material.

Using CSP Meta Variables in AI Planning

Mark Judge

Department of Computer and Information Sciences, University of Strathclyde, Glasgow
mark.judge@strath.ac.uk

Abstract: Reformulating Artificial Intelligence Planning problems as Constraint Satisfaction Problems (CSPs) brings a number of benefits. In this reformulation process however, the structure of the original planning problem is lost. Therefore, no planning problem specific guidance may be used during the assignment of values to the CSP variables. Extending work, in which we implemented a planning-specific CSP variable and value selection heuristic, the work described here aims to make better use of the propagation inherent in a given CSP solver by using CSP meta variables. Such meta variables are used here for goal-locking and for the assignment of resources to tasks within the problem solution process, with the aim being to reduce the search space, and to better guide the search within it.

1 Introduction

AI planning problems are generally described using the Planning Domain Definition Language (PDDL) [9]. Problems described in this way are then fed in to an application called a planner in order to find a solution. Planners are complex, often requiring much time and effort to build.

Instead of building a specialised application, it is possible to use the constraint programming paradigm, reformulating the original planning problem as a Constraint Satisfaction Problem (CSP) [2]. By doing this, it is possible to use generic CSP solution methods to solve the planning problem, and then convert the CSP solution back into a form that is a useful solution to the original problem.

Whilst generic CSP heuristics do aid the process of choosing values for each of the CSP variables (*labelling*), they fail to capitalise on the inherent structural information found in the planning problem. In previous work [8], the author presented a planning-specific CSP variable and value selection heuristic. This current work builds on that foundation by making use of CSP meta variables.

CSP meta variables are used both for goal-locking and for resource-to-task assignment. In the former case, additional inferences may be made as a result of the goal-locking meta variables being assigned values, which leads to reduced run time and less backtracking in the solution process for certain problems. In the latter case, preliminary work has shown that, by using meta variables to represent a given resource to task allocation, more efficient solutions are possible.

2 Background

The AI planning considered in this project is known as STRIPS planning [3], in which it is assumed that only specified actions can change a state and that they do so instantaneously, with the resulting state being predictable. The *state space* is required to be finite and fully observable.

A STRIPS planning problem, P = (O,I,G), where O is a set of operators, I is a conjunction of fact literals describing the initial state, and G another conjuction of facts describing a partially specified goal state.

In order to construct a CSP representation, the original planning problem (described in PDDL) was converted into a representation based on SAS$^+$ [1]. The SAS$^+$ approach may be considered more concise than STRIPS since a given planning problem can be encoded with many fewer variables, each with a larger range of possible values. This is helpful [2] when considering reformulation of a planning problem as a CSP.

Instead of labelling each of the CSP variables using a generic CSP variable and / or value selection heuristic, use is made of the problem's causal graph (CG) [6], which captures the causal dependencies between the SAS$^+$ variables. With this approach, the goal-state variables may be used in a planning specific heuristic algorithm [7] to better guide the variable and value selection process.

The result of using such a goal centric heuristic is a plan comprising a series of sub plans, each of which satisfies one of the goal-state goals. Figure 1 gives an example of this for a small logistics problem, showing the first sub plan.

```
3          5          2          . . .  Ac1:   1: boardtruckdriver1truck1s0
3          0          2          . . .  Ac2:   22: drivetrucktruck1s0s1driver1
3          0          2          . . .  Ac3:   44: loadtruckpackage2truck1s1
3          0          4          . . .  Ac4:   51: drivetrucktruck1s1s2driver1
3          0          4          . . .  Ac5:   68: unloadtruckpackage2truck1s2
3          0          0          . . .  Ac6:   _[[4,6,8,35,38,40,41,77,109]]
_[[2,3,7]],_[[0,3]],  _[[0,4]],  . . .  Ac7:   _[[2,4,6,8,9,12,15,16,19..22,31,...]]
_[0,4,7],  _[0,2..6],  _[0..2,4],  . . .  Ac8:   _[[1..10,12,15..53,55,56,64..77,109]]
_[0..7],   _[0..7],   _[0..4],   . . .  Ac9:   _[[1..79,106,109]]
_[0..7],   _[0..7],   _[0..4],   . . .  Ac10:  _[1..109]
_[0..7],   _[0..7],   _[0..4],   . . .  Ac11:  _[1..109]
_[0..7],   _[0..7],   _[0..4],   . . .  Ac12:  _[1..109]
_[0..7],   _[0..7],   _[0..4],   . . .  Ac13:  _[1..109]
_[0..7],   _[0..7],   _[0..4],   . . .  Ac14:  _[1..109]
_[0..7],   _[0..7],   _[0..4],   . . .  Ac15:  _[1..109]
_[0..7],   _[0..7],   _[0..4],   . . .  Ac16:  _[1..109]
_[0..7],   _[0..7],   _[0..4],   . . .  Ac17:  _[1..109]
_[0..7],   _[0..7],   _[0..4],   . . .  Ac18:  _[1..109]
_[0..7],   _[0..7],   _[0..4],   . . .  Ac19:  _[1..109]
_[0..7],   _[0..7],   _[0..4],   . . .  Ac20:  _[1..109]
_[0..7],   _[0..7],   _[0..4],   . . .  Ac21:  _[1..109]
_[0..7],   _[0..7],   _[0..4],   . . .  Ac22:  _[1..109]
_[0..7],   _[0..7],   _[0..4],   . . .  Ac23:  _[[1,2,4..10,13..34,36..44,46..64,...]]
_[0..7],   _[0..7],   _[[0,4]],  . . .  Ac24:  _[[5..8,13..21,23,26,27,29..34,...]]
_[0..7],   _[0..7],   _[[0,4]],  . . .  Ac25:  _[[5,6,13..20,26,27,29,30,32,34,...]]
_[[1..4,6]],_[1..5),  _[[0,4]],  . . .  Ac26:  _[[5,6,14,16,26,27,29,30,32,34,56,...]]
_[[2,4,6]],_[[1,2,4]],  _[[0,4]],  . . .  Ac27:  _[[14,16,56,59,68,84,107,109]]
4          4          0          . . .
```

Figure 1: Partial solution matrix with 1st subplan.

3 CSP Meta Variables for AI Planning

The solution of a CSP requires that a value is found for each of the variables, and that such values are taken from the respective domains of those variables. Additionally, the

set of values assigned must simultaneously satisfy all of the declared constraints. However, certain values for a given variable may be *interchangeable* [4] with no impact on the solution(s). A value, b, for a CSP variable, V, is *fully interchangeable* with a value, c, for V iff: **1**. Every solution to the CSP which contains b remains a solution when c is substituted for b, *and* **2**. Every solution to the CSP which contains c remains a solution when b is substituted for c. Where it is possible to eliminate interchangeable values in CSPs, a clustering of subsets of the original CSP variables may be achieved, with a set of *meta-variables* created.

A *meta*-CSP of a ground CSP, *X*, consists of variables that correspond to subsets of the variables in *X*. The values of the meta-variables are the solutions of the problems induced by the subsets of variables. The constraints between the meta-variables are satisfied when all of the constraints from the original CSP are satisfied. Applying this concept to a CSP encoded planning problem [5] allows a meta-CSP to be constructed, the variables of which represent the goals of the original planning problem. For a planning problem, P = (O,I,G), the meta-CSP is a CSP where $|X| = |G|$ and $D_i = \{\ a \mid g_i \in \text{add(a)}\}$. Each variable represents a goal, $g_i \in G$, with the associated domain containing only the actions that achieve g_i. Such actions have g_i in the add list of their effects. With a variable assigned a value in this representation, $\langle x_i, a \rangle$, the meaning is that a is the *final achiever* of g_i. This means that it is not possible for any action appearing later in the plan than a to delete g_i. This is shown in the partial solution matrix below (Figure 2) for the example in Section 2.

It is clear from Figure 2 that the domain (search space) of each the action variables, Ac8 to Ac23, has been reduced, giving the heuristic algorithm even less choice than before, leading to more efficient solving (Figures 3 and 4).

The second use of meta variables allows a meta variable to be assigned a value depending on which of the available *resources* are allocated to each of a problem's *tasks*. By labelling these meta variables, and using this to direct the search for actions to form a plan to satisfy the problem's goals (tasks), it is possible to show that such plans can be built faster, often with fewer backtracks, and sometimes with fewer actions. Although this part of the project is ongoing, results look promising.

```
3          5          2     . . .  Ac1:   1: boardtruckdriver1truck1s0
3          0          2     . . .  Ac2:   22: drivetrucktruck1s0s1driver1
3          0          2     . . .  Ac3:   44: loadtruckpackage2truck1s1
3          0          4     . . .  Ac4:   51: drivetrucktruck1s1s2driver1
3          0          4     . . .  Ac5:   68: unloadtruckpackage2truck1s2
3          0          0     . . .  Ac6:   _[[4,6,8,35,38,40,41,109]]
_[[2,3,7]],  _[[0,3]],    0     . . .  Ac7:   _[[2,4,6,8,9,12,15,16,19..22,31,...]]
_[0,4,7],    _[0,2..6],   0     . . .  Ac8:   _[[1,2,4..10,12,15..43,45..53,55,56,...]]
_[0..7],     _[0..7],     0     . . .  Ac9:   _[[1,2,4..10,12..43,45..66,70..75,78,79,...]]
_[0..7],     _[0..7],     0     . . .  Ac10:  [[1,2,4..10,12..43,45..66,70..75,78..80,...]]
_[0..7],     _[0..7],     0     . . .  Ac11:  [[1,2,4..10,12..43,45..66,70..75,78..80,...]]
_[0..7],     _[0..7],     0     . . .  Ac12:  [[1,2,4..10,12..43,45..66,70..75,78..80,...]]
_[0..7],     _[0..7],     0     . . .  Ac13:  [[1,2,4..10,12..43,45..66,70..75,78..80,...]]
_[0..7],     _[0..7],     0     . . .  Ac14:  [[1,2,4..10,12..43,45..66,70..75,78..80,...]]
_[0..7],     _[0..7],     0     . . .  Ac15:  [[1,2,4..10,12..43,45..66,70..75,78..80,...]]
_[0..7],     _[0..7],     0     . . .  Ac16:  [[1,2,4..10,12..43,45..66,70..75,78..80,...]]
_[0..7],     _[0..7],     0     . . .  Ac17:  [[1,2,4..10,12..43,45..66,70..75,78..80,...]]
_[0..7],     _[0..7],     0     . . .  Ac18:  [[1,2,4..10,12..43,45..66,70..75,78..80,...]]
_[0..7],     _[0..7],     0     . . .  Ac19:  [[1,2,4..10,12..43,45..66,70..75,78..80,...]]
_[0..7],     _[0..7],     0     . . .  Ac20:  [[1,2,4..10,12..43,45..66,70..75,78..80,...]]
_[0..7],     _[0..7],     0     . . .  Ac21:  [[1,2,4..10,12..43,45..66,70..75,78..80,...]]
_[0..7],     _[0..7],     0     . . .  Ac22:  [[1,2,4..10,12..43,45..66,70..75,78..80,...]]
_[0..7],     _[0..7],     0     . . .  Ac23:  _[[1,2,4..10,13..34,...]]
_[0..7],     _[0..7],     0     . . .  Ac24:  _[[5..8,13..21,23,26,...]]
_[0..7],     _[0..7],     0     . . .  Ac25:  _[[5,6,13..20,26,27,...]]
_[[1..4,6]],  _[1..5],    0     . . .  Ac26:  _[[5,6,14,16,26,27,...]]
_[[2,4,6]],  _[[1,2,4]],  0     . . .  Ac27:  _[[14,16,56,59,68,...]]
4          4          0     . . .
```

Figure 2: Partial solution matrix with goal locking applied.

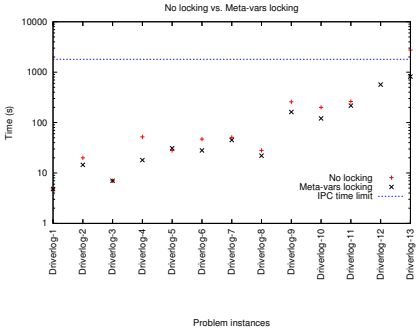

Figure 3: Runtime with & without meta-variables.

