Seventh Workshop on Automated Reasoning

Bridging the Gap between Theory and Practice

London, 20-21 July 2000

Hosted by the
Group of Logic and Computation
Department of Computer Science
King's College London

KING'S College LONDON *Founded* 1829

The ARW 2000 web site is `http://www.dcs.kcl.ac.uk/events/ARW/` and the web site for the workshop series is `http://www-users.cs.york.ac.uk/~frisch/AutoReason/`.

Foreword

Automated Reasoning got its initial boost after Alan Robinson invented the resolution principle in 1963. For many years after this historical event "automated reasoning" was therefore a synonym for "resolution based reasoning for predicate logic". But this has been almost 40 years ago. "Automated reasoning" in the early 21st century comprises a still growing number of research areas. Some of them are: theorem proving in classical and non-classical logics, equational reasoning, unification, induction, logic programming, functional programming, constraint solving, formal methods for specifying, transforming and verifying systems, in particular safety-critical systems, non-monotonic reasoning, abduction, logic-based knowledge representation, in particular description logics.

The United Kingdom has an established research tradition in many of these areas and there are strong and internationally recognized research groups working on these problems. These groups meet once a year at the "Workshop on Automated Reasoning" to exchange results and ideas. The workshop is a rather informal event with short presentations and poster sessions. It provides the opportunity to keep the community informed about research activities and to present and discuss brand new ideas.

This year the workshop took place at King's College in London. There were 24 contributions, not only from the UK, but from countries as far away as Brazil and Lithuania. The 2-page abstracts of the contributions, together with the abstracts of the invited speakers are contained in these workshop notes.

I would like to thank very much everybody who contributed to the workshop. It has been sponsored by the Max-Planck Institute for Computer Science in Saarbrücken. We are grateful for their support.

Hans Jürgen Ohlbach

Organisers

Organising Committee

Robin D. Arthan (Lemma 1)
`rda@lemma-one.com`

Alan Bundy (University of Edinburgh)
`a.bundy@edinburgh.ac.uk`

Tony Cohn (University of Leeds)
`agc@scs.leeds.ac.uk`

Michael Fisher (Manchester Metropolitan University)
`m.fisher@doc.mmu.ac.uk`

Alan M. Frisch (University of York)
Chair of the Organising Committe
`frisch@cs.york.ac.uk`

Ian P. Gent (University of St. Andrews)
`ipg@dcs.st-and.ac.uk`

Andrew Ireland (Heriot-Watt University)
`air@cee.hw.ac.uk`

Manfred Kerber (University of Birmingham)
`m.kerber@cs.bham.ac.uk`

Hans Jürgen Ohlbach (LMU Munich)
ARW 2000 Programme Chair
`ohlbach@pms.informatik.uni-muenchen.de`

Andrei Voronkov (University of Manchester)
`voronkov@cs.man.ac.uk`

Toby Walsh (University of York)
`tw@cs.york.ac.uk`

Local Arrangements

Ulrich Endriss (King's College London)
`endriss@dcs.kcl.ac.uk`

Odinaldo Rodrigues (King's College London)
`odinaldo@dcs.kcl.ac.uk`

Stefan Schlobach (King's College London)
`schlobac@dcs.kcl.ac.uk`

Table of Contents

Invited Talks

Exclusion of Symmetries in Search – A Spin-off from Bioinformatics Research

Rolf Backofen

Ludwig-Maximilians-Universität München, Institut für Informatik,
Oettingenstr. 67, D-80538 München (Germany),
Email: backofen@informatik.uni-muenchen.de

Exclusion of symmetry is a long-standing problem in automated deduction as well as in constraint-based search. Several approaches have been proposed, which usually try to restrict the problem in a way that only one solution of the equivalence class (with respect to the symmetries) is found. To our knowledge, in all previously proposed approaches, this element is fixed in advance and does not depend on the search strategy. This has the drawback that the search strategy might interfere with the symmetry exclusion.

We considered the problem of lattice protein folding (a mathematical simplification of the real protein folding problem, albeit still NP-hard), where we were faced with a different situation. Here, the symmetries are well-known (geometric symmetries: translations, rotations and reflections), but it was not known how to exclude these symmetries efficiently. The standard approaches could not be applied since they restricted the search strategy.

Therefore, we introduced a new method for dynamically excluding symmetries in during search. The method is not restricted to a specific problem but can be applied to arbitrary symmetries (where our emphasis is not to detect automatically symmetries but to excluded known symmetries). Our method is based on the notion of symmetric constraints, which are used in our modification of a general constraint based search algorithm. The method does not influence the search strategy. Furthermore, it can be used with either the full set of symmetries, or with an subset of all symmetries.

We will show how to apply the method in the special case of geometric symmetries (rotations and reflections) and permutation symmetries. Furthermore, we give results from practical applications, and compare our system with systems proposed in the literature.

Modal Experiments

Maarten de Rijke

University of Amsterdam, Department of Mathematics, Computer Science, Physics and Astronomy, Plantage Muidergracht 24, 1018 TV Amsterdam (The Netherlands),
Email: `mdr@wins.uva.nl`

Recent years have witnessed the development of sophisticated automated reasoning methods for modal logic, both direct ones (usually based on tableau calculi) and indirect ones (based on translations into first-order logic or monadic second-order logic).

In this talk I will discuss a number of questions to which the development of these tools have given rise (such as heuristics, refinements, and evaluation techniques), as well as some recent answers.

Goal Directed Mechanisms: Proofs, Interpolation and Abduction Procedures

Dov Gabbay

King's College London, Department of Computer Science, Strand, London WC2R 2LS (UK), Email: `dg@dcs.kcl.ac.uk`

A goal directed proof mechanism will be presented for some substructural logics. It will be used to show how to get theorems, interpolants and abduced hypotheses.

Using Deduction Techniques for Natural Language Understanding

Michael Kohlhase

Universität des Saarlandes, Fachbereich Informatik,
Im Stadtwald, D-66041 Saarbrücken (Germany),
Email: kohlhase@cs.uni-sb.de

The talk emphasises the opportunity of using deduction methods in natural language understanding.

We start out by explaining some some of the usages of reasoning, such as common ground maintenance, discourse structure and coherence and semantic disambiguation using world knowledge.

The technical part of the talk takes a closer look at the application of first-order automated theorem proving- and model-generation techniques in all of these processes and investigates the necessary integration and communication between deduction systems and the linguistic modules.

In particular, I will show how the model-generation paradigm can be extended with mechanisms for anaphora, salience, resources to arrive at a better, cognitively and computationally more adequate account of the natural-language understanding process.

Contributed Abstracts

Computer Algebra and Automated Reasoning

A. A. Adams

1 Computer Algebra and Definite Integration

My current work is in the area of using Automated Theorem Proving (ATP) to support Computer Algebra Systems (CAS). CAS, as the name suggests, are systems designed originally for performing algebraic calculations on computer. They generally began life as a collection of specific algorithms for algebra, with a common specification language. Gradually their domain of application was expanded and eventually they were packaged and marketed as general purpose mathematical environments. They retain two major legacies of computer algebra: firstly they are very good at calculation but poor at dealing with logical side conditions. Secondly they include many transformation routines which, while algebraically completely valid, are only analytically valid for a small subset of the real line.

As a pilot study in using ATP technology to support better CAS calculation (there are a number of different ways to approaching hytbrid CAS/ATP systems) we have focussed on the problem of definite integration. This is a specific problem area that highlights both of the problems mentioned above.

While indefinite integration is "simply" (it's not always so simple) a matter of solving the differential algebra question:

Given a function f find a function F such that:
$$F' = f,$$

definite integration involves calculating the area under the curve of f and as such requires much more attention to the domain of definition of f (the location and nature of discontinuities).

CAS systems deal badly with the analytical side conditions of continuity that are inlcuded in the fundamental theorem of calculus:

Given a function f and a function F such that:
$$F' = f,$$
and given two limits a, b such that
$[a, b] \subseteq \mathrm{Dom}(f)$ and f is continuous on $[a, b]$ then
$$\int_a^b f(x)\mathrm{dx} = F(b) - F(a)$$

While there are tricks to avoid problems with discontinuities computer algebra systems in general ignore many discontinuities and deal poorly with most others. It is usually left to the user (sometimes without informing them of this fact) that they must manually check the side conditions on using the fundamental theorem of calculus.

In particular, computer algebra systems deal very poorly with calcuations involving parameters. In cases where they will correctly identify (and sometimes correctly work around) problem points in completely concrete cases, they will often fail when presented with identical problems involving even one parameter.

As part of a larger scheme considering strong definite integration algorithms we have produced a prototype Definite Integral Table Look Up (DITLU) system.

Details of the system may be found in [AGLM99a, AGLM99b].

2 Real Number Theorem Proving

Harrison developed a theory of real numbers and a medium sized analysis library as part of his PhD [Har98]. Unfortunately, this development was performed within HOL-Light, an unreleased and unsupported version of HOL. Since then much of his library has been ported to HOL 98. In parallel with this, Gottliebsen has developed an analogue of much of Harrison's work within PVS [Got00]. Our ongoing work includes developments of this library to solve two major problems: automatic solution of sets of inequations involving real parameters and transcendetal functions; automatic checking of continuity of elementary functions.

The development of libraries and tactics for real number theorem proving is now an acheivable goal, and the possible benefits in software verification and automated support of other systems are huge. Until recently this realm seemed beyond the capabilities of theorem proving software, but we have reached a stage where that is no longer true, and the development of continuous mathematics within theorem proving is becoming more and more important.

References

[AGLM99a] A. A. Adams, H. Gottliebsen, S. A. Linton, and U. Martin. Automated theorem proving in support of computer algebra: symbolic definite integration as a case study. In [Doo99], 253–260.

[AGLM99b] A. A. Adams, H. Gottliebsen, S. A. Linton, and U. Martin. VSDITLU: a verifiable symbolic definite integral table look-up. In [Gan99], 112–126.

[Doo99] S. Dooley, editor. *Proceedings of the 1999 International Symposium on Symbolic and Algebraic Computation.* ACM Press, 1999.

[Gan99] H. Ganzinger, editor. *Automated Deduction — CADE-16.* Springer-Verlag LNAI 1632, 1999.

[Got00] H. Gottliebsen. Transcendental Functions and Continuity Checking in PVS. In TPHOLS00 [TPH00]. To appear.

[Har98] J. Harrison. *Theorem Proving with the Real Numbers.* Springer-Verlag, 1998.

[TPH00] *Theorem Proving in Higher Order Logics*, 2000.

Resource Guided Concurrent Deduction

Christoph Benzmüller*; Mateja Jamnik*; Manfred Kerber*; Volker Sorge†
*School of Computer Science, The University of Birmingham
Edgbaston, Birmingham B15 2TT, England, UK
†Fachbereich Informatik (FB 14), Universität des Saarlandes
D-66041 Saarbrücken, Germany
C.E.Benzmuller|M.Jamnik|M.Kerber@cs.bham.ac.uk; sorge@ags.uni-sb.de

1 Motivation

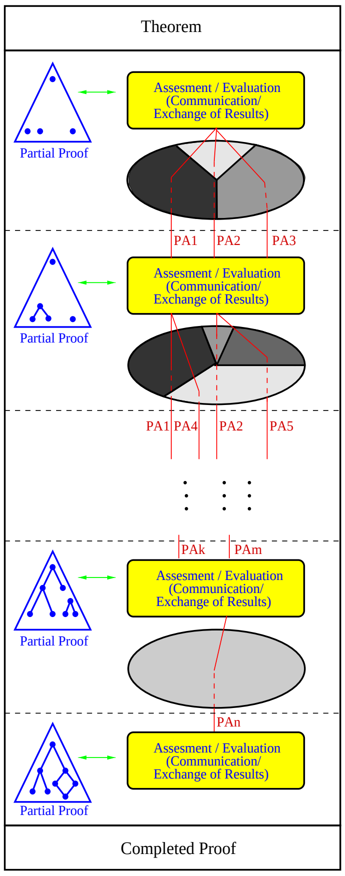
Our poster proposes an architecture for resource guided concurrent mechanised deduction which is motivated by some findings in cognitive science. Our architecture particularly reflects Hadamard's "Psychology of Invention" [Hadamard44]. In his study Hadamard describes the predominant rôle of the unconsciousness when humans try to solve hard mathematical problems. He explains this phenomenon by its most important feature, namely that it can make (and indeed makes) use of concurrent search (whereas conscious thought cannot be concurrent), see p. 22 Hadamard (1944): *"Therefore, we see that the unconscious has the important property of being manifold; several and probably many things can and do occur in it simultaneously. This contrasts with the conscious ego which is unique. We also see that this multiplicity of the unconscious enables it to carry out a work of synthesis."* That is, in Hadamard's view, it is important to follow different lines of reasoning simultaneously in order to come to a successful synthesis.