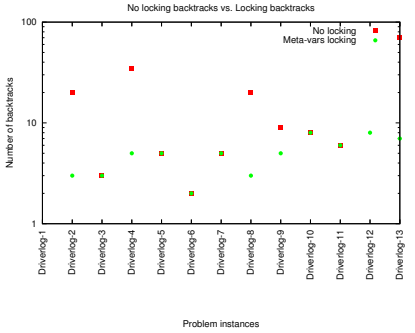

Figure 4: Backtracks with & without meta-variables.

4 Conclusion

Depending on the structure of a particular problem, action pruning resulting from the increased inferences (due to the use of meta variables) may lead to a reduction in the amount of both runtime and backtracking required to find a solution to the given problem. However, whilst such propagation may aid faster solution, the increased number of constraints and associated processing may be an overhead. Further, the overall impact will depend on the causal dependencies between as yet unassigned CSP variables and those recently locked.

References

[1] Christer Backstrom. Equivalence and Tractability Results for SAS+ Planning. In *Proceedings of KR'92*. Morgan Kaufmann, 1992.

[2] Rina Dechter. *Constraint processing*. Elsevier, 2003.

[3] R. E. Fikes and N. J. Nilsson. Strips. In *Readings in Planning*, pages 88–97. Kaufmann, San Mateo, CA, 1990.

[4] Eugene C. Freuder. Eliminating interchangeable values in csps. In *Procs of the 9th National Conference on AI - Volume 1*, 1991.

[5] Peter Gregory, Derek Long, and Maria Fox. A Meta-CSP Model for Optimal Planning. In *Proceedings of Abstraction, Reformulation and Approximation, Seventh International Symposium*, 2007.

[6] Malte Helmert and Silvia Richter. Fast Downward. In *Proceedings of the International Planning Competition 2004*, pages 41–43, 2004.

[7] Mark Judge. Constraint-based Heuristic Guidance for Solution of AI Planning Problems. In *Procs of International IEEE / EPSRC Workshop on Autonomous Cognitive Robotics. Univ of Stirling*, 2014.

[8] Mark Judge and Derek Long. A CSP Heuristic for AI Planning. In *Procs of 20th Automated Reasoning Workshop, School of Computing, University of Dundee, UK*, 2013.

[9] Craig Knoblock. Pddl. *AIPS98*, 78(4):1–27, 1998.

Computing Uniform Interpolants of $\mathcal{ALCH}$-Ontologies with Background Knowledge

Patrick Koopmann and Renate A. Schmidt

University of Manchester, UK
{koopmannp,schmidt}@cs.man.ac.uk

Abstract: Uniform interpolation is a technique to restrict the concept and roles symbols used in an ontology to a specified subset, while preserving all logical entailments that can be expressed using that subset. We propose the notion of relative uniform interpolation, where knowledge from a second ontology is taken into account, and present a method for the description logic $\mathcal{ALCH}$. Relative uniform interpolation of ontologies corresponds to strongest necessary conditions of formulae studied in classical logics.

1 Introduction

Uniform interpolation deals with the problem of restricting the symbols used in an ontology to a given set in such a way, that all entailments in that signature are preserved. It has applications in a range of areas covering ontology analysis, hiding confidential information, ontology reuse and ontology evolution, and methods have been developed for various description logics [5, 3, 2].

A topic that has only gained little attention in the past is computing uniform interpolants in the presence of an ontology with background knowledge. For example, it is possible that an application requires only the computation of the uniform interpolant of a subset of the ontology, whereas information about the terms to be forgotten is present in other parts of the ontology. We call these uniform interpolants *relative*, since they interpolate the ontology relative to a second ontology. Possible applications are analysing how certain concepts are related in a subset of the ontology, removing confidential information from ontologies to be shared, or approximating communication between agents that only share a limited set of common vocabulary.

Relative uniform interpolants have been investigated for classical logics under the name strongest necessary conditions (see for example [1]). We believe "relative uniform interpolant" is better suited for the description logic case, since it is unusual to speak of ontologies as sets of conditions.

We give a formal definition of relative uniform interpolants. Let $sig_c(E)$ denote the concept symbols occurring in E, where E is an ontology or a concept inclusion.

Definition 1. *Given two ontologies $\mathcal{O}$ and $\mathcal{O}_b$ and a set of concept symbols $\mathcal{S}$, an ontology $\mathcal{O}_{rui}$ is a* uniform interpolant of $\mathcal{O}$ for $\mathcal{S}$ relative to $\mathcal{O}_b$, *iff the following conditions are satisfied:*

1. $sig_c(\mathcal{O}_{rui}) \subseteq \mathcal{S}$
2. *For every axiom α with $sig_c(\alpha) \subseteq \mathcal{S}$, $\mathcal{O}_b \cup \mathcal{O}_{rui} \models \alpha$ iff $\mathcal{O}_b \cup \mathcal{O} \models \alpha$.*

If $\mathcal{O}_b$ is empty, $\mathcal{O}_{rui}$ is a uniform interpolant *of $\mathcal{O}$ for $\mathcal{S}$. We call $\mathcal{O}$ the* input ontology *and $\mathcal{O}_b$ the* background ontology.

Resolution:

$$\frac{C_1 \sqcup A \qquad C_2 \sqcup \neg A}{C_1 \sqcup C_2}$$

provided $C_1 \sqcup C_2$ contains at most one negative definer literal.

Role Propagation:

$$\frac{C_1 \sqcup \forall s.D_1 \qquad C_2 \sqcup \mathsf{Q}r.D_2}{C_1 \sqcup C_2 \sqcup \mathsf{Q}r.D_3} \qquad \mathcal{O} \models r \sqsubseteq s$$

where $\mathsf{Q} \in \{\exists, \forall\}$ and D_3 is a (possibly new) definer symbol representing $D_1 \sqcap D_2$, provided $C_1 \sqcup C_2$ contains at most one negative definer literal.

Figure 1: The calculus.

2 Computing Relative Uniform Interpolants

The method assumes that both the input ontology $\mathcal{O}$ and the background ontology $\mathcal{O}_b$ are normalised into a specific normal form, which is defined as follows. Let N_D be a set of special concept symbols that do not occur in the input ontology, called definer symbols.

Definition 2. *An $\mathcal{ALCH}$-*literal *is a concept description of the form A, $\neg A$, $\forall r.D$ or $\exists r.D$, where A is a concept symbol, r a role symbol and $D \in N_D$. A literal of the form $\neg D$, where $D \in N_D$, is called* negative definer literal. *A TBox is in $\mathcal{ALCH}$-*clausal form *if every axiom is of the form $\top \sqsubseteq L_1 \sqcup \ldots \sqcup L_n$, where each L_i is an $\mathcal{ALCH}$-literal and at most one L_i is a negative definer literal. The right part of such a concept inclusion is called $\mathcal{ALCH}$-*clause. *We assume that $\mathcal{ALCH}$-clauses are represented as sets of literals.*

Each ontology can be polynomially transformed into $\mathcal{ALCH}$-normal form by using standard flattening and conjunctive normal form transformations.

We first present the method for computing uniform interpolants relative to an empty background ontology (standard uniform interpolation) as described in [3].

In order to compute a uniform interpolant of an ontology $\mathcal{O}$ for $\mathcal{S}$, we *forget* each symbol $B \in sig(\mathcal{O}) \setminus \mathcal{S}$ one

Definer Purification:

$$\frac{\mathcal{T}}{\mathcal{T}^{[D\mapsto\top]}}$$

provided D occurs only positively in $\mathcal{T}$.

Non-Cyclic Definer Elimination:

$$\frac{\mathcal{T}\cup\{D\sqsubseteq C\}}{\mathcal{T}^{[D\mapsto C]}}$$

provided $D\notin sig_c(C)$.

Cyclic Definer Elimination:

$$\frac{\mathcal{T}\cup\{D\sqsubseteq C[D]\}}{\mathcal{T}^{[D\mapsto\nu X.C[X]]}}$$

Figure 2: Definer elimination.

after another using the rules in Figure 1.

The role propagation rule may involve the introduction of a new definer symbol D_{12} representing the conjunction $D_1\sqcap D_2$, which is performed by adding new clauses $\neg D_{12}\sqcup D_1$ and $\neg D_{12}\sqcup D_2$. By introducing only one definer symbol per pair, the number of introduced definer symbols is limited by a finite bound. This way, the number of symbols used in the derived clause set is always finite, which also gives a finite bound on the number of clauses and ensures termination.

In order to forget a symbol B, we only apply resolution on B or definer symbols, and only apply the role propagation rule if that enables us to do further resolution steps. Afterwards all clauses containing B are filtered out, as well as clauses of the form $\neg D\sqcup D'$, which were added when a new definer symbol was introduced.

After all symbols have been forgotten, the definers symbols are eliminated. For this we replace for each definer symbol D the clauses of the form $\neg D\sqcup C_i$ by a single concept inclusion $D\sqsubseteq\bigsqcap_i C_i$, and then apply the rules in Figure 2 exhaustively. The cyclic definer elimination rule introduces fixpoint operators to the result, which is in general unavoidable if we want to represent the uniform interpolant finitely. It is however possible to represent the result without fixpoints by approximation or using helper concepts [3]. The method can be further optimised by using redundancy elimination techniques as described in [3].

In order to compute uniform interpolants relative to non-empty background ontologies, we follow a set of support strategy. The background ontology $\mathcal{O}_b$ is transformed into a set of clauses $\mathbf{N}_b$ and the input ontology $\mathcal{O}$ into a set of clauses $\mathbf{N}_k$. Both sets will be updated during the computation, where $\mathbf{N}_b$ serves as set of support, and $\mathbf{N}_k$ contains the clausal form of the relative uniform interpolant in the end.

When forgetting a concept symbol B, the rules are applied with the additional requirement that at most one premise is taken from the set $\mathbf{N}_b$, and derived clauses are added to $\mathbf{N}_k$. If a clause of the form $\neg D\sqcup C$, $D\in N_D$, is added to $\mathbf{N}_k$, all clauses from $\mathbf{N}_b$ that contain D are moved to $\mathbf{N}_k$. This step is necessary to ensure that no information is lost when eliminating the definer symbols in the last step. After having forgotten a concept symbol B this way, all clauses containing B are moved to $\mathbf{N}_b$.

Through resolution with clauses from the set $\mathbf{N}_b$, it is possible that previously forgotten symbols get reintroduced to $\mathbf{N}_k$. For this reason, the process has to be repeated until no clause in $\mathbf{N}_k$ contains a symbol that is neither a definer symbol nor in $\mathcal{S}$, where we take care that we do not add any clauses to $\mathbf{N}_k$ that have previously been moved to $\mathbf{N}_b$. Since there is a finite bound on the set of clauses that can be derived using our calculus, this process always terminates. From the final clause set $\mathbf{N}_k$, the relative uniform interpolant $\mathcal{O}_{rui}$ is computed by eliminating all definer symbols using the rules in Figure 2.

3 Outlook

We implemented a prototype of the presented method based our implementation for computing uniform interpolants [4] and did some first experiments for small subsets of ontologies, the remaining part of the ontology was taken as background ontology. Sometimes a lot of information was transferred from the background ontology into the uniform interpolant, causing many iterations for each symbol to be forgotten, but usually we could compute relatively small relative uniform interpolants.

While the presented method extends our method for computing uniform interpolants in $\mathcal{ALCH}$, the key idea can be used with similar resolution based approaches, as for example for computing uniform interpolants of $\mathcal{SHQ}$-ontologies [5] or ontologies with ABoxes.

Open questions are how the method would perform in a deeper evaluation and whether the computed relative uniform interpolants are minimal in the sense that only as much information as needed is used from the background ontology.

References

[1] Patrick Doherty, Witold Lukaszewicz, and Andrzej Szalas. Computing strongest necessary and weakest sufficient conditions of first-order formulas. In *Proc. IJCAI'01*, pages 145–154. Springer, 2001.

[2] Boris Konev, Dirk Walther, and Frank Wolter. Forgetting and Uniform Interpolation in Large-Scale Description Logic Terminologies. In *Proc. IJCAI '09*, pages 830–835. AAAI Press, 2009.

[3] Patrick Koopmann and Renate A Schmidt. Forgetting concept and role symbols in $\mathcal{ALCH}$-ontologies. In *Proc. LPAR'13*, pages 552–567. Springer, 2013.

[4] Patrick Koopmann and Renate A. Schmidt. Implementation and evaluation of forgetting in $\mathcal{ALC}$-ontologies. In *Proc. WoMO'13*. CEUR-WS.org, 2013.

[5] Patrick Koopmann and Renate A. Schmidt. Count and forget: Uniform interpolation of $\mathcal{SHQ}$-ontologies. In *Proc. IJCAR'14*. Springer, 2014. To appear.

On Herbrand theorems for classical and non-classical logics

Alexander Lyaletski

Faculty of Cybernetics, Taras Shevchenko National University of Kyiv
4D, Hlushkov avenue, 03680 Kyiv, Ukraine
lav@unicyb.kiev.ua

Abstract: The talk is devoted to Herbrand-type theorems for an wide enough spectrum of first-order logics for both the classical and intuitionistic cases as well as for their modal extensions. A way for the construction of Herbrand theorems for first-order logics containing the ineliminable cut rule is discussed.

1 Introduction

The purpose of this talk is to present a general way for the construction of Herbrand theorems for an wide enough spectrum of first-order logics having the form of sequent calculi [1]. The research is based on some of the author's results relating to the construction of computer-oriented first-order sequent calculi not requiring performing preliminary skolemization before the starting of deduction.

As in the case of the original Herbrand theorem [2], any Herbrand theorem under consideration presents itself a (reduction) theorem reducing the problem of the deducibility of an initial sequent S of the form $\to F$ (where F is a closed formula) in a first-order sequent calculus to the problem of the deducibility of a certain quantifier-free sequent (being constructed on the basis of S) in its quantifier-free-rule part.

Herbrand theorems are divided into two classes: one contains the theorems for first-order classical (modal) logics and the other contains theorems for their intuitionistic modifications. Initially wording for logics without equality, all such results then are extended to logics with equality.

2 Preliminaries

The *signature* of any (sequent) logic contains: a finite (possibly, empty) set of function symbols; a finite non-empty set of predicate symbols containing, perhaps, the equality; all the logical connectives including the universal and existential quantifiers; and modal operators. A specifity of our formalism is that it uses *multisets* of formulas instead of sequences of formulas. This leads to that all the axioms of any logic have the form $\Gamma, A \to A, \Delta$, where Γ and Δ are multisets of formulas and A is an atomic formula including $t = t$ for any term t not containing so-called dummies [3] (i.e. variables admitting their replacement by any terms) in the case of equality.

Besides of the axioms, any sequent calculus contains the *usual inference rules* for all the logical connectives and, perhaps, the *cut rule*, which may be ineliminable in it or its expansion. Any logic *with equality* contains only the Kanger-type equality rules [3]. All the *modal logics* being considered in the paper are constructed from classical or intuitionistic ones (with or without equality) by adding specific inference rules for modal connectives. Any inference is supposed to have the form of an *inference tree* called a *proof tree*, if all its leaves are axioms.

For obtaining Herbrand theorems in the form of reduction, the usual notion of an *Herbrand universe* H_F for F (consisting of terms) is modified in a certain way and the notion of an *Herbrand expansion* of F being the result of (multiple) "doubling" of certain subformulas of F is introduced. The result of a Herbrand reduction for $\to F$ can be presented in the form $\to M \cdot \sigma$, where M is the result of the removing of all the quantifiers from an expansion F' of F, σ is a substitution of terms from $H_{F'}$ for all the dummies from M, and $M \cdot \sigma$ is the result of the applying of σ to M. Namely, the existence of a proof tree (satisfying certain restrictions) for $\to M \cdot \sigma$ in the propositional part of a calculus under consideration guarantees the deducibility of $\to F$ in the calculus.

It is well known that in classical logic the deducibility (validity) of any formula is invariant w.r.t. the skolemization operation. In this connection, most of Herbrand-type theorems for classical logic are formulated in the form of reduction theorems using skolemization. But already in the case of (usual) intuitionistic logic, deducibility (validity) is not invariant w.r.t. skolemization in general, which makes us look for ways to get Herbrand theorems for the intuitionistic logic and different modal extensions without performing preliminary skolemization. Attempts to move in this direction were made in some of the author's papers relating to inference search in non-classical (and classical) logics without equality, in which the original notions of the *admissibility* of a substitution (of terms for variables) for a formula (sequent) and the *compatibility* of an inference tree with a substitution were used. Namely, both these notions and analogues of a usual Herbrand universe and Herbrand expansion lead to the opportunity to obtain Herbrand theorems for all the logics under consideration.

To obtain Herbrand theorems for the logics, in which the cut rule cannot be eliminated, it is suggested to introduce the notion of a *tautological expansion* of an initial formula F, according to which formulas of the form $G \supset G$, where the structure of G is connected with F, are allowed to use for the construction of a Herbrand expansion of F.

3 Herbrand theorems for logics without equality

For modal intuitionistic logics without equality rule, we have the following results.

Theorem 1. Let **SC** be a first-order intuitionistic modal sequent logic with maybe the cut. For a closed formula F, the sequent $\rightarrow F$ is deducible in **SC** if and only if there are a sequence of tautological and Herbrand expansions of F leading to the construction of a formula F' and a substitution σ of terms from a Herbrand universe $H_{F'}$ for all the dummies of M being the result of removing all the quantifiers from F' such that

(i) there exists a proof tree Tr for $M \cdot \sigma$ in the propositional part of **SC**,

(ii) σ is an admissible substitution for F', and

(iii) the tree Tr is compatible with the substitution σ.

For other classes of logics without equality, we have:

(1) In the case of intuitionistic (modal) logics containing the eliminable cut, reminder about tautological expansion can be removed from Theorem 1.

(2) In the case of classical modal logics containing the ineliminable cut, the item (iii) can be removed Theorem 1.

(3) In the case of classical modal logics (or simply classical logic) containing the eliminable cut, both the reminder about tautological expansion and the item (iii) can be removed from Theorem 1.

(4) In the case of logics containing the ineliminable cut and admitting skolemization, the items (ii) and (iii) can be removed from Theorem 1, if skolemization is applied.

(5) In the case of logics containing the eliminable cut and admitting skolemization, both the reminder about tautological expansion and the items (ii) and (iii) can be removed from Theorem 1, if skolemization is applied.

Applying Theorem 1 to Gentzen's calculi **LK** and **LJ** without equality [1], we obtain the Herbrand theorems stating that the deducibility of sequents in **LK** and **LJ** is equivalent to the deducibility of certain quantifier-free sequents in their propositional parts.

4 Herbrand theorems for logics with equality

In the case of the logics with equality, the Herbrand universe H_F for a formula F is partitioned into disjoint classes of terms in accordance with the set of all the so-called negative "equality atoms" of F and a substitution σ selected for replacement of all the dummies of F by terms from H_F. The result of the applying of this partition operation is denoted by $\pi(H_F, \sigma)$. If M is the result of removing all the quantifiers from F then $[M \cdot \sigma]/\pi(H_F, \sigma)$ is defined as the result of the replacement of every term that occupies an argument place in $M \cdot \sigma$ by the class containing this term and being considered as a constant in the formula $[M \cdot \sigma]/\pi(H_F, \sigma)$.

The specifity of any logic with equality under consideration is that the deducibility of an initial sequent $\rightarrow F$ in a usual sequent logic with the equality (for example, in Getzen's logic **LK** or **LJ** with equality) is equivalent to the deducibility of the sequent $\rightarrow \forall x(x = x) \supset F$ in an appropriate calculus $\mathbf{SC}^{=}$ under consideration containing only Kanger's rules for equality handling.

Theorem 2. Let $\mathbf{SC}^{=}$ be a first-order intuitionistic modal sequent logic with equality and maybe the cut. For a closed formula F, the sequent $\rightarrow \forall x(x = x) \supset F$ is deducible in $\mathbf{SC}^{=}$ if and only if there exist a sequence of tautological and Herbrand expansions of $\rightarrow \forall x(x = x) \supset F$ leading to the construction of a formula F' and a substitution σ of terms from a Herbrand universe $H_{F'}$ for all the dummies of M being the result of removing all the quantifiers from F' such that

(i) there exists a proof tree Tr for $[M \cdot \sigma]/\pi(H_F, \sigma)$ in the propositional fragment of $\mathbf{SC}^{=}$,

(ii) σ is an admissible substitution for F', and

(iii) tree Tr is compatible with the substitution σ.

For other classes of logics with equality, we have:

(1) In the case of intuitionistic (modal) logics containing the eliminable cut, reminder about tautological expansion can be removed from Theorem 2.

(2) In the case of classical modal logics containing the ineliminable cut, the item (iii) can be removed Theorem 2.

(3) In the case of classical modal logics (or simply classical logic) containing the eliminable cut, both the reminder about tautological expansion and the item (iii) can be removed from Theorem 2.

(4) In the case of logics containing the ineliminable cut and admitting skolemization, the items (ii) and (iii) can be removed from Theorem 2, if skolemization is applied.

(5) In the case of logics containing the eliminable cut and admitting skolemization, both the reminder about tautological expansion and the items (ii) and (iii) can be removed from Theorem 2, if skolemization is applied.

Applying Theorem 2 to Gentzen's calculi **LK** and **LJ** with equality [1], we obtain the Herbrand theorems stating that the deducibility of sequents in **LK** and **LJ** with equality is equivalent to the deducibility of certain quantifier-free sequents in their propositional parts not requiring equality rule applications.

When proving the results, all the reasoning is made on deducibility. If for a certain sequent calculus under consideration, there is a theorem on its soundness and completeness, we automatically obtain a corresponding Herbrand theorem on validity. The calculi **LK** and **LJ** (with and without equality) can serve as such examples.

References

[1] G. Gentzen, Untersuchungen uber das Logische Schliessen, *Math. Z.*, 39 (1934-5): 176–210, 405–431, 1935.

[2] J. Herbrand, Recherches sur la theorie de la demonstration, *Travaux de la Societe des Sciences et de Lettres de Varsovie, Class III, Sciences Mathematiques et Physiques*, Vol. 33. (Also see in English: Logical Writing, Hinghan, Reidel Publishing Company, 1971).

[3] S. Kanger, Simplified proof method for elementary logic, *Computer Programming and Formal Systems: Studies in Logic*, 87–93, North-Holland, 1963.

Extended Resolution in Modern SAT Solving

Norbert Manthey

Knowledge Representation and Reasoning Group
Technische Universität Dresden
norbert.manthey@tu-dresden.de

Abstract: Modern SAT solvers are applied in many academic and industrial fields. The CDCL algorithm, which is employed by most SAT solvers, is based on resolution, but solvers also use stronger reasoning techniques in preprocessing steps. Only few attempts have been made to include stronger reasoning techniques in the search itself. The current work revisits the embedding of extended resolution into SAT solvers, and improves existing work. Experiments indicate that the approach is very promising.

1 Introduction

SAT solvers are used in many academic and industrial fields, either directly to solve a given formula, or SAT solving technology is embedded into other reasoners, for example bounded model checker, higher order logic provers, ASP or SMT solvers [4]. The huge application field is due to the recent development starting from the well known DPLL procedure [6]: modern solvers use *clause learning* [13] as deduction, employ *restarts* [7] to avoid searching at the wrong place, and have sophisticated decision heuristics to drive their search.

From a proof complexity point of view, this combination is known to be as powerful as general resolution [12], whereas the DPLL algorithm is only as strong as tree-like resolution. Hence, a CDCL solver can produce an unsatisfiability proof for a formula with a length that is polynomial compared to the shortest resolution proof. However, for propositional formulas, proof theory provides stronger deduction techniques, as for example *cutting planes* [5], or *extended resolution* [14]. There exist formulas where these three systems are known to be able to produce exponentially shorter proofs than resolution. Most modern SAT solvers do not exploit this reasoning power, or just exploit it in a limited way: During a formula simplification phase, Lingeling [3] uses a form of cutting planes, and extended resolution is used in *bounded variable addition*(BVA) [11]. Sat4J [9] can use cutting planes during search, but reports a slowdown of multiple orders of magnitude. Finally, Huang [8] and Audemard et al. [1] presented how to use extended resolution in the CDCL algorithm: Huang replaces a disjunction of two literals with a *fresh variable*, and Audemard et al. replace a conjunction of two literals with a fresh variable. However, both approaches are not used in any modern SAT solver. Even worse, *restricted extended resolution* (RER) has been dropped from the solver Glucose again, due to its high code complexity. The presented work reimplements and analyzes RER. With the current state of the work, the performance of the SAT solver Riss can be improved slightly.[1] The implementation is a starting point to integrate extended resolution into CDCL SAT solvers.

[1] The source code of the RER extension in Riss is available at `http://tools.computational-logic.org`.