Human reasoning has been described in traditional AI (e.g., expert systems) as a process of applying rules to a working memory of facts in a recognise-act cycle. In each cycle one applicable rule is selected and applied. While this is a successful and appropriate approximation for many tasks (in particular for well understood domains), it seems to have some limitations, which can be better captured by an approach that is not only cooperative but also concurrent. And Minsky (1985) gives convincing arguments that the mind of a single person can and should be considered as a society of agents. Put in the context of mathematical reasoning this indicates that it is necessary to go beyond the traditional picture of a single reasoner acting on a working memory – even for adequately describing the reasoning process of a single human mathematician.

There are two major approaches to automated theorem proving, machine-oriented methods like the resolution method (with all its ramifications) and human-oriented methods. Most prominent amongst the human-oriented methods is the proof planning approach first introduced by Bundy (1988). In our poster we argue that an integration of the two approaches and the simultaneous pursuit of different lines in a proof can be very beneficial. One way of integrating the approaches is to consider a reasoner as a collection of specialised problem solvers, in which machine-oriented methods and planning play different rôles.

2 System Architecture

The architecture (for further details see Benzmüller et al. (1999)) that we describe here allows a number of proof search attempts to be executed in parallel. Each specialised subsystem may try a different proof strategy to find the proof of a conjecture. Hence, a number of different proof strategies are used at the same time in the proof search. However, following all the available strategies simultaneously would quickly consume the available system resources consisting of computation time and memory space. In order to prevent this, and furthermore, to guide the proof search we developed and employ a resource management concept in proof search. Resource management is a technique which distributes the available resources amongst the available subsystems (cf. Zilberstein (1995)). Periodically, it assesses the state of the proof search process, evaluates the progress, chooses a promising direction for further search and redistributes the available resources accordingly. If the current search direction becomes increasingly less promising then backtracking to the previous points in the search space is possible. Hence, only successful or promising proof attempts are allowed to continue searching for a proof. This process is repeated until a proof is found, or some other terminating condition is reached. An important aspect of our architecture is that in each evaluation phase the global proof state is updated, that is, promising partial proofs and especially solved subproblems are reported to a special plan server that maintains the progress of the overall proof search attempt. Furthermore, interesting results may be communicated between the subsystems (for instance, an open subproblem may be passed to a theorem prover that seems to be more appropriate). This communication is supported by the shells implemented around the specialised problem solvers. The resource management mechanism analyses the theorem and decides which subsystems, i.e., which provers, should be launched and what proportion of the resources needs to be assigned to a particular prover. The mechanism is also responsible for restricting the amount of information exchange between subsystems, so that not all of the resources are allocated to the communication. The Figure to the right demonstrates this concurrent resource management based proof planning architecture. The involved planning agents are represented by PA$_n$ and the ovals indicate the amount of resources assigned to them in each reasoning phase.



Theorem
Assesment / Evaluation (Communication/ Exchange of Results)
Partial Proof
PA1
PA2
PA3
Assesment / Evaluation (Communication/ Exchange of Results)
Partial Proof
PA1
PA4
PA2
PA5
PAk
PAm
Assesment / Evaluation (Communication/ Exchange of Results)
Partial Proof
PAn
Assesment / Evaluation (Communication/ Exchange of Results)
Partial Proof
Completed Proof

We argue that the effect of resource management leads to a less brittle search technique which we call focused search.

Breadth-first search is robust in the sense that it is impossible to miss a solution. However, it is normally prohibitively expensive. Heuristic search may be considered as the other extreme case, it is possible to go with modest resources very deep in a search tree. However, the search is brittle in that a single wrong decision may make it go astray and miss a solution, independently of how big the allocated resources are. Focused search can be considered as a compromise — it requires more resources than heuristic search, but not as much as breadth-first search. As a result, a solution can still be found even if the focus of the search is misplaced. Clearly, more resources are necessary in the case of a bad than of a good focus.

We currently realise the so-called focused proof search as an adaptation of the multi-agent planning architecture, MPA Wilkins and Myers (1998), in the proof planning domain. Important infrastructure for this enterprise is provided by the ΩMEGAproof development environment. The main component of MPA is a multi-agent proof planning cell, which consists of 1) several planning agents, 2) a plan server, 3) a domain server, and finally 4) a planning cell manager.

1. The quite heterogeneous reasoning systems (FO-Reasoners, HO-Reasoners, CAS, etc.) already integrated to ΩMEGA are available as planning agents. And an interactive user may become a concurrent planning agent as well.

2. The plan server stores promising partial proof plans returned by the planning agents in their previous runs within a unified data format. This enables backtracking on two distinct levels: we can backtrack within the actual proof plan by taking back single proof steps or subproofs contributed by some of the planning agents and we can completely shift to some alternative proof attempt that has been abandoned previously.

3. A domain server provides the necessary knowledge for the planning cell manager as well as for the single planning agents. In our context it consists of a structured database of mathematical theories. Moreover, it should contain domain specific knowledge relevant to certain planning agents.

4. The planning cell manager re-organises and controls the reasoning process in each iteration phase based on its (and/or the users') crucial evaluation and assessment considerations. Its prototype is based on the agent-architecture described in Benzmüller and Sorge (1999) allowing for a close and flexible integration of an interactive user into automated reasoning processes.

References

C. Benzmüller and V. Sorge. Critical Agents Supporting Interactive Theorem Proving. *Proceedings of EPIA-99*, Volume 1695 of *LNAI*, 1999. Springer.

C. Benzmüller, M. Jamnik, M. Kerber, and V. Sorge. Towards concurrent resource managed deduction. Tech-Report CSRP-99-17, The University of Birmingham, School of Computer Science, 1999.

A. Bundy. The Use of Explicit Plans to Guide Inductive Proofs. *Proceedings of the CADE-9*, volume 310 of *LNCS*, 1988. Springer Verlag, Berlin, Germany.

J. Hadamard. *The Psychology of Invention in the Mathematical Field*. Dover Publications, New York, USA; edition 1949, 1944.

M. Minsky. *The Society of Mind*. Simon & Schuster, New York, USA, 1985.

OMEGA Homepage. (`http://www.ags.uni-sb.de/~omega/`)

D. E. Wilkins and K. L. Myers. A Multiagent Planning Architecture. *Proceedings of AIPS'98*, 1998. AAAI Press, Menlo Park, CA, USA.

S. Zilberstein. Models of Bounded Rationality. In *AAAI Fall Symposium on Rational Agency*, Cambridge, Massachusetts, November 1995.

The Description Logic Reasoner CICLOP (Version 2.0)

François de Bertrand de Beuvron, Martina Kullmann,
David Rudloff, Michael Schlick, François Rousselot

LIIA, ENSAIS, 24, bd de la Victoire, F-67084 Strasbourg Cedex
beuvron@liia.u-strasbg.fr
http://www-ensais.u-strasbg.fr/liia/ciclop/ciclop.htm

1 Introduction

Description logic knowledge representation languages provide means for expressing abstract knowledge about concepts composing a terminology (TBox), as well as knowledge about concrete facts, i.e. objects instantiating the concepts, which form a world description (ABox). Since description logics are provided with a formal syntax and formal model-theoretic semantics, sound and complete reasoning algorithms can be formulated.

The description logic system CICLOP (Customizable Inference and Concept Language for Object Processing) [2] has been developed as a system for practical use. Thus, its facilities have been motivated by its applications. For Version 2 of CICLOP, the former system (Version 1.3) has been completely re-implemented and some extensions have been made.

2 System Description

The applications which have motivated the development of CICLOP are from the areas of configuration [7], semantic indexing of corpora [8], natural language querying of databases [3], and decision support for disaster response [4]. They have shown a need for facilities to manage the knowledge base, as well as for highly expressive knowledge representation languages.

To structure big knowledge bases, CICLOP allows for the definition of multiple disjoint TBoxes and corresponding ABoxes. Thus, concepts can be grouped together according to their context. Each TBox can be associated with an appropriate expressiveness. Even if the domains associated with different TBoxes are disjoint, individuals from different domains can be related by so-called connector roles.

The basic expressiveness supported by CICLOP is $\mathcal{ALC}$. Besides, it is possible to define features, primitive role hierarchies, inverse and transitive roles, as well as general concept inclusion axioms, i.e. CICLOP can deal with cyclic axioms. These language features can be combined as required by the application, as far as the resulting logic is still decidable.

Reasoning with respect to a knowledge base composed of several TBoxes and ABoxes with different expressivenesses is based on tableau algorithms which are sound and complete. Inferences are provided for the standard TBox and ABox reasoning tasks. The implementation uses optimization techniques such as lexical normalization and encoding, dependency directed backtracking, and model caching [5]. Furthermore, CICLOP uses blocking to deal with terminologies containing cycles, transitive and inverse roles. A description of the underlying algorithms can be found in [7].

Besides, CICLOP provides means for terminology closure. A closed TBox does not allow for the introduction of new concepts. Also, concepts have to be defined to be concrete or abstract, and individuals are forced to belong to concrete concepts only ([9], [7]). Dividing the knowledge base into multiple TBoxes and ABoxes allows for closing only parts of it.

Furthermore, CICLOP allows for the definition of TBoxes which represent concrete domains [1]. A numeric TBox allows for representing constraints in form of linear equations and inequalities, whereas in string TBoxes concepts can be defined by means of a set of possible and impossible strings. The implementation of a concrete domain interface for CICLOP is a joint work with the Department of Computer Science of King's College, London, whose description logic system WELLINGTON is also implemented in Java. The interface allows for associating concrete domains without important changes in the main system.

Since CICLOP has been implemented in Java (JDK 1.2), it runs on any operating system providing a Java Virtual Machine. Also, it can be executed as an applet within any web browser supporting Java 2. This makes the system easily usable. Besides, CICLOP is provided with a graphical user interface and a text interface based on the syntax specified in [6]. Furthermore, the Java API can be used to build other applications on top of the system.

3 Future Work

The user interface is planned to be extended to visualize the satisfiability checking process, i.e. to display internal events like constraint expansion, individual creation, and clash detection together with the expansion graph. Besides, the expressiveness of the system will be extended to feature chains with corresponding agreement and disagreement. Preliminary tests show that CICLOP performs well compared to other description logic systems, but standard benchmark tests have still to be done.

References

[1] F. Baader and P. Hanschke. A scheme for integrating concrete domains into concept languages. In *Proc. of the* 12th *International Joint Conference on Artificial Intelligence*, pages 452–457, Sidney, Australia, 1991.

[2] F. de Bertrand de Beuvron, F. Rousselot, M. Grathwohl, D. Rudloff, and M. Schlick. Ciclop. In *Proc. of the International Workshop on Description Logics '99, System Comparison*, Linköping, Sweden, 1999.

[3] F. de Bertrand de Beuvron, F. Rousselot, and D. Rudloff. Interpretation of description logics for natural language and for databases. In *Proc. of the International Workshop on Description Logics '97*, Paris, France, 1997.

[4] M. Grathwohl, F. de Bertrand de Beuvron, and F. Rousselot. A new application for description logics: Disaster management. In *Proc. of the International Workshop on Description Logics '99*, Linköping, Sweden, 1999.

[5] I. Horrocks. *Optimizing Tableaux Decision Procedures for Description Logics.* PhD thesis, University of Manchester, Manchester, England, 1997.

[6] P. F. Patel-Schneider and B. Swartout. Description-logic knowledge representation system specification. Technical report, AI Principles Research Department, AT&T Bell Laboratories, 1993.