2 Extended Resolution

The SAT problem is to find a *model* for a propositional formula [4]. Formulas are sets of *clauses*, which are sets of *literals*. A literals x is a Boolean variable v, or a negated variable $\overline{v}$. Given two clauses $C = (l \vee E)$ and $D = (\overline{l} \vee F)$, then the resolvent R is $R = C \otimes D = (E \vee F)$. A resolution proof for a formula F is a sequence of resolvents R_i, where each resolvent is produced by resolving two clauses that are present either in the proof with a lower index, or in the formula F. Given a formula F, a variable v is *fresh*, if F does not contain v.

Let l_1 and l_2 be two literals that are present in the formula F. Extended resolution introduces a fresh variable v, and adds three clauses $(v \vee \overline{l_1})$, $(v \vee \overline{l_2})$, and $(\overline{v} \vee l_1 \vee l_2)$, to represent $v \leftrightarrow (l_1 \vee l_2)$ [14]. This functional dependency of v to l_1 and l_2 can be changed to any other Boolean function, and even the number of *input literals* can be modified, as in the simplification technique BVA [11].

2.1 Restricted Extended Resolution

Audemard et al. [1] introduce a fresh variable, if two consecutive learned clauses C_i and C_{i+j} share all except one literal: $C_i = (\overline{l_1} \vee D)$ and $C_{i+j} = (\overline{l_2} \vee D)$. The different literals l_1 and l_2 can occur only at the very first position. In the *local extension*, the fresh variable v represents the disjunction of l_1 and l_2: $v \leftrightarrow (l_1 \vee l_2)$. The corresponding clauses are added to the formula F. The clauses C_i and C_{i+j} are replaced by $C' = (\overline{v} \vee D)$, because C_i and C_{i+j} can be obtained by resolution from C' with the two binary extension clauses. Hence, by introducing a fresh variable, the number of clauses is reduced. The more the variable v is used further in the unsatisfiability proof, the higher is the potential to reduce the size of the proof.

Once an extension is made, all clauses C in the formula that contain l_1 and l_2 are rewritten to $C := (C \setminus \{l_1, l_2\}) \cup \{v\}$. Similarly to the two learned clauses, the clauses that have been rewritten can be obtained by resolution again. Furthermore, whenever a new clause is learned, its literals

are checked to contain the pair l_1 and l_2, so that this pair can be replaced by v. This check is done for all extensions that have been introduced. When an introduced variable becomes irrelevant, such a variables is removed again, and the original clauses are recovered, to decrease the cost of the check [1]. On average, Audemard et al. report an extension every 1000 conflicts.

Reducing the Procedural Overhead To find a matching pattern, Audemard et al. report a size based filter: if two consecutive learned clauses $C_i = (\overline{l_1} \vee D)$ and $C_{i+j} = (\overline{l_2} \vee E)$ do not have the same size, they are no candidates for an extension. Additionally, since D has to match E, a *Bloom filter* is introduced which checks the sums of the literals in D and E. The two sets D and E can only match, if $\sum_{l \in D} = \sum_{l \in E}$.

As presented in [11], increasing the number of variables in a formula to decrease the number of clauses in CDCL solvers does not influence the performance. Hence, the new RER implementation does not delete introduced variables. When an extension $v \leftrightarrow (l_1 \vee l_2)$ is added, for the smaller literal l_1 the literals l_2 and v are stored. When l_1 is used as the smaller literal in another extension with l_2 and w, then the old extension is not used for new clauses any more. Thus, the check for new clauses becomes less costly, and the cost for keeping all introduced variables is very low.

3 Evaluation and Future Work

The modified version of RER is implemented into the SAT solver Riss [10], and evaluated on the instances of the SAT competition with a 3600 s timeout. The results for satisfiable ($\top$) and unsatisfiable formulas ($\bot$) are presented in Table 1. Surprisingly, RER helps to solve more satisfiable instances. Compared to the results of [1], on the whole benchmarks an extension is performed at most every 1000 conflicts. Most of the time, less than 10 extensions are made. Furthermore, on crafted formulas more extensions are made than on application formulas.

Riss is a state-of-the-art SAT solver, and with the given implementation, a starting point for further research is given. With the current implementation, heuristics for deciding when to introduce a fresh variable with extended resolution can be developed, and tested. Furthermore, new functional dependencies can be tested. As a first step, and to stay as close as possible to RER, any functional dependency for a fresh variable v that can be hidden in two consecutive learned clauses should be considered. Candidates for such extensions are XOR gates, $v \leftrightarrow (a \oplus b)$, or If-Then-Else(ITE) gates, $v \leftrightarrow \mathrm{ITE}(s, t, f)$. Both gates appear frequently in propositional formulas from cryptographic instances and scheduling, because ITE gates are used to encode binary decision diagrams into SAT. A first attempt to use ITE gates similarly to the conjunction of two literals in RER did not lead to any improvements. We believe that good heuristics and useful functional dependencies can be discovered by automatic configuration tools. Then, the addition of extended resolution to the CDCL algorithm might result in the next exponential boost of SAT solvers.

Table 1: Evaluation on recent competition benchmarks.

		Riss		Riss+RER	
Category	Year	$\top$	$\bot$	$\top$	$\bot$
Crafted	2011	79	42	**81**	42
	2012	**229**	145	228	**146**
	2013	104	84	**106**	**86**
Application	2011	86	**117**	**89**	116
	2012	237	296	**243**	296
	2013	95	86	**98**	86

Acknowledgements

The author thanks the ZIH of TU Dresden for providing the computational resources for the empirical evaluation.

References

[1] Gilles Audemard, George Katsirelos, and Laurent Simon. A restriction of extended resolution for clause learning sat solvers. AAAI Press, 2010.

[2] A. Balint, A. Belov, M. J.H. Heule, and M. Järvisalo, editors. *Proceedings of SAT Challenge 2013*, volume B-2013-1 of *Department of Computer Science Series of Publications B*. University of Helsinki, Helsinki, Finland, 2013.

[3] A. Biere. Lingeling, Plingeling and Treengeling entering the SAT competition 2013. In Balint et al. [2], pages 51–52.

[4] Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors. *Handbook of Satisfiability*. IOS Press, Amsterdam, 2009.

[5] W. Cook, C.R. Coullard, and Gy. Turán. On the complexity of cutting-plane proofs. *Discrete Applied Mathematics*, 18(1):25 – 38, 1987.

[6] M. Davis, G. Logemann, and D. Loveland. A machine program for theorem-proving. *Commun. ACM*, 5(7):394–397, 1962.

[7] Jinbo Huang. The effect of restarts on the efficiency of clause learning. In *IJCAI*, pages 2318–2323, 2007.

[8] Jinbo Huang. Extended clause learning. *Artif. Intell.*, 174(15):1277–1284, October 2010.

[9] Daniel Le Berre and Anne Parrain. The SAT4J library, release 2.2, system description. *JSAT*, 7:59–64, 2010.

[10] N. Manthey. The SAT solver RISS3G at SC 2013. In Balint et al. [2], pages 72–73.

[11] Norbert Manthey, Marijn J. H. Heule, and Armin Biere. Automated reencoding of Boolean formulas. In *HVC 2012*, 2012.

[12] Knot Pipatsrisawat and Adnan Darwiche. On modern clause-learning satisfiability solvers. *J. Autom. Reasoning*, 44(3):277–301, 2010.

[13] João P. Marques Silva and Karem A. Sakallah. GRASP - a new search algorithm for satisfiability. In *ICCAD 1996*, pages 220–227, Washington, 1996. IEEE Computer Society.

[14] G. Tseitin. On the complexity of proofs in propositional logics. In J. Siekmann and G. Wrightson, editors, *Automation of Reasoning: Classical Papers in Computational Logic 1967–1970*, volume 2. Springer-Verlag, 1983.

A Resolution-Based Prover for Normal Modal Logics

Cláudia Nalon[1] George Bezerra Silva[1]

Department of Computer Science
University of Brasília, Brazil
nalon@unb.br, georgelione@gmail.com

Abstract: We present a prototype tool for automated reasoning for multimodal normal logics where combinations of the axioms **K**, **T**, **D**, **B**, **4**, and **5** hold. The theorem prover is based on previous work on resolution calculi for such logics. We briefly present the syntax, the semantics, and the calculus for the basic normal logic together with the inference rules for dealing with each specific axiom. We then give details of the implementation of the prover and discuss future work.

1 Introduction

In [1], sound, complete, and terminating resolution-based methods for fifteen families of propositional normal modal logics are presented. These calculi deal with multimodal logics K_n, in which the schemata $\Box_a(\varphi \Rightarrow \psi) \Rightarrow (\Box_a\varphi \Rightarrow \Box_a\psi)$ ($\mathbf{K}_a$), where φ and ψ are well-formed formulae, are valid, and the extensions of K_n where the (combination of the) following axioms for reflexive ($\mathbf{T}_a$: $\Box_a\varphi \Rightarrow \varphi$), serial ($\mathbf{D}_a$: $\Box_a\varphi \Rightarrow \Diamond_a\varphi$), symmetric ($\mathbf{B}_a$: $\Diamond_a\Box_a\varphi \Rightarrow \varphi$), transitive ($\mathbf{4}_a$: $\Box_a\varphi \Rightarrow \Box_a\Box_a\varphi$), and Euclidean systems ($\mathbf{5}_a$: $\Diamond_a\varphi \Rightarrow \Box_a\Diamond_a\varphi$) also hold.

Here, we present a prototype theorem-prover for those families of logics using the methods given in [1]. In the next section, we present the language of K_n. In Section 3, we introduce the resolution-based method for K_n and the rules for dealing with the axioms given above. In Section 4, we introduce the theorem-prover for K_n, giving details of its implementation. Conclusions are given in Section 5.

2 The Normal Logic K_n

Formulae in K_n are constructed from a denumerable set of *propositional symbols*, $\mathcal{P} = \{p, q, p', q', p_1, q_1, \ldots\}$. Besides classical connectives ($\neg, \wedge$), a set of unary modal operators $\Box_a$, $a \in \mathcal{A}$, is introduced, where $\Box_a\varphi$ is read as "the agent a considers φ necessary" and $\mathcal{A} = \{1, \ldots, n\}$ is the set of agents.. The operator $\Diamond_a\varphi$ is an abbreviation for $\neg\Box_a\neg\varphi$. The set of well-formed formulae, WFF, is defined in the usual way: $p \in \mathcal{P}$ is in WFF; **true** is in WFF; if φ and ψ are in WFF, then so are $\neg\varphi$, $(\varphi \wedge \psi)$, and $\Box_a\varphi$, for all $a \in \mathcal{A}$. A *literal* is either p or $\neg p$, for $p \in \mathcal{P}$. $\mathcal{L}$ is the set of literals. A *modal literal* is either $\Box_a l$ or $\neg\Box_a l$, $l \in \mathcal{L}$.

A *Kripke structure* M *over* $\mathcal{P}$ is a tuple $M = \langle \mathcal{S}, \pi, \mathcal{R}_1, \ldots, \mathcal{R}_n \rangle$, where $\mathcal{S}$ is a set of possible *worlds* (or *states*) with a distinguished world s_0; the function $\pi(s) : \mathcal{P} \to \{\mathit{true}, \mathit{false}\}$, $s \in \mathcal{S}$, is an interpretation that associates with each state in $\mathcal{S}$ a truth assignment to propositions; and each $\mathcal{R}_a \subseteq S \times S$ is a binary relation on $\mathcal{S}$, where $a \in \mathcal{A}$.

We write $(M, s) \models \varphi$ to say that φ is true at world s in the Kripke structure M. Truth of classical formulae is given as usual; for modal formulae, we have that $(M, s) \models \Box_a\varphi$ iff for all t, such that $(s, t) \in \mathcal{R}_a$, $(M, t) \models \varphi$. The formulae **false**, $(\varphi \vee \psi)$, and $(\varphi \Rightarrow \psi)$ are introduced as the usual abbreviations for $\neg$**true**, $\neg(\neg\varphi \wedge \neg\psi)$, and $(\neg\varphi \vee \psi)$, respectively. Formulae are interpreted with respect to the distinguished world s_0. Let $M = \langle \mathcal{S}, \pi, \mathcal{R}_1 \ldots, \mathcal{R}_n \rangle$ be a Kripke structure with a distinguished world s_0. A formula φ is said to be *satisfiable in M* if $(M, s_0) \models \varphi$; φ is said to be *satisfiable* if there is a model M such that $(M, s_0) \models \varphi$; and φ is said to be *valid* if for all models M, $(M, s_0) \models \varphi$.

3 The Resolution Method for K

The resolution method is applied to formulae in the Separated Normal Form for Normal Logics (SNF_K). A nullary connective **start** is introduced to deal with reasoning within the distinguished world s_0; formally, $(M, s) \models$ **start** iff $s = s_0$. A formula in SNF_K is a conjunction of clauses, in one of the following forms: *initial* (**start** $\Rightarrow \bigvee_{b=1}^{r} l_b$), *literal* (**true** $\Rightarrow \bigvee_{b=1}^{r} l_b$), *positive modal* ($l' \Rightarrow \Box_a l$), or *negative modal* ($l' \Rightarrow \neg\Box_a l$), where l, l', $l_b \in \mathcal{L}$. Transformation rules and their correctness can be found in [1].

Once a formula has been transformed in its normal form, the inference rules are applied to the set of clauses until either a contradiction is found (in the form of **true** $\Rightarrow$ **false** or **start** $\Rightarrow$ **false**) or no new clauses can be generated. If a contradiction is found when the method is applied to a set of clauses $\mathcal{S}$, we say that there is a *refutation* for $\mathcal{S}$. The inference rules are shown in Figure 1, where l, l', l_i, $l'_i \in \mathcal{L}$ ($i \in \mathbb{N}$) and D, D' are disjunctions of literals. Figure 2 shows the inference rules for dealing with the satisfiability problem in systems where reflexivity ([REF]), seriality ([SER]), symmetry ([SYM]), transitivity ([TRANS]), and Euclideanness ([EUC1] and [EUC2]) hold. The resolvents of the inference rules [TRANS], [EUC1], and [EUC2] introduce literals of the form $pos_{a,l}$ and $nec_{a,l}$, which are used to keep the normal form of the resolvents by renaming the formulae $\neg\Box_a\neg l$ and $\Box_a l$, respectively,

4 The Prover

The theorem prover for K_n together with rules to deal with systems where the axioms **K**, **T**, **D**, **B**, **4**, and **5** hold can be found in [2]. The prover, which has been implemented

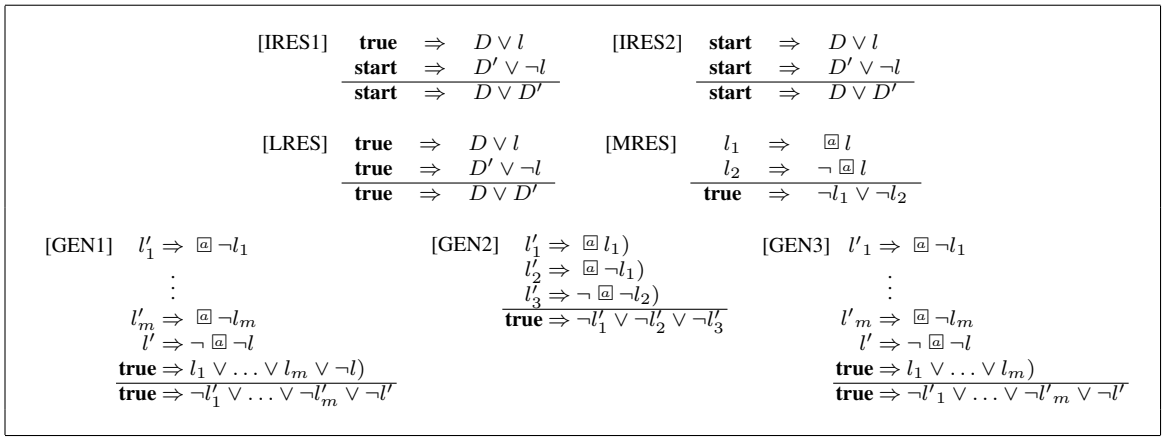

Figure 1: Inference Rules for K

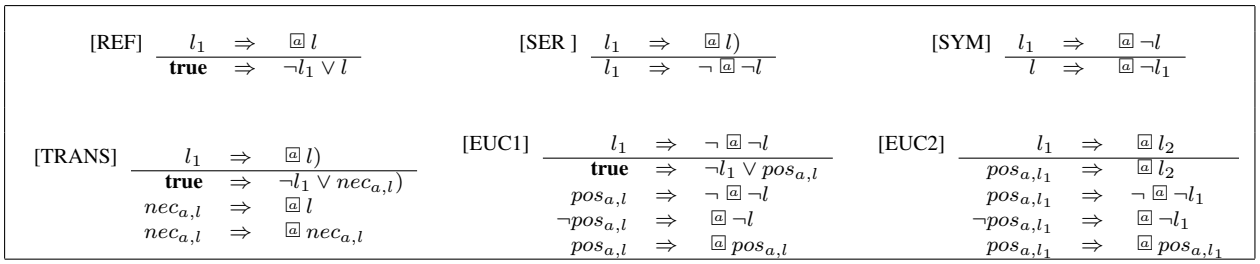

Figure 2: Inference Rules for Other Systems

in C++, takes as input a file with formulae in the language of K_n (where several different notations are allowed). For each formula, the prover carries out the transformation into the normal form and applies the resolution method, taking into consideration all inference rules specified as arguments at a command line. For instance, with the argument `-kt4`, the prover applies all the inference rules for K_n and the rules [REF] and [TRANS]. The prover has several levels of verbosity: only the result (satisfiable/unsatisfiable), the proof (or the last attempt of a proof), or all the steps in the attempt of finding a proof.

The prover cycles over the set of clauses applying, firstly, the inference rules [SER], [SYM], [TRANS], [EUC1], and [EUC2], until no new clauses can be found. Note that these rules are applied to modal clauses only, so they can all be applied before the cycle corresponding to the resolution method for $\mathsf{K}_{(n)}$, as the resolution rules for this system (i.e. [IRES1], [IRES2], [MRES], [LRES], [GEN1], [GEN2], and [GEN3]) do not generate modal clauses. For the same reason, the rules [MRES] and [GEN2] are also applied before the other rules. The prover for K_n then tries to employ linear resolution to the propositional part of the language as far as possible, but does not backtrack. If the last generated clause cannot be resolved with any other clause, then modal resolution is exhaustively applied and a new cycle of linear resolution over the literal clauses begins. Backtracking only takes place if literal and modal resolution cannot be further applied. Initial resolution consists on checking if the literal, which renames the original formula and is introduced during translation, appears negated in the set of literal clauses. Forward subsumption is implemented for the set of literal clauses.

5 Conclusion

The theorem-prover for K_n performs well and the implementation of the linear strategy helps the prover to make better use of space. Nevertheless, the implemented prover is not robust enough to deal with large formulae and is currently under revision. The architecture of the prover is also to be changed to allow for an easier inclusion of new inference rules. Moreover, all the features of the present version are hard-coded in the prover. We therefore intend to provide more options for testing some features of the calculi as, for instance, giving priorities for inference rules, better selection of clauses and literals (e.g. by using ordered resolution), and the use of pure set of support instead of the combination with linear resolution. Another desirable feature is to allow for modal operators under different logics (e.g. $\boxed{1}$ in KD_n and $\boxed{2}$ in $\mathsf{S5}_n$).

References

[1] C. Nalon and C. Dixon. Clausal resolution for normal modal logics. *J. Algorithms*, 62:117–134, July 2007.

[2] G. B. Silva. Implementação de um provador de teoremas por resolução para lógicas modais normais. Monografia de Conclusão de Curso, Universidade de Brasília, 2013. Prover available at `http://www.cic.unb.br/~nalon/#software`.

Models Minimal Modulo Subset-Simulation for Expressive Propositional Modal Logics

Fabio Papacchini Renate A. Schmidt

School of Computer Science, The University of Manchester, {papacchf,schmidt}@cs.man.ac.uk

Abstract: Terminating procedures for the generation of models minimal modulo subset-simulation for normal modal logics have recently been presented. In this abstract we explain what are the challenges to generalise those procedures to more expressive modal logics, and discuss possible solutions.

1 Introduction

Model generation and minimal model generation are useful for computer science tasks such as fault analysis, model checking and debugging of logical specifications [8, 4]. For this reason, there have been several studies on minimal model generation for classical and non-classical logics [1, 5, 6, 7, 4, 2].

[7] presents terminating, minimal model sound and complete procedures for the generation of minimal models for all the normal modal logics in between $\mathbf{K}$ and $\mathbf{S5}$, where a semantic notion of minimality, similar to the notions in [6, 4], is used. The minimality criterion is designed so that minimal models are semantically meaningful, more natural than models minimal with respect to other minimality criteria, and contain a minimal amount of information. The procedures in [7] are a combination of tableaux calculi and a minimality test to close branches.

In this abstract we discuss what are the main challenges that need to be faced to generalise the procedures in [7] to more expressive modal logics. Specifically, we are interested in extensions to multi-modal logics with universal modalities and inclusion axioms. Our ultimate aim is to obtain terminating, minimal model sound and complete procedures for all these generalisations.

2 Logics and Minimality Criterion

Syntax of modal formulae is defined as usual. Only one remark needs to be made to avoid confusion. That is, we use the notation $[\mathcal{U}]$ and $\langle\mathcal{U}\rangle$ for the universal modalities, and we use $[R_i]$ and $\langle R_i\rangle$ for the other modalities.

An inclusion axiom has the following form.

$$[R_i]\phi \rightarrow [R_1]\dots[R_n]\phi.$$

Such axioms represent frame properties of the form $R_1 \circ \dots \circ R_n \subseteq R_i$, where $\circ$ denotes relational composition.

We adopt the standard Kripke semantics of modal formulae. An *interpretation* $\mathcal{I}$ is a tuple $(W, \mathcal{R}, V)$, where W is a non-empty set of worlds, $\mathcal{R}$ is a set of accessibility relations $R_i \subseteq W \times W$ over W, and V is an interpretation function that assigns to each world $u \in W$ a set of propositional symbols, meaning that such propositional symbols hold in u. Given an interpretation $\mathcal{I}$, a world u and a modal formula ϕ, if $\mathcal{I}, u \models \phi$, then $\mathcal{I}$ is a *model* for ϕ.

Let $\mathcal{I} = (W, \mathcal{R}, V)$ and $\mathcal{I}' = (W', \mathcal{R}', V')$ be two models of a modal formula ϕ. A *subset-simulation* is a binary relation $S \subseteq W \times W'$ such that for any two worlds $u \in W$ and $u' \in W'$, if uSu' then the following hold.

- $V(u) \subseteq V'(u')$, and
- if uR_iv for some $R_i \in \mathcal{R}$, then there exist a $v' \in W'$ and $R_i \in \mathcal{R}'$such that $u'R_iv'$ and vSv'.

If S is such that for all $u \in W$ there is at least one $u' \in W'$ such that uSu', then we call S a *full subset-simulation* from $\mathcal{I}$ to $\mathcal{I}'$. Given two models $\mathcal{I}$ and $\mathcal{I}'$, if there is a full subset-simulation S from $\mathcal{I}$ to $\mathcal{I}'$, we say that $\mathcal{I}'$ *subset-simulates* $\mathcal{I}$, or $\mathcal{I}$ *is subset-simulated* by $\mathcal{I}'$. We write $\mathcal{I} \leq_\subseteq \mathcal{I}'$ if $\mathcal{I}$ is subset-simulated by $\mathcal{I}'$.

Subset-simulation is a preorder on models. That is, subset-simulation is a reflexive and transitive relation on models. For this reason it can be used to define the following minimality criterion. A model $\mathcal{I}$ of a modal formula ϕ is *minimal modulo subset-simulation* iff for any model $\mathcal{I}'$ of ϕ, if $\mathcal{I}' \leq_\subseteq \mathcal{I}$, then $\mathcal{I} \leq_\subseteq \mathcal{I}'$.

To have a visual idea of the minimality criterion, Figure 1 shows two possible models of a modal formula. The subset-simulation relationship is represented by directed dashed lines. Given the definition of the minimality criterion, the model on the left is considered minimal because it is subset-simulated by the model on the right.

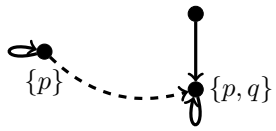

Figure 1: Example of minimality w.r.t. subset-simulation

As subset-simulation is not anti-symmetric, there can be models that subset-simulate each other, resulting in a symmetry class w.r.t. the ordering. As a result, many models minimal modulo subset-simulation can belong to the same symmetry class. To avoid the generation of all such models, and because they share a lot of positive information, we consider a procedure to be minimal model complete if it generates at least a minimal model for each symmetry class of minimal models.