[7] M. Schlick. *CICLOP: Les logiques de description appliques la configuration.* PhD thesis, Université de Haute Alsace de Mulhouse, France, 1999.

[8] A. Todirascu, F. de Bertrand de Beuvron, and F. Rousselot. Using description logics for indexing documents. In *Proc. of the IAR Annual Meeting '99*, Strasbourg, France, 1999.

[9] R. Weida. *Closed Terminologies and Temporal Reasoning in Descriptions for Plan Recognition.* PhD thesis, Columbia University, New York, NY, 1998.

Automata on infinite words and Temporal Logic Normal Forms.

Alexander Bolotov
Department of Computing and Mathematics
Manchester Metropolitan University, Manchester M1 5GD, UK.
A.Bolotov@doc.mmu.ac.uk, www.doc.mmu.ac.uk/STAFF/A.Bolotov

We consider the relationship between non-deterministic automata on infinite words [1, 12] and alternating automata [10, 1, 12] and a specific logical formulation based on a normal form for temporal logic formulae, called $\mathrm{SNF}_{\mathrm{PLTL}}$ [6]. While this normal form was developed for use with clausal resolution in temporal logics [5, 8], we here show how it can represent, *syntactically*, these types of automata in a high-level way [2].

The general problem structure that we are trying to solve is the analisys of a system specification followed by some (formal) verification of its properties.

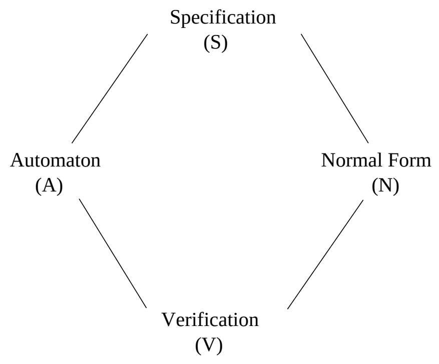


Figure 1: Specification-Verification Problem

If the specification (S) (see Fgure 1) is given in a high-level language, for example a logic, then the translation from (S) to an automaton (A) is exponential [12] in case of non-deterministic automata, for example, Büchi automaton. In contrast, the step from (A) to its verification (V) here is usually polynomial [4, 12], comprising an automaton emptiness check. Thus, if a model is given, we can see here the attraction of a *model-checking* approach. The situation is different when we carry out translation into alternating automata [1, 12], where, translation (S)$\rightarrow$(A) is polynomial while checking non-emptiness is hard. No direct methods of checking non-emptiness of alternating automata is known. The usual way is to simulate an alternating automaton by a standard non-deterministic automaton and then apply the emptiness check to the former [1, 12, 11].

If a specification (S) is given in a high-level language, for example in some logical language, then, aiming to apply as a verification (V) some form of efficient deduction, for example resolution, we carry out the translation from (S) to the Normal Form (N). Here the complexity is polynomial or often linear. In contrast, verification of formulae in the Normal Form is usually exponential since it involves some form of proof (in our case, clausal resolution). However, we are often able to use either improved proof strategies [7] or restricted forms of the normal form [3] in order to improve the practical efficiency of such proof.

One particular concern is the relationship between the Normal Form (N) given as normal form for PLTL ($\mathrm{SNF}_{\mathrm{PLTL}}$) and the Automaton (A) in the diagram above. We first show [2] that $\mathrm{SNF}_{\mathrm{PLTL}}$ can represent a specific type of non-deterministic automata, Büchi automata. In translating a problem specification into our normal form, we actually derive clauses within a fragment of quantified propositional linear temporal logic (QPLTL) [9, 13] such that formulae within $\mathrm{SNF}_{\mathrm{PLTL}}$ are existentially quantified and then effectively skolemize the normal form producing temporal formulae without any quantification.

Having established this relationship between $\mathrm{SNF}_{\mathrm{PLTL}}$ and Büchi automata, we will be able to represent problem specification directly as a set of formulae in Normal Form and apply resolution based verification technique to the latter. Also, we believe that varying the formulation of acceptance conditions in our syntactic representation of Büchi automata, our approach allows us to specify other types of ω-automata such as Rabin or Street automata. Finally, note that structurally $\mathrm{SNF}_{\mathrm{PLTL}}$ is similar to alternating automata. Thus, if we can show that $\mathrm{SNF}_{\mathrm{PLTL}}$ can represent alternating automata, then clausal resolution method applicable to a set of $\mathrm{SNF}_{\mathrm{PLTL}}$ clauses can be considered as another non-direct method of checking emptiness for alternating automata. This the subject of the ongoing work.

References

[1] O. Bernholtz, M. Y. Vardi, and P. Wolper. An automata-theoretic approach to branching-time model checking. In *Computer Aided Verification, Proc. 6th Int. Workshop*, volume 818 of *Lecture Notes in Computer Science*, pages 142–155, Stanford, California, June 1994. Springer-Verlag.

[2] A. Bolotov, M. Fisher, and C. Dixon. On the relationship between w-automata and temporal logic normal forms. In *To be published in Proc. of Advances in Modal Logic/International Conference on Temporal Logic 2000*, 2000.

[3] C. Dixon, M. Fisher, and M. Reynolds. Execution and Proof in Horn-Clause Temporal Logic. In *Proceedings the Second International Conference on Temporal Logic (ICTL)*, Manchester, July 1997. Kluwer.

[4] E. A. Emerson. Automated reasoning about reactive systems. In *Logics for Concurrency: Structures Versus Automata, Proc. of International Workshop*, volume 1043 of Lecture Notes in Computer Science. Springer-Verlag, 1996.

[5] M. Fisher. A Resolution Method for Temporal Logic. In *Proc. of the XII International Joint Conference on Artificial Intelligence (IJCAI)*, 1991.

[6] M. Fisher. A normal form for temporal logic and its application in theorem-proving and execution. *Journal of Logic and Computation*, 7(4), 1997.

[7] M. Fisher and C.Dixon. Guiding Clausal Temporal Resolution. In *Proceedings the Second International Conference on Temporal Logic (ICTL)*, Manchester, July 1997. Kluwer.

[8] M. Fisher, C. Dixon, and M. Peim. Clausal Temporal Resolution. To appear in ACM Transactions on Computational Logic (TOCL), 2000.

[9] Y. Kesten and A.Pnueli. A complete deductive system for QPTL. In *Proceedings of the 10th Annual IEEE Symposium of Logic in Computer Science*, 1995.

[10] D. E. Muller, A. Saoudi, and P. Schupp. Alternating automata, the weak monadic theory of the tree and its complexity.

[11] D. E. Muller and P. Schupp. Simulating alternating tree automata by nondeterministic automata: New results and new proofs of theorems of Rabin and McNaughton.

[12] M. Vardi. An automata-theoretic approach to linear temporal logic. In *Logics for Concurrency: Structures Versus Automata, Proc. of International Workshop*, volume 1043 of Lecture Notes in Computer Science. Springer-Verlag, 1996.

[13] P. Wolper. Temporal logic can be more expressive. *Information and Control*, 56(1–2):72–99, 1983.

Towards Automating Inductive Proofs for State Monads

Richard J. Boulton
Department of Computing Science, University of Glasgow
17 Lilybank Gardens, Glasgow G12 8QQ, Scotland
E-mail: `boulton@dcs.gla.ac.uk`

July 2000

1 Introduction

A *monad* is an algebraic structure that distinguishes between values and computations [Mog91]. Monads are used extensively in functional programming to produce programs that are easily modified and also to capture non-functional features such as input/output [Wad92]. They are also used in programming language semantics. Thus, reasoning about functional programs and certain styles of semantics may require proofs involving monads. Typically, the proofs will also involve recursive functions, so mathematical induction is usually required.

One successful approach to automating inductive proofs is proof planning, which uses artificial intelligence planning techniques where the objects considered are proof methods. I have been investigating how proof planning might be used to automate proofs involving monads and *state monads* in particular. I have identified two features of these proofs that challenge current proof planning technology: higher-order terms and nested recursion. Each of these is discussed briefly below but first I give an introduction to monads.

2 Monads and State Monads

There is more than one (equivalent) formulation of a monad. Here, the formulation using the functions `unit` and `bind` is used. In this formulation a monad is a triple consisting of a polymorphic type constructor `monad` and the two functions just mentioned. In their most general form the functions have the following types:

$$\begin{aligned}\texttt{unit} &: \alpha \to (\alpha)\texttt{monad}\\ \texttt{bind} &: (\alpha)\texttt{monad} \to (\alpha \to (\beta)\texttt{monad}) \to (\beta)\texttt{monad}\end{aligned}$$

and satisfy the following three equations:

$$\begin{aligned}\texttt{bind}\ (\texttt{unit}\ a)\ k &= k\ a\\ \texttt{bind}\ m\ \texttt{unit} &= m\\ \texttt{bind}\ m\ (\lambda a.\ \texttt{bind}\ (k\ a)\ h) &= \texttt{bind}\ (\texttt{bind}\ m\ (\lambda a.\ k\ a))\ h\end{aligned}$$

In a state monad the type $(\alpha)\texttt{monad}$ is specialised to $\texttt{state} \to \alpha \times \texttt{state}$ and `unit` and `bind` can be defined (and renamed) as follows:

$$\begin{aligned}\texttt{unitS}\; x &= \lambda s_0.\; (x, s_0) \\ \texttt{bindS}\; m\; f &= \lambda s_0.\; (\lambda(x, s_1).\; f\; x\; s_1)\; (m\; s_0)\end{aligned}$$

The function `bindS` is higher-order and expressions containing it typically involve explicit λ-abstractions.

3 Proof Planning for Higher-Order Logic

Until recently proof planning research had focused on first-order formulae but the λ*Clam* system developed at Edinburgh [RSG98] has provided a way to plan proofs about higher-order formulae. As indicated above, this will be a necessary feature for reasoning about state monads, but as yet I have not implemented such reasoning in λ*Clam* or any other system.

4 Nested Recursion

A nested recursive function is one who's definition involves a recursive call nested within an argument position of another recursive call, e.g.:

$$f(\mathsf{c}(x)) \;=\; \ldots f(\ldots f(\ldots)\ldots)\ldots$$

Recursive functions defined in terms of nested applications of `bindS` often have an implicit nested recursion due to the way `bindS` is defined.

It turns out that nested recursive functions can be split into two categories according to whether or not a nested recursive call occurs in a recursive argument position of the outer call. If the nested call is in a non-recursive position, less needs to be done to extend the existing proof planning methods, and fortunately monads are in this category. Nevertheless, extensions to λ*Clam* for the more benign form of nested recursion will be required if it is to be used for planning proofs about monads.

References

[Mog91] E. Moggi. Notions of computation and monads. *Information and Computation*, 93(1):55–92, July 1991.

[RSG98] J. Richardson, A. Smaill, and I. Green. System description: Proof planning in higher-order logic with λ*Clam*. In *Proceedings of the 15th International Conference on Automated Deduction (CADE-15)*, volume 1421 of *Lecture Notes in Artificial Intelligence*, pages 129–133, Springer, July 1998.

[Wad92] P. Wadler. The essence of functional programming. In *Conference Record of the Nineteenth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 1–14, Albuquerque, New Mexico, USA, January 1992.

Propositional Temporal Resolution Revised

Anatoli Degtiarev and Michael Fisher
Department of Computing and Mathematics, Manchester Metropolitan University
Manchester M1 5GD, U.K.
(e-mails: {A.Degtiarev,M.Fisher}@doc.mmu.ac.uk)

We propose a refinement of clausal proposional temporal resolution introduced in [Fis91] and considered thoroughly in [FDP00]. Knowledge of all basic notions related to this version of propositional temporal resolution is assumed.

Recall that the general form of the temporal resolution rule from [Fis91] is the following

$$\frac{A \Rightarrow \bigcirc \Box \neg l \quad C \Rightarrow \Diamond l}{C \Rightarrow (\neg A)\,\mathcal{W}\,l} \ (gtr)$$

Now, in some cases instead of involving the connective $\mathcal{W}$ ("unless"), we can manage with an "ordinary" version of resolution[1], which will be referred to as *(general) weak temporal resolution*:

$$\frac{A \Rightarrow \bigcirc \Box \neg l \quad C \Rightarrow \Diamond l}{A \wedge C \Rightarrow l} \ (gwtr)$$

This rule gives a weaker conclusion, however, it is more convenient. The convenience and relative sufficiency of this rule can be justified in particular by checking all examples in [FDP00], where only this rule is required. Moreover, in some cases, we retain completeness replacing temporal resolution by weak temporal resolution .