3 Challanges and Possible Solutions

Due to lack of space, we do not explain the procedures in [7]. For this abstract it is enough to say that those procedures are based on tableaux calculi that are minimal model complete, minimal model soundness is achieved by using a minimality test called the subset-simulation test, and termination is guaranteed by the use of variations of ancestor equality blocking (an example of equality blocking can be found in [3]).

Three challenges need to be addressed to expand the procedures in [7]. First, some rules of the calculus need to be modified to the multi-modal case, and new rules need to be added. Second, minimal model completeness needs to be preserved. Finally, termination needs to be ensured. Minimal model soundness is not a challenge. This is because the subset-simulation test is logic independent enough to result in minimal model sound procedures for all logics we consider. This is true only if the procedures are minimal model complete.

Adopting the rules for the multi-modal case is easy. Also the introduction of rules for universal modalities and inclusion axioms does not pose particular problems. As an example, the following rule can be used for the universal modality $\langle \mathcal{U} \rangle$.

$$(\langle \mathcal{U} \rangle) \; \frac{u : \langle \mathcal{U} \rangle \phi}{v : \phi} \qquad \text{where } v \text{ is fresh.}$$

Regarding the inclusion axioms, it is enough to add a rule for each of them as in the following example. Suppose there is an inclusion axiom of the form $[R_1]\phi \rightarrow [R_2][R_3]\phi$. Then the following rule is added to the calculus.

$$\frac{(u, v) : R_2 \qquad (v, w) : R_3}{(u, w) : R_1}.$$

Once the rules of the calculus are established, minimal model completeness needs to be proved. This can be proved by showing that for each minimal model there exists a branch of the tableau from which an equivalent model can be extracted. The proof we have in mind is a variation of the minimal model completeness proof in [6], where a similar minimality criterion and a similar calculus are used.

The last challenge is to preserve termination. This is the most complex problem to be solved. Decision procedures for reasoning in the logics under consideration already exist. What makes the task of minimal model generation harder is that termination techniques have a clear impact on models. It might happen, if a wrong termination strategy is used, that minimal model soundness and completeness are lost. What is clear from our previous studies is that some termination techniques, such as subset blocking, cannot be used because they conflict with the minimality criterion. This led us to think that only strategies based on equality blocking can be used, but we are still investigating which technique can be used to achieve termination while preserving minimal model completeness. Finding terminating procedures for the logics under consideration is important for theoretical and practical reasons. First, it would imply that all the logics under consideration have a finite number of symmetry classes of minimal models. Second, termination is fundamental for an efficient and effective implementation of the procedures.

4 Conclusion

The procedures in [7] are an important contribution towards the generation of models minimal modulo subset-simulation for modal logics. In this abstract we focused on generalising such procedures to more expressive modal logics. We believe that it is possible to design minimal model sound and complete procedures for more expressive modal logics, and this can be done by modifying carefully the tableau calculus in [7].

The main remaining challenge is to ensure termination. We believe that some variation of equality blocking can help us to reach our goal, to prove that all the logics under consideration have a finite number of symmetry classes of minimal models, and to provide a basis for practical implementations of the procedures.

References

[1] Bry, F., Yahya, A.: Positive unit hyperresolution tableaux and their application to minimal model generation. J. Automated Reasoning 25(1), 35–82 (2000)

[2] Hintikka, J.: Model minimization - an alternative to circumscription. J. Automated Reasoning 4(1), 1–13 (1988)

[3] Horrocks, I., Sattler, U.: A description logic with transitive and inverse roles and role hierarchies. J. Logic Computation 9(3), 385–410 (1999)

[4] Nguyen, L.A.: Constructing finite least Kripke models for positive logic programs in serial regular grammar logics. Journal of the IGPL 16(2), 175–193 (2008)

[5] Papacchini, F., Schmidt, R.A.: A tableau calculus for minimal modal model generation. Electr. Notes Theoret. Computer Sci. 278(3), 159–172 (2011)

[6] Papacchini, F., Schmidt, R.A.: Computing minimal models modulo subset-simulation for propositional modal logics. In: Proc. FroCoS'13. LNAI, vol. 8152, pp. 279–294. Springer (2013)

[7] Papacchini, F., Schmidt, R.A.: Terminating Minimal Model Generation Procedures for Propositional Modal Logics. In: Proc. IJCAR'14. Springer (2014). To appear

[8] Reiter, R.: A theory of diagnosis from first principles. Artificial Intelligence 32(1), 57–95 (1987)

Tableau Development for a Bi-Intuitionistic Tense Logic*

John G. Stell[1] Renate A. Schmidt[2] David Rydeheard[2]

[1] School of Computing, University of Leeds, Leeds, UK
[2] School of Computer Science, University of Manchester, Manchester, UK

Abstract: Motivated by the theory of relations on graphs and applications to spatial reasoning, we present a bi-intuitionistic logic BISKT with tense operators. The logic is shown to be decidable and have the effective finite model property. We present a sound, complete and terminating tableau calculus for the logic and use the MetTeL system to obtain an implementation.

1 Introduction

In image processing the operations of mathematical morphology are used to derive new images from given ones. These new images approximate the input images in useful ways. The operations transform sets of pixels to sets of pixels in a way which depends on a parameter which is effectively a relation on the set of all pixels. Motivated by the aim of developing similar approximation operations for subgraphs of a graph, we have developed a theory of relations on graphs [5]. These are ordinary relations on the set of all edges and nodes of the graph which satisfy a stability condition. A generalisation of these relations is used here to provide accessibility relations for a modal logic. Semantically, formulae can represent subgraphs of a graph, or more generally, downward closed sets in a pre-order. A particular novel feature of the semantics is the use of a weaker form of the converse operation on relations. This has resulted in a bi-intuitionistic tense logic, called BISKT, in which the four modalities are not mutually independent. In exploring the properties of this logic we have made essential use of tableau systems generated using the MetTeL software [1, 8].

2 Bi-intuitionistic stable tense logic

BISKT is a modal bi-intuitionistic logic with four modalities $\Box$, $\blacklozenge$, $\Diamond$ and $\blacksquare$. The remaining connectives are $\bot$, $\neg$ (intuitionistic negation), $\lrcorner$ (dual negation), $\wedge$, $\vee$, $\rightarrow$ (intuitionistic implication) and $\succ$ (dual implication).

Kripke frames for BISKT consist of a pre-order H interacting with an accessibility relation R via a stability condition. A binary relation R on a set U is *stable*, if $H \,;\, R \,;\, H \subseteq R$, where ; denotes relational composition. The semantics interprets formulae as H-sets, i.e., the downwardly closed sets of the pre-order, and the bi-intuitionistic connectives are interpreted as usual. This means, for example, $\neg$ is interpreted as pseudo-complement with respect to H, $\rightarrow$ as relative pseudo-complement (i.e., intuitionistic implication) with respect to H, etc. $\Box$ and $\blacklozenge$ are interpreted respectively as the standard box modality and the standard backward looking diamond over R. The pair $\blacksquare$ and $\Diamond$ are interpreted similarly as the standard box modality and the standard backward looking diamond, but this time over the left converse of R. The *left converse* of a stable relation R is $\smile R = H \,;\, \breve{R} \,;\, H$, where $\breve{R}$ is the (ordinary) converse of R.

Unlike in (bi-)intuitionistic logics and other (bi-)intuitionistic modal logics, where all connectives are independent from each other, in BISKT the white $\Box$ and white $\Diamond$ are related as follows:

$$\Diamond\varphi \equiv \lrcorner\Box\neg\varphi.$$

We have shown:

Theorem 1 BISKT *is decidable, it has the effective finite model property and the computational complexity is PSPACE-complete.*

The proof is by showing that BISKT can be embedded into a modal logic, called $Kt(H, R)$, which itself can be embedded into the guarded fragment, which is known to be decidable and has the effective finite model property [2]. $Kt(H, R)$ is a traditional modal logic with forward and backward looking modal operators defined by H and R as accessibility relations. The frame conditions are reflexivity and transitivity of H, and stability of R with respect to H. As both embeddings have linear complexity and define in fact an effective translation into a subfragment of the guarded fragment, which is PSPACE-complete, the result follows.

*This research was supported by UK EPSRC research grant EP/H043748/1.

3 Tableau development

Since the accessibility relations in the Kripke models of BISKT involve converse relations it is natural to use a semantic tableau method. In particular, we use a labelled signed tableau approach based on an explicit tableau system because this ensures proof confluence. This means there is no need for backtracking over the proof search, and there is more flexibility in defining search heuristics in an implementation. These are aspects that make it harder to develop implicit tableau systems where the rules operate on formulae of the logic and do not include any meta-logical entities referring to semantics. An additional advantage of semantic tableau calculi is that they return concrete models for satisfiable formulae.

Semantic tableau deduction calculi are easy to develop. We follow the methodology of tableau synthesis and refinement as introduced in [3, 7] to develop a tableau calculus for BISKT. Tableau synthesis amounts to a normalisation process of the definition of the semantics of the logic. In the case of BISKT atomic rule refinement was sufficient to obtain a set of tableau rules with reduced branching. Soundness and completeness of the obtained calculus follows from the results of the general framework in [3, 7]. To guarantee termination we use the unrestricted blocking technique [3, 4]. Compared to other blocking techniques it imposes no restrictions and is generic, so independent of any logic and can even be used for undecidable logics. Unrestricted blocking has the property that adding it to a sound and complete semantic tableau calculus guarantees termination, for any logic that has the finite model property. Since we proved that BISKT has the finite model property the tableau calculus is thus terminating.

Implementing a prover requires lots of specialist knowledge and there are various non-trivial obstacles to overcome, but using the MetTeL tableau prover generator requires just feeding in the rules of the calculus into the tool which then fully automatically generates an implementation in Java [1, 8]. Our tableau calculus is in exactly the form as accepted by MetTeL and unrestricted blocking is supported by MetTeL. We have therefore implemented the calculus using MetTeL. MetTeL turned out to be useful to experiment with several initial versions of the calculus. In combination with the tableau synthesis method it was easy to run tests on a growing collection of problems with different provers for several preliminary versions of formalisations of bi-intuitionistic tense logics before settling on a definition. MetTeL has also allowed us to experiment with different refinements of the rules and different variations of blocking.

4 Concluding remarks

Further details and a detailed description of the tableau calculus are presented in the long version [6] of this abstract. The MetTeL specification of the tableau calculus and the generated prover for BISKT can be downloaded from `http://staff.cs.manchester.ac.uk/~schmidt/publications/biskt13/`, where also performance graphs and the problems used can be found.

References

[1] MetTeL website. `http://www.mettel-prover.org`.

[2] R. A. Schmidt, J. G. Stell, and D. Rydeheard. Axiomatic and tableau-based reasoning for Kt(H,R). In *Advances in Modal Logic, Volume 10*, London, 2014. College Publ. To appear.

[3] R. A. Schmidt and D. Tishkovsky. Automated synthesis of tableau calculi. *Logical Methods in Computer Science*, 7(2):1–32, 2011.

[4] R. A. Schmidt and D. Tishkovsky. Using tableau to decide description logics with full role negation and identity. *ACM Transactions on Computational Logic*, 15(1), 2014.

[5] J. G. Stell. Relations on hypergraphs. In *Proc. RAMiCS 2012*, volume 7560 of *LNCS*, pages 326–341. Springer, 2012.

[6] J. G. Stell, R. A. Schmidt, and D. Rydeheard. Tableau development for a bi-intuitionistic tense logic. In *Proc. RAMiCS 14*, volume 8428 of *LNCS*, pages 412–428. Springer, 2014.

[7] D. Tishkovsky and R. A. Schmidt. Refinement in the tableau synthesis framework. arXiv e-Print 1305.3131v1, 2013.

[8] D. Tishkovsky, R. A. Schmidt, and M. Khodadadi. The tableau prover generator MetTeL2. In *Proc. JELIA 2012*, volume 7519 of *LNCS*, pages 492–495. Springer, 2012.