To give a standard form of weak temporal resolution for PLTL-clauses we follow [Fis91, FDP00] and join the derivation of $A \Rightarrow \bigcirc \Box \neg l$ and temporal resolution in a single combined rule:

$$\frac{A_1 \Rightarrow \bigcirc B_1 \quad \dots \quad A_n \Rightarrow \bigcirc B_n \quad C \Rightarrow \Diamond l}{(\bigvee_{i=1}^{n} A_i) \wedge C \Rightarrow l} \ (wtr)$$

where the *loop* side conditions

$$\vdash B_i \supset \neg l \quad \text{and} \quad \vdash B_i \supset \bigvee_{j=1}^{n} A_j \quad \text{for all} \quad i \in \{1, \dots, n\}.$$

have to be satisfied[2].

Proposition 1 A propositional temporal resolution system consisting of *initial resolution, step resolution and weak temporal resolution* rules is complete with respect to sets of PLTL-clauses containing at most one eventuality literal[3].

Unfortunately if an initial set of PLTL clauses contains more than one eventuality literal weak resolution is not sufficient. It is demonstrated by the following example.

[1]taking into account that the clause $C \Rightarrow \Diamond l$ is equivalent to $C \Rightarrow l \vee \bigcirc\Diamond l$.

[2]Let $I = \bigvee_{j=1}^{n} A_j$. Under the side conditions given above I can be considered as an *invariant formula* that gives derivability of $\Box\bigcirc\neg l$. Indeed, $I = \bigvee A_j \Rightarrow \bigvee \bigcirc B_j = \bigcirc \bigvee B_j \Rightarrow \bigcirc\neg l$ and $I = \bigvee A_j \Rightarrow \bigvee \bigcirc B_j = \bigcirc \bigvee B_j \Rightarrow \bigcirc \bigvee A_j = \bigcirc I$.

[3]A formula $\Diamond l$ (a literal l) from the right-hand side of a sometime PLTL-clause is called *eventuality* (*eventuality literal*).

Example Consider the following set of (merged) PLTL-clauses containing two eventuality literals:

1.	$\mathbf{start} \Rightarrow a \wedge \neg l_1 \wedge \neg l_2$;	3.	$(\neg a \wedge l_1 \wedge \neg l_2) \Rightarrow \bigcirc(\neg a \wedge l_1 \wedge \neg l_2)$;	5.	$a \Rightarrow \Diamond l_1$;
2.	$a \Rightarrow \bigcirc(\neg a \wedge (l_1 \vee l_2) \wedge (\neg l_1 \vee \neg l_2))$;	4.	$(\neg a \wedge \neg l_1 \wedge l_2) \Rightarrow \bigcirc(\neg a \wedge \neg l_1 \wedge l_2)$;	6.	$a \Rightarrow \Diamond l_2$.

This set is unsatisfiable, however we cannot derive a contradiction by weak temporal resolution (wtr).

To get completeness in general we use the augmentation approach of [FDP00] in a slightly different way. For each eventuality literal l occurring in an initial set S we introduce a new proposition w_l as a name of the formula $\bigcirc\Diamond l$. After that we apply, to every *sometime clause* $C \Rightarrow \Diamond l$ containing this eventuality literal, a fix point definition of the connective $\Diamond$ renaming the subformulas $\bigcirc\Diamond l$ by w_l. Namely,

$C \Rightarrow \Diamond l$ is replaced by
$\{C \Rightarrow l \vee w_l, w_l \Rightarrow \bigcirc\Diamond l\}$ is replaced by
$\{C \wedge \neg l \Rightarrow w_l, w_l \Rightarrow \bigcirc\Diamond l, w_l \Rightarrow \bigcirc(l \vee \bigcirc\Diamond l)\}$ is replaced by
$\{C \wedge \neg l \Rightarrow w_l, w_l \Rightarrow \bigcirc\Diamond l, w_l \Rightarrow \bigcirc(l \vee w_l)\}$ is replaced by
$\{\mathbf{start} \Rightarrow \neg C \vee l \vee w_l, \mathbf{true} \Rightarrow \neg C \vee l \vee w_l, w_l \Rightarrow \bigcirc\Diamond l, w_l \Rightarrow \bigcirc(l \vee w_l)\}$.

It results in the same augmented set of clauses as in [FDP00]. Let us denote it by S^{aug}. This set is satisfiable if, and only if, the initial set S is satisfiable because renaming does not affect satisfiability.

Now, we define an analogue of the weak temporal resolution rule for S^{aug} as follows:

$$\frac{A_1 \Rightarrow \bigcirc B_1 \quad \ldots \quad A_n \Rightarrow \bigcirc B_n \quad w_l \Rightarrow \bigcirc\Diamond l}{(\bigvee_{i=1}^{n} A_i) \Rightarrow \neg w_l}\ (atr)$$

with the same loop side conditions for the step clauses as above.

Let us refer to this rule as *augmented temporal resolution* (atr) . It is not difficult to see that augmented resolution is sound. Indeed, the loop side conditions imply $(\bigvee_{i=1}^{n} A_i) \supset \bigcirc \Box\neg l$, and in addition the last premise of the rule is equivalent to $\bigcirc \Box\neg l \supset \neg w_l$. Completeness of augmented resolution is obtained by simple adaptation of the completeness proof given in [FDP00].

Proposition 2 A propositional temporal resolution system consisting of *initial resolution, step resolution and augmented temporal resolution* rules is complete with respect to augmented sets of PLTL-clauses.

Example (continuation) We can obtain a contradiction from 1-4 as follows.

At first let us produce augmentation clauses corresponding to eventuality literals l_1 and l_2.

7. $w_1 \Rightarrow \bigcirc\Diamond l_1$;	9. $w_1 \Rightarrow \bigcirc(l_1 \vee w_1)$;	11. $\mathbf{start} \Rightarrow (\neg a \vee l_1 \vee w_1)$;	13. $\mathbf{true} \Rightarrow \bigcirc(\neg a \vee l_1 \vee w_1)$;
8. $w_2 \Rightarrow \bigcirc\Diamond l_2$;	10. $w_2 \Rightarrow \bigcirc(l_2 \vee w_2)$;	12. $\mathbf{start} \Rightarrow (\neg a \vee l_2 \vee w_2)$;	14. $\mathbf{true} \Rightarrow \bigcirc(\neg a \vee l_2 \vee w_2)$.

Now, we can apply augmented temporal resolution (atr) to pairs $7, 4$ and $8, 3$ obtaining clauses 15 and 16:

15. $\mathbf{true} \Rightarrow \bigcirc(a \vee l_1 \vee \neg l_2 \vee \neg w_1)$; 16. $\mathbf{true} \Rightarrow \bigcirc(a \vee \neg l_1 \vee l_2 \vee \neg w_2)$.

Further we can derive by step resolution from 2 and $9, 10, 15, 16$

17. $a \wedge w_1 \wedge w_2 \Rightarrow \bigcirc\mathbf{false}$.

At last we can get a contradiction just by initial resolution from $17, 1, 11, 12$.

The refinements of clausal propositional temporal resolution described in this abstract have appeared in the process of our work on developing resolution decision procedures for restricted fragments of the the first-order temporal logic. This work is in progress now.

References

[FDP00] M. Fisher, C. Dixon, and M. Peim. Clausal temporal resolution. *ACM Transactions on Computation Logic*, 2000. To appear.

[Fis91] M. Fisher. A resolution method for temporal logic. In *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence (IJCAI)*. Morgan Kaufman, 1991.

Reasoning in Description Logics with Wellington 1.0 System Description[*]

Ulrich Endriss

Department of Computer Science, King's College London, Strand, London WC2R 2LS, UK, Email: endriss@dcs.kcl.ac.uk

http://www.dcs.kcl.ac.uk/research/groups/logic/wellington/

1 Introduction

Description logics are formal knowledge representation languages with a relatively simple syntax and well-defined semantics. According to the description logic paradigm, knowledge is divided into a terminological part (TBox), where concepts like *beverages that are carbonated and have some ingredient that is alcoholic* are defined, and an assertional part (ABox), where individuals are related to each other and asserted as being instances of certain concepts. For an introduction to the field and an overview of main directions of current research we refer to [1].

In this paper we introduce a new description logics based knowledge representation and reasoning tool, the WELLINGTON system, which is currently being developed by the Group of Logic and Computation at King's College London.

2 System Description

Unlike a number of other description logic systems, that have been written in functional languages, WELLINGTON is being developed in Java. By choosing a mainstream object-oriented language rather than a functional one we hope to make the system more accessible to users outside the description logics community. Java in particular allows for the development of (almost) platform-independent software. For most system configurations applets can be launched from a web browser without the need to install any additional software. WELLINGTON 1.0 is available as both a Java application and an applet and may be run online over the Internet or can be downloaded for local use from our project web site (see top of page).

Currently, the system supports ABox reasoning in the standard description logic $\mathcal{ALC}$ (without global axioms). Using the ABox consistency checking algorithm it is also possible to check the consistency of a given concept formula and to check the subsumption relation between two given concept formulas.

WELLINGTON 1.0 implements a multi-modal tableaux-like calculus with a number of optimisations, including lexical normalisation, semantic branching with heuristic guided search, beta simplification (disjunctions entailed by one of their subformulas on the same branch are not expanded), non-branching beta rules (also called boolean constraint propagation), and backjumping. Furthermore, in order to minimise the time required for comparing formulas the implementation assures that for each (syntactically) distinct formula not more than one object is

[*]This work is part of the *Data Driven Logic Algorithms* project, which is funded by the EPSRC under grant reference number GR/L91818.

created. An overview of optimisation techniques for description logic tableaux may be found in [5]. WELLINGTON seems to perform well, but to date no detailed evaluation has been carried out.

On the calculus level, one aspect where our system apparently differs from many others is, that one proof gives rise to exactly one tableau, on which each branch may hold formulas labelled by different ABox individuals. The standard *algorithmic* presentation [4], on the other hand, assumes a number of so-called nodes, each of which contains the formulas associated with one of the ABox individuals. These formulas again are (at least implicitly) structured as a tableau. Besides being semantically clearer and closer to the presentation of tableaux calculi for e.g. modal logics, we believe that our approach will simplify the integration of mechanisms for reasoning about concrete domains [2].

3 Future Developments

WELLINGTON 1.0 is only the beginning. In the long run we intend to develop a system for ABox and TBox reasoning in the description logic proposed in [6], which extends $\mathcal{ALC}$ by a number of features, notably arithmetical constraints over numerical aspects of sets of role-fillers, complex role terms and hierarchies, as well as various generalised quantifiers. The current prototype can already be used to manage knowledge bases encoded in that language, but the reasoning services are yet to be implemented.

The first obvious extension of the current version will be to allow for unfolding of acyclic concept definitions. Then it will be possible to check ABox consistency, concept consistency, and concept subsumption with respect to a TBox. This in turn will provide the basis for a concept classification algorithm.

Furthermore, we plan to augment WELLINGTON with the ability to reason about concrete domains [2]. In cooperation with the LIIA Strasbourg we are currently defining a general Java interface for concrete domain reasoning that will be integrated into both CICLOP [3], the description logic system developed in Strasbourg, and WELLINGTON. This will allow us to exchange implementations of particular domains without the need to alter any code in the main systems.

In the context of concrete domains we are particularly interested in domains that can be used to combine description logics with temporal reasoning mechanisms.

References

[1] F. Baader. Logic-based knowledge representation. In M. J. Wooldridge and M. Veloso, editors, *Artificial Intelligence Today, Recent Trends and Developments.* Springer-Verlag, 1999.

[2] F. Baader and P. Hanschke. A scheme for integrating concrete domains into concept languages. In *Proceedings of the 12th International Joint Conference on Artificial Intelligence, IJCAI'91*, Sydney, 1991.

[3] F. Beuvron, F. Rousselot, M. Grathwohl, D. Rudloff, and M. Schlick. CICLOP (system description). In *Proceedings of the International Workshop on Description Logics, DL'99*, Linköping, 1999.

[4] I. Horrocks. *Optimising Tableaux Decision Procedures for Description Logics.* PhD thesis, University of Manchester, 1997.

[5] I. Horrocks and P. F. Patel-Schneider. Optimising description logic subsumption. *Journal of Logic and Computation*, 9(3):267–293, 1999.

[6] H. J. Ohlbach. A theory resolution style ABox calculus. Extended abstract. In *M4M, Methods for Modalities 1, Workshop Proceedings.* ILLC, University of Amsterdam, 1999.

Efficient Control of Temporal Reasoning

M.Carmen Fernández-Gago,
Centre for Agent Research and Development
Department of Computing and Mathematics,
Manchester Metropolitan University
Email: C.Gago@doc.mmu.ac.uk

1 Introduction

An important part of all logics is proof theory which is concerned with how statements in the logical language can be combined. A proof transforms statements in the language using only the inference rules and the axioms. Employing inference rules such as binary resolution and factoring, the resolution method is complete although it is not very efficient. For applications in Computer Science, we require that such methods become more effective. In particular, we require strategies. A strategy is a rule or set of rules that governs the use of inference rules. In resolution-based theorem-provers developed for classical logics, a particulary successful strategy for avoiding the generation of redundant information during proof has been the *set of support* strategy [8].

Temporal Logic is a variety of non-classical logic used in a wide variety of areas within Computer Science and Arificial Intelligence, for example, Robotics [1], Databases [2], Program Specification [3], Hardware verification [4], and Agent-Based Systems [5]. Proof in temporal logic is inherently complex and its implementation can be achieved by different appoaches. Within the Centre for Agent Research and Development a proof method for temporal logics has been developed. This method is based upon the use of clausal resolution [6, 7] and it has been defined, proved correct, implemented and extended in a number of ways. However, in a number of cases, the basic method leads to the generation of an unnecessarily large set of formulas during the proof. As some of these formulas are irrelevant, i.e., are not needed in the proof, it is clear that refinements are needed. In order to make the temporal reasoning more efficient, high-level strategies will be needed. Following its success within classical logics, it is our intention to develop a *set of support* strategy for temporal resolution.

2 Some Strategies for Temporal Resolution

The resolution method developed for linear temporal logics [6], besides the translation to a normal form, involves classical resolution between formulae that occur at the same moment in time, and temporal resolution over states.
In the first case, i.e, where the clauses contain no eventualities ($\Diamond l$, i.e, *l holds now or at sometime in the future*), we have found a method to apply the set of support strategy. Further we have proved that by choosing the set of support in a specific way, the resolution method is complete. That is, we choose the set of support as all the *rules* with positive right hand sides (analogously negative right hand sides). With this strategy the amount of irrelevant information during the proof is reduced.
In the second case, the resolution method is more complex and involves the detection of a set of rules known as a *loop*, that is, rules that together imply $\Box l$ (*l holds now and at all moments in the future*) for resolution with $\Diamond\neg l$. The resolution rule is as follows:

$$\frac{\begin{array}{ccc} \phi_1 & \Rightarrow & \bigcirc\psi_1 \\ \vdots & \vdots & \vdots \\ \phi_n & \Rightarrow & \bigcirc\psi_n \\ \chi & \Rightarrow & \Diamond\neg l \end{array}}{\chi \Rightarrow (\neg \bigvee_{i=1}^{n} \phi_i)\mathcal{W}\neg l}$$

where $\bigwedge_{i=1}^{n}(\phi_i \Rightarrow \bigcirc(l \wedge \bigvee_{i=1}^{n} \phi_j))$ ($\bigcirc$ means *in the next moment in time*). This condition ensures that the set of $\phi_i \Rightarrow \bigcirc\psi_i$ rules together imply $\bigvee_{i=1}^{n} \phi_i \Rightarrow \bigcirc \square l$.

But the process of detecting the rules that characterise a loop is not easy or obvious, although it is crucial for the resolution method. Because of that, a new resolution rule is introduced [9] and it is proved sound and complete. The idea behind this new temporal resolution rule is that, rather than insisting that we have a loop, we derive a more complex resolvent, that allows for the possibility that such a loop does not exist. This new temporal resolution is:

$$\frac{\begin{array}{ccc} \phi_1 & \Rightarrow & \bigcirc\psi_1 \\ \vdots & \vdots & \vdots \\ \phi_n & \Rightarrow & \bigcirc\psi_n \\ \chi & \Rightarrow & \Diamond\neg l \end{array}}{\chi \Rightarrow \left[(\Diamond\neg\text{looping}) \vee (\neg \bigvee_{i=1}^{n} \phi_i)\mathcal{W}\neg l \right]}$$

This new resolvent includes the side condition (looping), needed for applying the traditional temporal resolution , in the resolvent. If such a loop is not obvious the resolvents produced will guide the detection of the right loop, after to apply step resolution. If such a loop exists, the new resolvent turns out to be equivalent form.
In our current work we are considering how to make the initial 'guess' of a loop and how to use output from the proof to improve our guess.
We hope to apply these results to the development of strategies for temporal resolution that allows us to reduce the search space. In particular, we are interested to incorporate the set of support strategy, hoping it will be so successfully like in the classical case.

References

[1] M. Shanahan, *Solving the Frame Problem.* MIT Press, 1997.

[2] J. Chomicki and G. Saake, eds, *Logics for Databases and Information Systems.* Kluwer, 1998

[3] Z. Manna and A. Pnueli, *The Temporal Logic of Reactive and Concurrent Systems: Specification.* Springer-Verlag, 1992.

[4] G. Holzmann.The Model-Checker SPIN. *IEEE Trans. on Software Engineering* 23(5), 1997.

[5] A. Rao and M. Georgeff. Modeling Rational Agents within a BDI-Architecture. In *Proc. Int. Conf. on Principles of Knowledge Representation and Reasoning.* Morgan Kaufmann Publishers, 1991.

[6] M. Fisher. A Resolution Method for Temporal Logic. In *Proc. Int. Joint Conf. on Artificial Intelligence.* Morgan Kauffmann Publishers, 1991.

[7] C. Dixon. Temporal Resolution using a Breadth-First Search Algorithm. *Annals of Mathematics and Artificial Intelligence, 22.* Baltzer Science Publishers, 1998.

[8] L. Wos, R. Overbeek, E.Lusk and J. Boyle. *Automated Reasoning: Introduction and Application*, Prentice-Hall International, 1984.

[9] M. Fisher and C. Dixon, Guiding Clausal Temporal Resolution. In *Advances in Temporal Logic*,Kluwer Academic Publishers,1999.

[10] W. Mc.Cune, Solution of the Robbins Problem. *Journal of Automated Reasoning* 19(3), 1997.

[11] C. Dixon and M. Fisher, The Set of Support Strategy in Temporal Resolution. *In Proc. Int. Workshop on Temporal Reasoning.* IEEE Computer Society Press, 1998.

Automatic Generation of Implied Constraints: Project Description

Alan M. Frisch[*] and Toby Walsh[†]
Artificial Intelligence Group
Dept. of Computer Science, Univ. of York
York, YO10 5DD, United Kingdom

Constraint satisfaction is a highly successful technology for tackling a wide variety of search problems including resource allocation, transportation and scheduling. Several recent studies show that implied constraints added by hand to a problem representation can lead to significant reductions in search (e.g. [8]). The aim of this project is to develop, analyse and evaluate methods for generating implied constraints automatically. One of the key ideas will be to combine theorem proving techniques (like ordered resolution) with constraint satisfaction algorithms.

As an example of the value of implied constraints, consider colouring the nodes in a graph so that adjacent nodes have different colours. This constraint satisfaction problem models a variety of assignment problems like exam and classroom timetabling. If we have a near k-clique in which all but one pair of k nodes are connected and $k-1$ colours available, then we can infer an additional constraint that the two unconnected nodes must take the same colour. Adding this implied constraint explicitly to the problem representation could prevent a backtracking algorithm like forward checking from exploring an exponential number of partial colourings for the k nodes in the near k-clique.

Though those who formulate problems for constraint solvers appreciate the importance of adding implied constraints, there has been little research on how to generate such implied constraints automatically outside of highly focused domains like planning (see, for example, [4]). One exception is [5], which generalises resolution to multi-valued clauses (in which variables can take more than just the two values *True* and *False*), and proves that implied constraints generated by the closure of this operation will eliminate search. In practice, we cannot expect to eliminate search completely as the closure can be exponentially large to compute. One of the objectives of this research is to identify how much of the closure to generate to reduce the total time needed to solve a problem.

Many inference techniques can be viewed as methods for generating restricted classes of implied constraints. For example, consistency techniques like arc-consistency generate implied unary constraints. As a second example, nogood recording techniques [2] generate implied constraints from dead-ends in search. In the area of operations research cutting planes are highly effective at strengthening linear relaxations and at pruning search [7]. Cutting planes are linear inequalities implied by the original set of inequalities. Hooker has shown [7] that resolution (and its generalisation to multi-valued clauses) is a general method for generating cutting planes analogous to more traditional techniques like Chvátal's method. As a final example, in propositional satisfiability inference techniques such as directional resolution [3] can be highly competitive with more traditional branching techniques like the Davis-Putnam procedure [1]. Such methods generate resolvents which can again be viewed as implied constraints.

The aims of this project are to be achieved through four major objectives. Though each primarily focuses on the use of implied constraints for systematic search, we will also consider their use in local search.

The first objective is to develop theorem proving techniques for generating implied constraints automatically. We will encode the initial problem representation into a propositional

[*]Email: frisch@cs.york.ac.uk, tel: +44 1904 432745.
[†]Email: tw@cs.york.ac.uk, tel: +44 1904 432793.

theory and attempt to determine the extent to which standard theorem proving technologies, such as resolution, can generate constraints that have previously proven useful when generated by hand.

As theorem proving techniques are likely to generate both useful and unuseful constraints, the second objective is to develop heuristics for identifying which of the generated constraints to retain and which to discard. We will attempt to determine whether measures such as its "constrainedness" [6], size or tightness can guide this decision. It is known that an ordered form of resolution [5] can be used to generate constraints sufficient to give backtrack free search for a fixed variable ordering. But which of these implied constraints is necessary? Can we give similar results for dynamic variable orderings?

The first two objectives focus on the generation of constraints prior to the start of search. The third objective considers algorithms that interleave the generation of implied constraints with the search process itself. Consistency techniques, such as arc-consistency, can be viewed as generating unary constraints during search. The great success of these techniques in practice motivates us to consider whether other methods of deriving implied constraints during search can also be effective. A critical question to be addressed is how to partition effort between inference (generation of implied constraints) and search.

The fourth and final objective is to apply these results to a closely related domain, propositional satisfiability (SAT) and some extensions of SAT. In recent years, there has been considerable interest in encoding constraint satisfaction and other problems like planning into SAT, and then using either an efficient complete procedure like Davis-Putnam or a fast semi-decision procedures like GSAT or WalkSAT. The success of such an approach often depends critically on the implied constraints included in the encoding [4]. Are the techniques developed for generating implied constraints for constraint satisfaction problems useful for generating implied constraints when encoding problems into SAT?

The research methodology followed in this project is that of theoretical analysis backed up by large scale empirical tests. We will therefore prototype the various technologies being explored, and test these implementations on the wide variety of problems found in CSPLib.

We expect automated methods for generating implied constraints to become a vital component of the next generation of constraint satisfaction toolkits, and we hope to contribute to this development. Our results will be available at http://www.cs.york.ac.uk/aig/projects/implied.

References

[1] M. Davis, G. Logemann, and D. Loveland. A machine program for theorem-proving. *Communications of the ACM*, 5:394–397, 1962.

[2] R. Dechter. Enhancement schemes for constraint processing: Backjumping, learning and cutset decomposition. *Artificial Intelligence*, 41(3):273–312, 1990.

[3] R. Dechter and I. Rish. Directional resolution: The Davis-Putnam procedure, revisited. In *Proceedings of KR-94*, pages 134–145, 1994. A longer version is available from `http://www.ics.uci.edu/~irinar`.