Socratic Proofs for Propositional Linear-Time Logic

Mariusz Urbanski[1] Alexander Bolotov[2] Vasilyi Shangin[3] Oleg Grigoriev[3]

[1] Adam Mickiewicz University, Poznań, Poland
Mariusz.Urbanski@amu.edu.pl
[2] University of Westminster, UK
A.Bolotov@wmin.ac.uk
[3] Moscow State University, Russia
shangin@philos.msu.ru, grigoriev@philos.msu.ru

Abstract:

This paper presents a calculus of Socratic proofs for Propositional Linear-Time Logic (PLTL) and discusses potential automation of its proof search.

1 Introduction

Propositional Linear-Time Logic (PLTL) [6] gained various deductive constructions: axiomatic [5], tableau [11], resolution [4], and natural deduction [1]. In this paper we present a calculus of Socratic proofs for Propositional Linear-Time Logic (PLTL) [6], [3], [7] abbreviated as $\mathsf{PLTL_{SP}}$. The calculus is based upon the hypersequent calculus [2] and it fits into the framework of Socratic proofs by Wisniewski (cf. [8], [10] and [9]).

2 Logic PLTL$_T$

We utilise the language of PLTL which extends the language of Classical Propositional Calculus (CPC) by temporal operators: $\mathcal{U}$ (*until*) $\bigcirc$ (*at the next moment in time*), $\Box$ (*always in the future*), and $\Diamond$ (*at sometime in the future* or *eventually*). The semantics for the temporal part of the logic PLTL$_T$ is defined in the standard way over linear sequence of states, finite in the past, infinite in the future.

In order to formulate PLTL$_T$ we need to extend the language of PLTL with the following signs: $\vdash$, ?, 1 and 2. Intuitively, $\vdash$ stands for derivability relation and ? is a question-forming operator. The numerals 1 and 2 will be used to encode tree-structure of a Socratic transformation.

There are two disjoint categories of wffs: *declarative* wffs (d-wffs) and *erotetic* wffs (e-wffs), or questions. There are also two types of d-wffs: *atomic* d-wffs and *indexed d-wffs*. Atomic d-wffs are expressions of the form $S \vdash A$, where S is a finite sequence (possibly with repetitions) of PLTL-wffs, and A is a PLTL-wff, and if A is an empty formula, then S is a non-empty sequence. Indexed d-wffs are expressions of the form $S \vdash^n A$ or of the form $T \vdash^n$, where $S \vdash A$ and $T \vdash$ are atomic d-wffs of and n is a sequence of 1's or 2's, starting with 1. E-wffs, or questions are expressions of the form $?(\Phi)$, where Φ is a non-empty finite sequence of indexed atomic d-wffs (*constituents* of Φ).

In the formulation of rules we shall use the following classification of PLTL formulae to α and β types:

α	α_1	α_2
$A \wedge B$	A	B
$\neg(A \vee B)$	$\neg A$	$\neg B$
$\neg(A \rightarrow B)$	A	$\neg B$
$\Box A$	A	$\bigcirc\Box A$
$\neg\Diamond A$	$\neg A$	$\bigcirc\Box\neg A$
$\neg(A\mathcal{U}B)$	$\neg B$	$\neg(A \wedge \bigcirc(A\mathcal{U}B))$

β	β_1	β_2	β_1^*
$\neg(A \wedge B)$	$\neg A$	$\neg B$	A
$A \vee B$	A	B	$\neg A$
$A \rightarrow B$	$\neg A$	B	A
$\neg\Box A$	$\neg A$	$\bigcirc\Diamond\neg A$	A
$\Diamond A$	A	$\bigcirc\Diamond A$	$\neg A$
$A\mathcal{U}B$	B	$A \wedge \bigcirc(A\mathcal{U}B)$	$\neg B$

Rules for PLTL$_T$:

$$\mathbf{L}_\alpha: \quad \frac{?\,(\Phi; S\,'\,\alpha\,'\,T \vdash^n C; \Psi)}{?\,(\Phi; S\,'\,\alpha_1\,'\,\alpha_2\,'\,T \vdash^n C; \Psi)}$$

$$\mathbf{L}_\beta: \quad \frac{?\,(\Phi; S\,'\,\beta\,'\,T \vdash C; \Psi)}{?\,(\Phi; S\,'\,\beta_1\,'\,T \vdash^{n1} C; S\,'\,\beta_2\,'\,T \vdash^{n2} C; \Psi)}$$

$$\mathbf{L}_{\neg\neg}: \quad \frac{?\,(\Phi; S\,'\,\neg\neg A\,'\,T \vdash^n C; \Psi)}{?\,(\Phi; S\,'\,A\,'\,T \vdash^n C; \Psi)}$$

$$\mathbf{L}_{\neg\bigcirc}: \quad \frac{?\,(\Phi; S\,'\,\neg\bigcirc A\,'\,T \vdash^n C; \Psi)}{?\,(\Phi; S\,'\,\bigcirc\neg A\,'\,T \vdash^n C; \Psi)}$$

$$\mathbf{R}_\alpha: \quad \frac{?\,(\Phi; S \vdash^n \alpha; \Psi)}{?\,(\Phi; S \vdash^{n1} \alpha_1; S \vdash^{n2} \alpha_2; \Psi)}$$

$$\mathbf{R}_\beta: \quad \frac{?\,(\Phi; S \vdash^n \beta; \Psi)}{?\,(\Phi; S\,'\,\beta_1^* \vdash^n \beta_2; \Psi)}$$

$$\mathbf{R}_{\neg\neg}: \quad \frac{?\,(\Phi; S \vdash^n \neg\neg A; \Psi)}{?\,(\Phi; S \vdash^n A; \Psi)}$$

$$\mathbf{R}_{\neg\bigcirc}: \quad \frac{?\,(\Phi; S \vdash^n \neg\bigcirc A; \Psi)}{?\,(\Phi; S \vdash^n \bigcirc\neg A; \Psi)}$$

If none of the above rules is applicable to a PLTL formula B, such a formula is called *marked*. If all PLTL-formulas within an indexed formula $S \vdash^n A$ are marked, such a formula is called *a state*.

The following is a state-prestate rule:

$$\mathbf{S-P}: \quad \frac{?\,(\Phi; S \vdash^n A; \Psi)}{?\,(\Phi; S^\circ \vdash^n A^\circ; \Psi)}$$

where $S \vdash^n A$ is a state and S° (resp. A°) results from S (resp. A) by replacing all the formulas of the form $\bigcirc B$ with B and deleting all the remaining formulas. Every formula

of the form $S^* \vdash^m A^*$, where n is an initial subsequence of m or m is an initial subsequence of n, is called *a pre-state* (cf. [11]).

Definition 1. *Let $\mathbf{q} = \langle Q_1, \ldots, Q_r \rangle$ be a finite sequence of questions of $\mathbf{P}^*$. Let Q_g, Q_{h-1}, Q_h $(1 \leq g < h-1 \leq r)$ be elements of the sequence $\mathbf{q}$. Let $S_j \vdash^n A_j$ be a constituent of Q_g and let $S_k \vdash^m A_k$ be a constituent of Q_h such that $S_j = S_k$, $A_j = A_k$ and the sequence n is an initial subsequence of the sequence m. Let $S_l \vdash^i A_l$ be a constituent of Q_{h-1} such that $S_k \vdash^m A_k$ is obtained from $S_l \vdash^i A_l$ by application of a PT*-rule. Then $S_j \vdash^n A_j, \ldots, S_l \vdash^i A_l$ form a* loop *(a sequence of atomic d-wffs of $\mathbf{P}^*$ ... etc.), and $S_k \vdash^m A_k$ is called a* loop-generating formula*.*

Socratic transformations are sequences of questions that aim at deciding derivability of formuls from sets of formuls.

Definition 2. *A finite sequence $\langle Q_1, \ldots, Q_r \rangle$ of questions of $\mathbf{P}^*$ is a* Socratic transformation *of $S \vdash A$ iff the following conditions hold: (i)$Q_1 = ?(S \vdash^1 A)$; (ii)Q_i (where $i = 2, \ldots, r$) results from Q_{i-1} by applying a PT*-rule.*

Definition 3. *A constituent ϕ of a question Q_i is called* successful *iff one of the following holds: (a) ϕ is of the form $T'B'U \vdash^n B$, or (b) ϕ is of the form $T'B'U'\neg B'W \vdash^n C$, or (c) ϕ is of the form $T'\neg B'U'B'W \vdash^n C$.*

Definition 4. *A Socratic transformation $\langle Q_1, \ldots, Q_r \rangle$ of $S \vdash A$ is* completed *iff the for each constituent ϕ of Q_r at least one of the following conditions hold: (a) no rule is applicable to PLTL-formulas in ϕ, or (b) ϕ is successful, or (c) ϕ is a loop-generating formula.*

Definition 5. *A formula B is called an* eventuality *in $S \vdash^n A$ iff one of the following holds: (i) B is a term of S and there exists a PLTL-formula C such that $B = \Diamond C$, or (ii) there exists a PLTL-formula C such that $B = A = \Box C$.*

Definition 6. *A completed Socratic transformation $\mathbf{q} = \langle Q_1, \ldots, Q_r \rangle$ is a Socratic proof of $S \vdash A$ iff: (a) all the constituents of Q_n are successful, or (b) for each non-successful constituent ϕ of Q_n, ϕ is a loop-generating formula and the loop generated by ϕ contains a pre-state with an unfulfilled eventuality.*

The presented system is sound and complete. Proofs of these theorems involve construction of a canonical model with maximal consistent sets of formulae as its states.

3 Examples

In the examples below by highlighting we indicate a formula which is analyzed at the current step. Double underlining of a formula reflects that it is a state. The question following the one containing a state is obtained by state-prestate rule.

Example 1

1. $?(\vdash^1 \Box p \rightarrow p)$
2. $?(\Box p \vdash^1 p)$
3. $?(p, \bigcirc\Box p \vdash^1 p)$

Example 2

1. $?(\vdash^1 \Box p \rightarrow \bigcirc p)$
2. $?(\Box p \vdash^1 \bigcirc p)$
3. $?(\underline{\underline{p, \bigcirc\Box p \vdash^1 \bigcirc p}})$
4. $?(\underline{\underline{\Box p}} \vdash^1 p)$
5. $?(p, \bigcirc\Box p \vdash^1 p)$

References

[1] O. Grigoryev A. Basukoski, A. Bolotov and V. Shangin. Natural deduction system for linear time temporal logic . *Logical investigations*, (13):71–95, 2004.

[2] A. Avron. *The Method of Hypersequents in the Proof Theory of Propositional Non-Classical Logics*, pages 1–32. Logic: Foundations to Applications. Clarendon Press, 1996.

[3] M. Finger, D. M. Gabbay, and M. Reynolds. *Advanced Tense Logic*, volume 7 of *Handbook of Philosophical Logic*, pages 43–203. Springer, 2002.

[4] Michael Fisher, Clare Dixon, and Martin Peim. Clausal temporal resolution. *ACM Trans. Comput. Log.*, 2(1):12–56, 2001.

[5] D. M. Gabbay, A. Pnueli, S. Shelah, and J. Stavi. On the temporal analysis of fairness. In *7th ACM Symposium on Principles of Programming Languages*, pages 163–173, 1980.

[6] A. Pnueli. The temporal logic of programs. In *Proceedings of the 18th Symposium on Foundations of Computer Science*, pages 46–57, 1977.

[7] A. P. Sistla and E. M. Clarke. The complexity of Propositional Linear Temporal Logic. *Journal of the Association for Computing Machinery*, 32(3):733–749, 1985.

[8] A. Wiśniewski. Socratic Proofs. *Journal of Philosophical Logic*, 33(2):299–326, 2004.

[9] A. Wiśniewski and V. Shangin. Socratic proofs for quantifiers. *Journal of Philosophical Logic*, 35(2):147–178, 2006.

[10] A. Wiśniewski, G. Vanackere, and D. Leszczyńska. Socratic Proofs and Paraconsistency. A Case Study. *Studia Logica*, 80:431–466, 2005.

[11] P. Wolper. The tableau method for temporal logic: An overview. *Logique et Analyse*, 28:119–136, 1985.

Second-Order Characterizations of Definientia in Formula Classes

Christoph Wernhard

Technische Universität Dresden

Abstract: Predicate quantification can be applied to characterize definientia of a given formula that are in terms of a given set of predicates. Methods for second-order quantifier elimination and the closely related computation of forgetting, projection and uniform interpolants can then be applied to compute such definientia. Here we address the question, whether this principle can be transferred to definientia in given classes that allow efficient processing, such as Horn or Krom formulas.