[4] M. Ernst, T.D. Millstein and D.S. Weld. Automatic SAT-Compilation of Planning Problems. In *Proceedings of the 14th IJCAI*, pages 1169-1177. International Joint Conference on Artificial Intelligence, 1997.

[5] A.M. Frisch. Solving Constraint Satisfaction Problems with NB-Resolution. In S. Muggleton, D. Michie and Luc De Raedt, editors, *Machine Intelligence 16*, 2000. Electronic Transactions in Artificial Intelligence. Available from `http://www.cs.york.ac.uk/~frisch/papers/mi16.ps`.

[6] I.P. Gent, E. MacIntyre, P. Prosser, and T. Walsh. The constrainedness of search. In *Proceedings of the 13th National Conference on AI*, pages 246–252. American Association for Artificial Intelligence, 1996.

[7] J.N. Hooker. Constraint satisfaction methods for generating valid cuts. In D. L. Woodruf, editor, *Advances in Computational and Stochastic Optimization, Logic Programming, and Heuristic Search*, pages 1–30. Kluwer, 1997.

[8] B.M. Smith, K. Stergiou, and T. Walsh. Modelling the Golomb ruler problem. In *Proceedings of the IJCAI-99 Workshop on Non-Binary Constraints.* International Joint Conference on Artificial Intelligence, 1999. Also available as APES report, APES-11-1999 from http://apes.cs.strath.ac.uk/reports/apes-11-1999.ps.gz.

Hyperresolution for Guarded Formulae

Lilia Georgieva[1], Ullrich Hustadt[2] and Renate A. Schmidt[1]

[1] Department of Computer Science, University of Manchester
Manchester M13 9PL, United Kingdom, {georgiel,schmidt}@cs.man.ac.uk
[2] Centre for Agent Research and Development, Manchester Metropolitan University
Manchester M1 5GD, United Kingdom, U.Hustadt@doc.mmu.ac.uk

Abstract. Recently we have been investigating the use of hyperresolution as a decision procedure and model builder for guarded formulae. In general hyperresolution is not a decision procedure for the entire guarded fragment. However we show that there are natural fragments which can be decided by hyperresolution [9]. As hyperresolution is closely related to various tableaux methods the work is also relevant for tableaux methods. We compare our approach to hypertableaux, and mention the relationship to other clause classes solvable by hyperresolution.

The guarded fragment of first-order logic was introduced in Andréka, van Benthem and Neméti [1, 2]. It extends the modal fragment which corresponds to basic modal logic (via the relational translation) and is an important decidable class which contains many extended modal logics and description logics. Among the most notable properties of the guarded fragment in addition to decidability are Craig interpolation, bisimulation invariance, Beth definability, finite model property, and preservation under submodels.

Several extensions of the guarded fragment, like the loosely guarded fragment [7, 12], guarded fixpoint logic [11], or monadic GF^2 with transitive guards [8] have been shown decidable. The various decision procedures exploit the finite model property, use ordered resolution, alternating automata, or embeddings into monadic second-order logic. This is an interesting contrast to the literature on decidable modal logics and description logics, where tableaux-based decision procedures are predominant for testing satisfiability (see for example [5, 10]).

In [17] Lutz, Sattler and Tobies investigate whether tableaux-based decision procedures exist for subclasses of the guarded fragment. They introduce the fragment $\mathrm{GF1}^-$ which is obtained from the first guarded fragment GF1 [2] by restricting the way the variables may occur in guards, and show that the fragment is decidable by semantic tableaux [17].

In [9] we continue their line of investigation. However, we make use of the close correspondence between tableaux-based decision procedure for modal logics and hyperresolution combined with eager splitting on an encoding of modal formulae in clausal logic, as described in [4, 15], and in [13, 14] for description logics. By using a structure preserving transformation of guarded formulae into clausal form we are able to recast the method in a first-order setting using hyperresolution, combined with positive factoring and eager splitting.

The method of proving termination of hyperresolution combined with positive factoring and eager splitting for the relational translation of extended multi-modal logics used in [4, 15] does not generalise to $\mathrm{GF1}^-$. We investigate a different argument which takes into consideration the form of the derived clauses. The obtained decision procedure is practical. Standard resolution provers can be used without adaptation.

Furthermore, following the approach we show decidability of a larger class of formulae. We describe such generalisations of $\mathrm{GF1}^-$in [9].

We also investigate how our method relates to other inference methods such as hypertableaux [3], and how the work fits into the bigger picture of hyperresolution as a decision procedure [6, 16].

Currently we are looking into defining an abstract atom complexity measure μ in analogy to Leitsch [16] which would generalise the specific complexity measures and orderings used in the termination proofs presented in this paper and in [4, 13–15]. We are also attempting to define a larger solvable class which would accommodate more formulae outside the guarded fragment. Further it would be of interest to extend the approach to the entire guarded fragment, by using proper blocking conditions in the context of resolution.

References

1. H. Andréka, I. Németi, and J. van Benthem. Modal languages and bounded fragments of predicate logic. *J. Philos. Logic*, 27(3):217–274, 1998.
2. H. Andréka, J. van Benthem, and I. Németi. Back and forth between modal logic and classical logic. *Bull. IGPL*, 3(5):685–720, 1995.
3. P. Baumgartner, U. Furbach, and I. Niemelä. Hyper tableaux. In *European Workshop on Logic in AI (JELIA'96)*, volume 1126 of *LNAI*, pages 1–17. Springer, 1996.
4. H. de Nivelle, R. A. Schmidt, and U. Hustadt. Resolution-based methods for modal logics. *Logic J. IGPL*, 8(3):265–292, 2000.
5. F. M. Donini, M. Lenzerini, D. Nardi, and A. Schaerf. Reasoning in description logics. In G. Brewka, editor, *Principles in Knowledge Representation*, Studies in Logic, Language and Information, pages 191–236. CSLI Publications, Stanford, 1996.
6. C. G. Fermüller, A. Leitsch, U. Hustadt, and T. Tammet. Resolution decision procedures. In *Handbook of Automated Reasoning*. Elsevier, 2000. To appear.
7. H. Ganzinger and H. de Nivelle. A superposition decision procedure for the guarded fragment with equality. In *Proc. LICS'99*, pages 295–303. IEEE Computer Society Press, 1999.
8. H. Ganzinger, C. Meyer, and M. Veanes. The two-variable guarded fragment with transitive relations. In *Proc. LICS'99*, pages 24–34. IEEE Computer Society, 1999.
9. Lilia Georgieva, Ullrich Hustadt, and Renate A. Schmidt. Hyperresolution for guarded formulae. In Peter Baumgartner and Hantao Zhang, editors, *Proceedings of the Third International Workshop on First-Order Theorem Proving (FTP 2000)*, volume 5/2000 of *Fachberichte Informatik*, pages 101–112. Institut für Informatik, Universität Koblenz-Landau.
10. R. Goré. Tableau methods for modal and temporal logics. In M. D'Agostino, D. Gabbay, R. Hähnle, and J. Posegga, editors, *Handbook of Tableau Methods*. Kluwer, 1999.
11. E. Grädel. Decision procedures for guarded logics. In *Automated Deduction—CADE-16*, volume 1632 of *LNAI*, pages 31–51. Springer, 1999.
12. E. Grädel. On the restraining power of guards. Manuscript. Submitted to the *J. Symbolic Logic*, 1999.
13. U. Hustadt and R. A. Schmidt. On the relation of resolution and tableaux proof systems for description logics. In *Proc. IJCAI'99*, pages 110–115. Morgan Kaufmann, 1999.
14. U. Hustadt and R. A. Schmidt. Issues of decidability for description logics in the framework of resolution. In *Automated Deduction in Classical and Non-Classical Logics*, volume 1761 of *LNAI*, pages 191–205. Springer, 2000.
15. U. Hustadt and R. A. Schmidt. Using resolution for testing modal satisfiability and building models. To appear in the *SAT 2000* Special Issue of *J. Automated Reasoning*, 2000.
16. A. Leitsch. Deciding clause classes by semantic clash resolution. *Fundamenta Informatica*, 18:163–182, 1993.
17. C. Lutz, U. Sattler, and S. Tobies. A suggestion of an n-ary description logic. In *Proc. DL'99*, pages 81–85. Linköping University, 1999.

Forward and Backward Chaining in Linear Logic

James Harland[1] David Pym[2] Michael Winikoff[1]
jah@cs.rmit.edu.au, pym@dcs.qmw.ac.uk, winikoff@cs.rmit.edu.au

[1] Department of Computer Science, Royal Melbourne Institute of Technology, GPO Box 2476V, Melbourne, 3001, Australia
[2] Department of Computer Science, Queen Mary and Westfield College, University of London, Mile End Road, London, E1 4NS UK

Backward chaining is a standard technique in automated deduction, particularly in *logic programming* systems, often taking the form of a version of Robinson's *resolution rule* [10]. The fundamental question is to determine whether or not a given formula follows from a given set of formulæ, and there are various techniques which can be used to guide the search for a proof.

An instance of this approach is the analysis of logic programming in intuitionisitic logic [8]. The standard such analysis is that of Miller et al. [8], based around the notion of *uniform proofs*. These are defined in terms of the sequent calculus, which is well-known to be suited for analyses of backward-chaining.

With more recent interest in logic programming languages based on *linear logic* [5], the natural extension of these backward chaining techniques to linear logic has been much studied [1, 2, 4, 6, 9]; generally, it follows a similar pattern to intuitionistic logic. The details of the analysis are more intricate than in intuitionistic logic, and there are a number of points of diversion amongst the various approaches, but the same general procedure is followed.

Whilst the sequent calculus is a good basis for backward chaining, other systems for inference in intuitionistic logic provide forward chaining capabilities. Hilbert-type systems are the oldest and perhaps best-known of such systems [7]. Such systems allow different logics to be specified by different sets of axioms whilst maintaining modus ponens as the sole means of inference.

Another technical expression of forward chaining in intuitionistic logic may be found in the T_P operator used in the semantics of logic programs. Here, a mapping is made from interpretations to interpretations, in which the image is the result of applying the rules of the program to the initial interpretation via a combination of modus ponens and unification. The semantics of the program is then given by the least fixed point of this operator. It is interesting to note that this forward chaining system is traditionally used to provide a fixpoint semantics for SLD-resolution [3], a backward-chaining system.

It should be noted that a key property of the modus ponens rule in intuitionistic logic is that it preserves equivalence: $\phi \wedge (\phi \supset \psi) \equiv \phi \wedge \psi$. This strong property greatly simplifies the analysis of this rule of inference.

A combination of both backward and forward chaining may be found in deductive database systems such as Aditi [11]. In such systems, which are based on variants of Prolog, forward chaining is generally used in order to compute all answers to a query using efficient join algorithms and other techniques from relational databases, whilst backward chaining is used for less data-intensive computational tasks (such as format conversions).

The properties of forward and backward chaining systems for intuitionistic logic are generally well-understood. However, the question of how to best integrate the two models still remains. Give that Hilbert-type systems generally do not make any provision for backward chaining techniques, it seems reasonable to address the question of integration by investigating the incorporation of forward chaining features into the sequent calculus.

This is achieved by inserting *directed cuts* into an otherwise cut-free sequent calculus proof. Cut-free proofs are generally used in proof search to avoid a significant amount of non-determinism (*i.e.*, having to choose a cut formula arbitrarily); in the case of a directed cut, the cut formula will be calculated from the antecedent, and hence will not have the same problem.

The presence of both backward and forward chaining mechanisms in deductive databases suggests that a similar integration for linear logic will prove fruitful, especially as linear logic has been used to model database updates, state and action problems and concurrency.

However, the use of modus ponens in linear logic is not as simple as in intuitionistic logic since modus ponens does not preserve linear equivalence. For example, in linear logic $p \otimes (p \multimap q) \vdash q$ but in proving q we have to "consume" p.

We have recently developed a method for integrating forward chaining into the sequent calculi for intuitionistic and linear logics. One particular result of interest is that we can show that these inference rules respect an encoding of intuitionistic logic programs into linear ones.

Acknowledgements Harland is grateful for the hospitality of the Department of Computer Science of Queen Mary and Westfield College, University of London during a period of sabbatical leave. Pym is partially supported by a Fellowship from the EPSRC.

References

1. J.-M. Andreoli. Logic Programming with Focusing Proofs in Linear Logic. *J. Logic Computat.* 2(3), 1992.
2. J.-M. Andreoli and R. Pareschi. Linear Objects: Logical Processes with Built-in Inheritance. Proceedings of the International Conference on Logic Programming, 496-510, Jerusalem, June, 1990.
3. M.H. van Emden and R.A. Kowalski, The Semantics of Predicate Logic as a Programming Language, *Journal of the Association for Computing Machinery* 23:4:733-742, October, 1976.
4. D. Galmiche and G. Perrier. On proof normalization in Linear Logic. Theoretical Computer Science 135:76-100, 1994.
5. J.-Y. Girard. Linear Logic. *Theoretical Computer Science* 50, 1-102, 1987.
6. J. Hodas, D. Miller. Logic Programming in a Fragment of Intuitionistic Linear Logic: Extended Abstract. Proceedings of the Symposium on Logic in Computer Science, 32-42, Amsterdam, July, 1991.
7. S.C. Kleene. *Introduction to Metamathematics*. North Holland, 1952.
8. D. Miller, G. Nadathur, F. Pfenning and A. Ščedrov. Uniform Proofs as a Foundation for Logic Programming. *Annals of Pure and Applied Logic:51:125-157*, 1991.
9. D.J. Pym, J.A. Harland. A Uniform Proof-theoretic Investigation of Linear Logic Programming. Journal of Logic and Computation 4:2:175-207, April, 1994.
10. J.A. Robinson. A Machine-Oriented Logic Based on the Resolution Principle. *Journal of the Association for Computing Machinery* 12:1:23-41, 1965.
11. J. Vaghani, K. Ramamohanarao, D. Kemp, Z. Somogyi, P. Stuckey, T. Leask and J. Harland. The Aditi Deductive Database System. *VLDB Journal* 3:2:245-288, April, 1994.

Congruence Classes with Logic Variables

Joe Hurd[*]
Computer Laboratory
University of Cambridge
joe.hurd@cl.cam.ac.uk

12 July 2000

Abstract

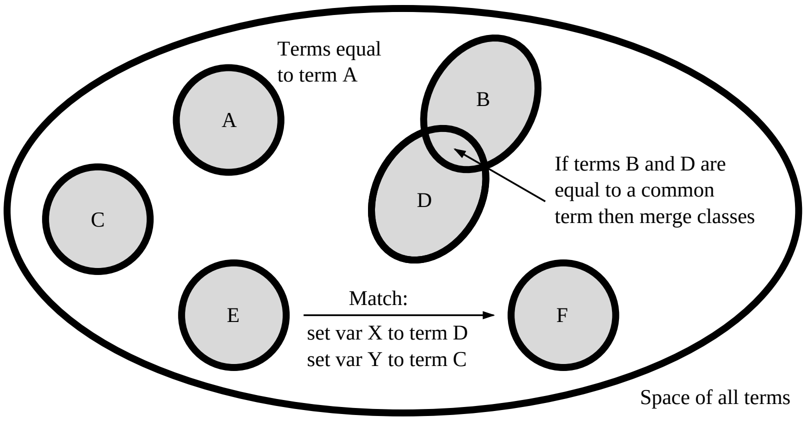
We are improving equality reasoning in automatic theorem-provers, and congruence classes provide an efficient storage mechanism for terms, as well as the congruence closure decision procedure. We describe the technical steps involved in integrating logic variables with congruence classes, and present an algorithm that can be proved to find all matches between classes (modulo certain equalities). An application of this algorithm makes possible a *percolation* algorithm for undirected rewriting in minimal space; this is described and an implementation in `hol98` is examined in some detail.

1 Summary

Blending together equality steps (Leibniz' rule of substituting equals for equals) and deduction steps (e.g., Modus Ponens or specialization) in a proof search is problematic. Equality tends to dramatically blow up the search space, because of the vast number of ways of expressing a given term.

In this work we take *congruence classes*—a way of storing terms that maximizes sharing and performs congruence closure—and show how they can be used when the terms contain logic variables. This makes them appropriate for storing terms in a deductive prover.

[*]Supported by an EPSRC studentship



Our results so far are:

- a matching algorithm between classes, guaranteed to find all matches (modulo certain equalities) between classes;
- a *percolation* algorithm that performs undirected rewriting on the classes (using the equalities represented by the classes);
- an implementation of the above as a derived rule in hol98, with results on some test cases.

The full paper is available at the following URL:

`http://www.cl.cam.ac.uk/users/jeh1004/research/papers/congruence1.html`

Practical Proof Methods for Combined Modal and Temporal Logics

Ullrich Hustadt

Centre for Agent Research and Development, Manchester Metropolitan University, Chester Street, Manchester M1 5GD, United Kingdom
`U.Hustadt@doc.mmu.ac.uk`

For a number of years, temporal and modal logics have been applied outside pure logic in areas such as formal methods, theoretical computer science and artificial intelligence. In our research we are particularly interested in the use of modal logics in the characterisation of complex components within software systems as *intelligent* or *rational* agents. This approach allows the system designer to analyse applications at a much higher level of abstraction. In order to reason about such agents, a number of theories of rational agency have been developed, for example the BDI (Rao and Georgeff 1991) and KARO (van Linder, van der Hoek, and Meyer 1996) frameworks. The leading agent theories and formal methods in this area all share similar logical properties, more precisely, they all exhibit (i) an *informational* component, being able to represent an agent's beliefs (by the modal logic $\mathsf{KD45}$) or knowledge (by the modal logic $\mathsf{S5}$), (ii) a *dynamic* component, allowing the representation of dynamic activity (by temporal or dynamic logic), and, (iii) a *motivational* component, often representing the agents desires, intentions or goals (by the modal logic KD).

While many of the basic properties of such combinations of modal and temporal or dynamic logics are well understood (Baader and Ohlbach 1995; Fagin et al. 1996; Gabbay 1996; Wolter 1998), very little work has been carried out on practical proof methods for such logics.

Our aim in recent work has been to develop proof methods that are general enough to capture a wide range of combinations of temporal and modal logics, but still provide viable means for effective theorem proving. Currently, we are investigating an approach with the following properties:

– The approach covers the combination of discrete, linear, temporal logic with extensions of multi-modal K_m by any combination of the axiom schemata $\mathsf{4}$, $\mathsf{5}$, B, D, and T. This extends the results presented in (Dixon, Fisher and Wooldridge 1998; Wooldridge, Dixon, and Fisher 1998).
– Instead of combining two calculi operating according to the same underlying principles, like for example two tableaux-based calculi, we combine two different approaches to theorem-proving in modal and temporal logics, namely the translation approach for modal logics (using first-order resolution) and the SNF approach for temporal logics (using modal resolution).
– The particular translation we use has only recently been proposed by de Nivelle (1999) and can be seen as a special case of the T-encoding introduced by Ohlbach (1998). It allows for conceptually simple decision procedures for

extensions of $\mathsf{K4}$ by ordered resolution without any reliance on loop checking or similar techniques.

In more detail, this approach consists of (i) a normal form transformation of formulae of the combined logics into sets of so-called *SNF$_K$ clauses* (similar to those presented in Dixon et. al. 1998), (ii) a translation of modal subformula in SNF$_K$ clauses into a first-order language, and (iii) a calculus $\mathsf{C}_{\mathsf{MTL}}$ for the combined logic which can be divided into standard resolution inference rules for first-order logic and a modified version of the temporal resolution inference rules of Fisher (1991).

The calculus $\mathsf{C}_{\mathsf{MTL}}$ provides a decision procedure for combinations of the basic multi-modal logic K_m and its extensions by arbitrary combinations of the axiom schemata $\mathsf{4}$, $\mathsf{5}$, B, D, and T with linear, temporal logic.

For a more detailed description of our approach see Hustadt, Dixon, Schmidt, and Fisher (2000). Related work on proof methods for the KARO framework of agency can be found in Hustadt, Dixon, Schmidt, Fisher, Meyer, and van der Hoek (2000).

References

Baader, F. and Ohlbach, H. J. (1995). A multi-dimensional terminological knowledge representation language. *Journal of Applied Non-Classical Logics*, 2:153–197.

Blackburn, P. and de Rijke, M. (1997). Why combine logics? *Studia Logica*, 59(1):5–27.

de Nivelle, H. (1999). Translation of S4 into GF and 2VAR. Manuscript.

Dixon, C., Fisher, M., and Wooldridge, M. (1998). Resolution for temporal logics of knowledge. *Journal of Logic and Computaton*, 8(3):345–372.

Fagin, R., Halpern, J. Y., Moses, Y., and Vardi, M. Y. (1996). *Reasoning About Knowledge.* MIT Press.

Fisher, M. (1991). A Resolution Method for Temporal Logic. In *Proc. IJCAI'91*, pages 99–104. Morgan Kaufmann.

Gabbay, D. M. (1996). Fibred semantics and the weaving of logics. Part 1. Modal and intuitionistic logics. *Journal of Symbolic Logic*, 61(4):1057–1120.

Hustadt, U., Dixon, C., Schmidt, R. A., and Fisher, M. (2000). Normal Forms and Proofs in Combined Modal and Temporal Logics. In *Proc. FroCoS'2000*, *LNAI* 1794, pages 73–87. Springer.

Hustadt, U., Dixon, C., Schmidt, R. A., Fisher, M., Meyer, J-J., and van der Hoek, W. (2000) Verification within the KARO Agent Theory. In Proc. First Goddard Workshop on Formal Approaches to Agent-Based Systems. Springer, to appear.

Ohlbach, H. J. (1998). Combining Hilbert style and semantic reasoning in a resolution framework. In *Proc. CADE-15*, *LNAI* 1421, pages 205–219. Springer.

Rao, A. S. and Georgeff, M. P. (1991). Modeling agents withing a BDI-architecture. In *Proc. KR-91*, pages 473–484. Morgan Kaufmann.

van Linder, B., van der Hoek, W. and Ch. Meyer, J.-J. (1996). How to motivate your agents. In *Intelligent Agents II*, pages 17–32, *LNAI* 1037. Springer.

Wolter, F. (1998). Fusions of Modal Logics revisited, *Advances in Modal Logic*, Volume 1, *CSLI Lecture Notes* 87, pages 361–379. CSLI Publications.

Wooldridge, M., Dixon, C., and Fisher, M. (1998). A tableau-based proof method for temporal logics of knowledge and belief. *Journal of Applied Non-Classical Logics*, 8(3):225–258.

The existential theories of term algebras with the Knuth-Bendix orderings are decidable

Konstantin Korovin Andrei Voronkov

Department of Computer Science
University of Manchester
{korovin|voronkov}@cs.man.ac.uk

We consider term algebras with simplification orderings (which are monotonic, and well-founded). Solving constraints (quantifier-free formulas) in term algebras with this kind of orderings has several important applications like pruning search space in automated deduction and proving termination and confluence of term rewriting systems. Two kinds of ordering are normally used in automated deduction: Knuth-Bendix ordering and various versions of recursive path orderings. There exists extensive literature on solving recursive path ordering constraints [1, 4], but no algorithms for solving Knuth-Bendix ordering constraints are known. We proved that the problem of solving Knuth-Bendix ordering constraints is decidable and NP–hard.

Let us briefly describe the proof, for the full version we refer to [3]. We consider term algebras in a finite signature Σ with at least one constant, denoted TA(Σ). Let us now define Knuth-Bendix orderings on TA(Σ) [2]. The definition of Knuth-Bendix ordering is parametrized by a *weight function* on Σ, i.e., a function $w : \Sigma \to \mathbb{N}$, and a linear ordering $\gg$ on Σ. We require from the weight function the following: if $w(f) = 0$ and f is unary, then f must be the greatest w.r.t. $\gg$ in Σ, and weights of constants are positive. We define the weight of a ground term as a sum of weights of functors occurring in the term. Given a weight function w and a linear ordering $\gg$ on Σ, the *Knuth-Bendix ordering* on TA(Σ) is the binary relation $>_{KB}$ defined as follows. For any ground terms $g(t_1, \ldots, t_n)$ and $h(s_1, \ldots, s_k)$ we have $g(t_1, \ldots, t_n) >_{KB} h(s_1, \ldots, s_k)$ if

1. $|g(t_1, \ldots, t_n)| > |h(s_1, \ldots, s_k)|$

or

2. $|g(t_1, \dots, t_n)| = |h(s_1, \dots, s_k)|$ and one of the following holds:

   (a) $g \gg h$ or

   (b) $g = h$ and for some $1 \leq i \leq n$ we have $t_1 = s_1, \dots, t_{i-1} = s_{i-1}$ and $t_i >_{KB} s_i$.