Tasks in knowledge processing such as view-based query rewriting with exact rewritings [1, 14, 16, 5, 21] involve the *computation of a definiens* R of a given "query" formula Q within a second given "background" formula F, such that R meets certain conditions, for example, that it is expressed in terms of a given set S of predicates. That is, for given Q, F and S, a formula R must be computed, such that $F \models (R \leftrightarrow Q)$ and only predicates from S do occur in R. If the requested property of R is indeed that it involves only predicates from a restricted set, then the class of solution formulas R can be characterized straightforwardly with predicate quantification. This allows to relate the computation of formulas R to the various advances concerning applications of and methods for second-order quantifier elimination and its variants, the computation of forgetting, of projection, and of uniform interpolants, in particular with respect to knowledge representation in first-order logic and description logics [7, 6, 9, 12, 18, 2, 19, 10], and in the preprocessing of propositional formulas [4, 8, 13]. The underlying basic principle is that the second-order formula $\exists P\, F$, where F is a first-order formula and P is a predicate, is – if it admits elimination of the predicate quantifier – equivalent to a first-order formula that does not involve P but is equivalent to F with respect to the other predicates.

The question addressed here is whether also definientia in given classes of formulas that are typically *characterized by other means than vocabulary restrictions*, such as Horn formulas, conjunctions of atoms or Krom formulas, can be specified with predicate quantification and can thus be computed by second-order quantifier elimination methods. The envisaged main application is to compute such definientia as query rewritings that are in restricted classes which allow further processing in particularly efficient ways or by engines with limited deductive capability. It seems that also the requirement that the *negation* of a definiens R is in some given formula class can be useful: R might be evaluated by proving that a given knowledge base is unsatisfiable when conjoined with $\neg R$. The case of negated definientia is subsumed by the general case: The requirement that R is a definiens for Q, where $\neg R$ is in some formula class, can be expressed just as the requirement that R' is a definiens in some formula class for $\neg Q$ and letting R be $\neg R'$.

A starting point for the second-order characterizations of the considered formula classes is the semantic characterization of the class of propositional Horn formulas [3, 15]: A propositional formula is equivalent to a Horn formula if and only if it has the *model intersection property*. Literal projection [11, 17] is a generalization of predicate quantification that allows to specify that only positive or negative predicate occurrences are affected. It can be combined with the semantic characterization of Horn formulas to express the requirement that the definiens is a conjunction of atoms. This restriction can be – up to equivalence – characterized purely by second-order operators that can be defined in terms of predicate quantification. It can be applied to express restrictions to further classes as vocabulary restrictions by meta-level encodings. We show this for Krom formulas, which can be represented as conjunctions of "meta-level" atoms of the form $\mathsf{clause}(L, M)$, defined in the background formula with equivalences $(\mathsf{clause}(L, M) \leftrightarrow L \vee M)$ for literals L, M of the original vocabulary.

As elaborated in [22], such characterizations of definienta and definability, that is, the existence of definientia, can be developed in a framework on the basis of three second-order operators: For literal projection, for "raising" [18], which can be applied to express generalizations of circumscription, including model maximization, and for "scoped difference" [22], which allows, for example, to specify a formula whose models are exactly all the lower bounds of the set of models of a given formula (interpretations are compared there w.r.t. the subset relationship between the sets of the ground atoms that they satisfy). The characterizations of definientia and definability are then expressed by further second-order operators, which can be defined like macros in terms of these three basic ones. As also shown in [22], the three basic operators themselves can all be encoded as predicate quantification, such that characterizations of definability and definientia in terms of these operators can be translated to characterizations with predicate quantification as the only second-order operator.

An inherent feature or limitation of the presented approach is that it applies only to formula classes that are *closed under equivalence*. Nevertheless, with respect to vocabulary restrictions, elimination methods usually produce outputs that do no longer contain the quantified predicates, thereby ensuring that results are also in the corresponding syntactic classes. It needs to be investigated, in which way elimination methods applied to the suggested second-order expressions for the considered formula classes yield results that are actually also in the corresponding syntactic classes.

Although the envisaged underlying logic is classical first-order logic, the material in has been technically developed in [22] essentially for propositional logic, for simplicity of presentation and because the considered formula classes there have immediate correspondence to syntactic classes such as Horn and Krom formulas as well as conjunctions of atoms. Actually, the material transfers to a large extent easily to first-order logic, following the principles in [17, 18, 21]. Some minor particularities are indicated in [22]. What still need to be examined for the first-order case is the correspondence of the semantic characterizations of formula classes to expressibility in syntactic classes.

Operators and properties are formally defined in terms of each other in a way that fits mechanization. In fact, they have been defined similarly on top of the *ToyElim* system [20].[1] This is currently only suitable for small experiments and an advanced implementation of the suggested operators seems to be a major challenge on its own. At least in principle, the presented characterizations of definientia should be expressible also on top of other systems for second-order quantifier elimination and its variants, the computation of forgetting, projection and uniform interpolants.

References

[1] Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Moshe Y. Vardi. View-based query processing: On the relationship between rewriting, answering and losslessness. *TCS*, 371(3):169–182, 2007.

[2] B. Cuenca Grau and B. Motik. Reasoning over ontologies with hidden content: The import-by-query approach. *JAIR*, 45, 2012.

[3] Rina Dechter and Judea Pearl. Structure identification in relational data. *AI*, 58:237–270, 1992.

[4] Niklas Eén and Armin Biere. Effective preprocessing in SAT through variable and clause elimination. In *SAT 2005*, volume 3569 of *LNCS*, pages 61–75, 2005.

[5] Enrico Franconi, Volha Kerhet, and Nhung Ngo. Exact query reformulation over databases with first-order and description logics ontologies. *JAIR*, 48:885–922, 2013.

[6] Dov M. Gabbay, Renate A., Schmidt, and Andrzej Szałas. *Second-Order Quantifier Elimination: Foundations, Computational Aspects and Applications*. College Publications, London, 2008.

[7] Silvio Ghilardi, Carsten Lutz, and Frank Wolter. Did I damage my ontology? A case for conservative extensions in description logics. In *KR 2006*, pages 187–197. AAAI Press, 2006.

[8] Marijn Heule, Matti Järvisalo, and Armin Biere. Clause elimination procedures for CNF formulas. In *LPAR-17*, volume 6397 of *LNCS*, pages 357–371. Springer, 2010.

[9] Boris Konev, Dirk Walther, and Frank Wolter. Forgetting and uniform interpolation in large-scale description logic terminologies. In *IJCAI-09*, pages 830–835. AAAI Press, 2009.

[10] Patrick Koopmann and Renate A. Schmidt. Uniform interpolation of $\mathcal{ALC}$-ontologies using fixpoints. In *FroCoS 2013*, volume 8152 of *LNCS (LNAI)*, pages 87–102. Springer, 2013.

[11] Jérôme Lang, Paolo Liberatore, and Pierre Marquis. Propositional independence – formula-variable independence and forgetting. *JAIR*, 18:391–443, 2003.

[12] Carsten Lutz and Frank Wolter. Foundations for uniform interpolation and forgetting in expressive description logics. In *IJCAI-11*, pages 989–995. AAAI Press, 2011.

[13] Norbert Manthey, Tobias Philipp, and Christoph Wernhard. Soundness of inprocessing in clause sharing SAT solvers. In *SAT 2013*, volume 7962 of *LNCS*, pages 22–39. Springer, 2013.

[14] Maarten Marx. Queries determined by views: pack your views. In *PODS '07*, pages 23–30. ACM, 2007.

[15] J. C. C. McKinsey. The decision problem for some classes of sentences without quantifiers. *JSL*, 8:61–76, 1943.

[16] Alan Nash, Luc Segoufin, and Victor Vianu. Views and queries: Determinacy and rewriting. *TODS*, 35(3), 2010.

[17] Christoph Wernhard. Literal projection for first-order logic. In *JELIA 08*, volume 5293 of *LNCS (LNAI)*, pages 389–402. Springer, 2008.

[18] Christoph Wernhard. Projection and scope-determined circumscription. *JSC*, 47(9):1089–1108, 2012.

[19] Christoph Wernhard. Abduction in logic programming as second-order quantifier elimination. In *FroCoS 2013*, volume 8152 of *LNCS (LNAI)*, pages 103–119. Springer, 2013.

[20] Christoph Wernhard. Computing with logic as operator elimination: The ToyElim system. In *INAP 2011/WLP 2011*, volume 7773 of *LNCS (LNAI)*. Springer, 2013.

[21] Christoph Wernhard. Expressing view-based query processing and related approaches with second-order operators. Technical Report KRR 14–02, TU Dresden, 2014. `http://www.wv.inf.tu-dresden.de/Publications/2014/report-2014-02.pdf`.

[22] Christoph Wernhard. Second-order characterizations of definientia in formula classes. Technical Report KRR 14–03, TU Dresden, 2014. `http://www.wv.inf.tu-dresden.de/Publications/2014/report-2014-03.pdf`.

[1] See `http://cs.christophwernhard.com/toyelim/`.

The Leo-III Project

Max Wisniewski[1] Alexander Steen[2] Christoph Benzmüller[3]

[1] FU-Berlin, Arnimallee 7, max.wisniewski@fu-berlin.de
[2] FU-Berlin, Arnimallee 7, a.steen@fu-berlin.de
[3] FU-Berlin, Arnimallee 7, c.benzmueller@fu-berlin.de

Abstract: We introduce the recently started Leo-III project — a Higher-Order Logic Theorem Prover and successor to LEO-II.

1 Summary

We report on the recently started Leo-III project, in which we design and implement a state-of-the-art Higher-Order Logic Theorem Prover, the successor of the well known LEO-II prover [2]. Leo-III will be based on ordered paramodulation/superposition.

In constrast to LEO-II, we replace the internal term representation (the commonly used simply typed lambda calculus) by a more expressive system supporting type polymorphism. In the course of the project, we plan to further enhance the type system with type classes and type constructors similar to System F^{ω}.

In order to achieve a substantial performance speed-up, the architecture of Leo-III will be based on massive parallelism (e.g. And/Or-Parallelism, Multisearch) [3]. The current design is a multi-agent blackboard architecture [10] that will allow to independently run agents with our proof calculus as well as agents for external (specialized) provers. Leo-III will focus right from the start on compatibility to the widely used TPTP infrastructure [8]. Moreover, it will offer built-in support for specialized external prover agents and provide external interfaces to interactive provers such as Isabelle/HOL [5]. The implementation will excessively use term sharing [6, 7] and several indexing techniques [4, 9]. Leo-III will also offer special support for reasoning in various quantified non-classical logics by exploiting a semantic embedding [1] approach.

References

[1] Christoph Benzmüller. A top-down approach to combining logics. In *Proc. of the 5th International Conference on Agents and Artificial Intelligence (ICAART)*, Barcelona, Spain, 2013. SciTePress Digital Library.

[2] Christoph Benzmller, Lawrence C. Paulson, and Frank Theiss. Leo-ii a cooperative automatic theorem prover for higher-order logic. In *In Fourth International Joint Conference on Automated Reasoning (IJCAR08), volume 5195 of LNAI*. Springer, 2008.

[3] Maria Paola Bonacina. A taxonomy of parallel strategies for deduction. *Ann. Math. Artif. Intell.*, 29(1-4):223–257, 2000.

[4] Robert Nieuwenhuis, Thomas Hillenbrand, Alexandre Riazanov, and Andrei Voronkov. On the evaluation of indexing techniques for theorem proving, 2003.

[5] T. Nipkow, L.C. Paulson, and M. Wenzel. *Isabelle/HOL: A Proof Assistant for Higher-Order Logic*. Lecture Notes in Computer Science. Springer, 2002.

[6] Alexandre Riazanov and Andrei Voronkov. The design and implementation of vampire. *AI Commun.*, 15(2,3):91–110, August 2002.

[7] Stephan Schulz. E - a brainiac theorem prover. *AI Commun.*, 15(2,3):111–126, August 2002.

[8] Geoff Sutcliffe. The tptp problem library and associated infrastructure. *J. Autom. Reason.*, 43(4):337–362, December 2009.

[9] Frank Theiss and Christoph Benzmüller. Term indexing for the LEO-II prover. In *IWIL-6 workshop at LPAR 2006: The 6th International Workshop on the Implementation of Logics*, Pnom Penh, Cambodia, 2006.

[10] Gerhard Weiss, editor. *Multiagent Systems*. MIT Press, Cambridge, MA, 2013.