To prove the theorem we first extend our term algebra with the natural numbers with addition and the weight function on terms. We show how to transform arbitrary quantifier-free formula into an equivalent disjunction of conjunctions such that all occurring terms in the formula are variables. Then we introduce formulas which express that there exists at least n terms of the weight x, where n is a fixed parameter and show how to write them as linear Diophantine equations on weights of terms. Using obtained formulas we transform initial formula into an equivalent constraint which consists of linear Diophantine equations on weights of terms. Then we show that satisfiability of that constraint is equivalent to satisfiability of the systems of linear Diophantine equations over the natural numbers.

References

[1] H. Comon. Solving symbolic ordering constraints. *International Journal of Foundations of Computer Science*, 1(4):387–411, 1990.

[2] D. Knuth and P. Bendix. Simple word problems in universal algebras. In J. Leech, editor, *Computational Problems in Abstract Algebra*, pages 263–297. Pergamon Press, Oxford, 1970.

[3] K. Korovin and A. Voronkov. A decision procedure for the existentional theory of term algebras with the Knuth-Bendix orderings. Technical Report UMCS-2000-6-3, Department of Computer Science, University of Manchester, January 2000.

[4] R. Nieuwenhuis. Simple LPO constraint solving methods. *Information Processing Letters*, 47:65–69, 1993.

Automatic Generation of Concurrent Provers

Raul H. C. Lopes

Departamento de Informática – UFES, Caixa Postal 01-9011/290670 – Brazil
raulh@inf.ufes.br

1. Introduction

A framework is outlined that combines automatic generation of proof search strategies for theorem provers with concurrent proof search. The framework consists of two programs: $\mathcal{P}_2$, that implements an algorithm that generates prioritized logic programs, representing proof search strategies; and $\mathcal{P}_2$*-frame*, a proof search engine that can use strategies generated by $\mathcal{P}_2$ to drive concurrent provers.

$\mathcal{P}_2$ has been applied to generate provers for Intuitionistic Propositional Calculus(**IPC**), modal logics T, S4, and S5, and for classical second-order logic (see, for example, [4].) Figure 1 shows a few difficult theorems proved by a concurrent prover, using a proof search strategy generated by $\mathcal{P}_2$. Particularly interesting are the automatic proofs obtained for the problems 1 to 4 (taken from [1]) and for Cantor's theorem that the power set of a set S is larger than S (problem 15.)

1. $\vdash \forall U \forall V((U < 0) \supset \neg(U = abs(V))) \supset$
   $\exists A(\forall Y(\neg A(abs(Y))) \wedge A(-(2)))$
2. $\vdash \forall X(\exists U(X = (2 * U)) \leftrightarrow \neg\exists V((X+1) = (2 * V))) \supset$
   $\exists A \forall X(A(X) \leftrightarrow \neg A(X+1))$
3. $\vdash \forall P \forall Y((\forall A((A(0) \wedge \forall X(A(X) \supset A(X+1))) \supset A(Y)) \wedge$
   $(P(0) \wedge \forall X(P(X) \supset P(X+1)))) \supset P(Y))$
4. $\forall X \forall Y \forall Z((F_2(X,Y) = F_2(Z,Z)) \supset (X = Y)) \vdash$
   $\forall P \forall U \forall V((\forall A((A(F_2(0,0)) \wedge$
   $\forall X \forall Y(A(F_2(X,Y)) \supset A(F_2(F_1(X), F_1(Y))))) \supset A(F_2(U,V))) \wedge P(U)) \supset P(V))$
5. $\vdash \neg\exists G \forall F \exists J(G(J) = F)$

Fig. 1. A problem set for $\mathcal{P}_2$

2. Concurrent proof strategies

$\mathcal{P}_2$ can generate proof search strategies, that are composed of assertions (called *methods*) about the uses of the inference rules of a given logic taken from examples of proofs. It assumes that proofs can be performed in a goal-oriented fashion, and that they can be decomposed in *proof steps*, containing a *goal* and a designation of an inference rule applied to it. Its main components are: a set of randomized algorithms for generating *methods* from proof steps, and algorithms for ordering *methods*, for matching methods with proof goals, and for establishing redundancy of *methods*.

A method is generated from a step by *schematization* of its goal into a meta-goal (uniform replacement of variables by meta-variables), and from other methods by *lifting* (random replacement of one meta-variable by a new meta-variable), and *thinning* (random dropping of a formula of the meta-goal.) A method is *confirmed* by a step when its *meta-goal* unifies with the step's goal and they both designate the same inference rule. A method is *contradicted* by a step when its *meta-goal* unifies with the step's goal and they designate different inferences rule. Methods with negative conditions can be automatically generated and they restrict the range of steps that one method can match (see [4]).

Methods are ordered with respect to the set of input steps. A method is given highest priority if it has no contradiction in the given examples. There can be several methods in that condition and they concurrently compete to be applied in a proof. Immediately after them are assigned to highest priority methods whose contradictions are all confirmed by methods in the preceding level. This idea is used by $\mathcal{P}_2$ to define partial order on methods. A *synchronous single pool* parallel branch-and-bound algorithm ([3]) is used to drive the proof search: methods with the same priority are scheduled for concurrent matches with a given goal. A successful match produces a new branch in the search tree.

3. Conclusion

The $\mathcal{P}_2$*-frame* has two strong points: it can generate proof search strategies for arbitrary logics; and the proof search strategies it produces can drive proofs either with traditional depth-first or breadth-first search procedure (as described in [4]), or with parallel algorithms, based, for example, on branch-and-bound. This last option is particularly attractive in higher-order proving, where the the undecidability of unification can lead provers into endless loops. Concurrent matches were important in several steps of the proofs obtained for theorems 1, 2, and 4 of fig. 1.

References

1. W.W. Bledsoe and Guohui Feng, *SET-VAR*, Journal of Automated Reasoning **11** (1993), 293–314.
2. Harald Ganzinger (ed.), *Proceedings of the 16th International Conference on Automated Deduction*, Springer-Verlag, 1999, LNAI, 1632.
3. Bernard Gendron and Teodor Gabriel Crainic, *Parallel branch-and-bound algorithms: survey and synthesis*, Operations Research **42** (1994), no. 6, 1042–1066.
4. Raul H.C. Lopes, *Automatic generation of proof search strategies for second-order logic*, In Ganzinger [2], LNAI, 1632, pp. 414–428.

Finite Model Building for Propositional Gödel–Logics as an Example for Projective Logics

Markus Moschner[1]
TU Vienna

The aim of model building consists in delivering a model together with a decision result. For classical logics there are substantial achievements on automated model building [3, 4]. Although a remarkable amount of work exists for nonclassical logics (particularly for model building and automated theorem proving), the work of Negri and Plato [6] on propositional intuitionistic logic is one of the few on the problem of automatization.
My proposal refers to Gödel logics which are projective logics in the sense of [2] with respect to automated building of truth–valued models.
Prominent nonclassical logics (Gödel, intuitionistic, Łukasiewicz or lattice–valued logics) come with ordered–structures for truth–values or semantic structures (for Gödel and Łukasiewicz logics only linear truth–value structures are sufficient) Since the construction of a (counter) model is a task of semantics the usage of such orders seems suggestive. Transitivity of partial orders obviously plays a crucial role. The interpretation of the connectives refers usually to a partial ordering of its arguments and the corresponding mapping:

- conjunction — Infimum
- disjunction — Supremum
- implication — order–relation of arguments (case distinction)

Strictly speaking implication in projective logics may be defined through pure case distinction (in the finite many–valued case), thus argumentation via partial–ordering can get inelegant. Gödel logic simplifies matters through the linear order of the truth values (ensuring trichotomy between the bounds of the truth–value set) and the projector–like behavior of negation and implication([5]). Negation evaluates only to the bound values, whereas implication projects either onto the second argument or to the designated value 1. This accounts also for infinite–valued Gödel logics, since this work aims mainly at counter models the analogy to projection logics does not get senseless (on the propositional level at least).
Processing a formula from top down to its propositional variables (or constants) gives order conditions for its propositional variables:
a) for con– and disjunction there are order relations between the formula and its subformulas,
b) for implication there are order relations between its subformulas (but the value of one subformula – its succedent – may determine the value of the whole formula).
Intuitively this process is similar to a tableaux method, but "constraints" for the range of valuations are given at the nodes. These "constraints" may be order relations between different formulas — not only direct given values.
The appearance of a contradiction within these conditions means that there is no model (for the refutation — so it must be a tautology); such a contradiction has to be within every case distinction. Otherwise the conditions need not give single values for a variable, but (as a side–effect) there is information about the minimal numbers of truth–values for a nontrivial counter–model (respectively: if the conditions give 2 distinct variables which may not be interpreted to 0 or 1, a 4–valued truth–set is necessary). In general there is no restriction to finite–valued logics because only information about the structure of the (interpretations of) subformulas from finite

[1]Technische Universität Wien, Wiedner Hauptstr. 8–10, A–1040 Vienna, Austria; e–mail: *moschm@logic.at*

formulas is constructed.
My aim is an implementation that yields an automated generation of a model (if existing) for a refuted formula of propositional Gödel logics; since Gödel logics are a special case of projective logics I am interested in extending the method to classes of projective logics.

There is an elucidation of the proposed method via an extension of infinite–valued Gödel logics with 0–1–projections from [1].
The Δ–operator, representing such a 0–1–projection, gets the designated value 1 if the evaluation of the formula gets 1, otherwise 0, Such projections can express a certain order between the subformulas of an implication. Take the implicational hull with 0–1–projections of a formula (the Δ–operator gets the designated value 1 if the evaluation of the formula gets 1, otherwise 0):
$\mathcal{H}_F = \{\, \Delta(F_1 \to F_2) \,|\, F_1, F_2 \in \mathit{Subform}(F) \,\}$;
a disjunction of all conjunctions $\bigwedge_{f \in \mathcal{H}_F} (\neg^i f)$ for $i \in \{0, 1\}$ (in each conjunction every formula occurrence is either negated or unnegated) expresses all the possible schemas of valuations for a formula. The result of the proposed method (resp. one result of some case distinctions) represents a part of one of these conjunctions (there is an empty disjunction for no refutation); by laws of classical logics a completion to some of the conjunctions can be done. At the beginning of the procedure the hypothesis $\neg \Delta (1 \to A)$ is added; for a tautology always at least one conjunction admits $\Delta (1 \to A)$ for a valuation. Thus refutations of tautologies give only contradictions.

It has to be clarified if this method can be done efficiently. Further investigations have to clarify the adaptability of this method to a broader class of projective logics. But finite–valued logics can be seen as a special case of projective logics. A Within this context such a conception seems worthy of further investigations.

References

[1] M. Baaz. Infinite–valued Gödel logics: 0–1–projections and relativizations. In P. Hájek, editor, *Proc. Gödel'96, Logic Foundations of Mathematics, Computer Science and Physics — Kurt Gödel's Legacy*, Lecture Notes in Logic 6, pages 23–33. Springer–Verlag, 1999.

[2] Matthias Baaz and Christian G. Fermüller. Analytic Calculi for Projective Logics. In Neil V. Murray, editor, *Automated Reasoning with Analytic Tableaux and Related Methods*, LNAI 1617, pages 36–50. Springer–Verlag, 1999.

[3] Ricardo Caferra and Nicolas Peltier. *Decision Procedures Using Model Building Techniques*, pages 131–144. Number 1092 in LNCS. Springer Verlag, 1996.

[4] C.G. Fermüller and A. Leitsch. Hyperresolution and automated model building. *Journal of Logic and Computation*, 6(2):173–230, 1996.

[5] Siegfried Gottwald. *Mehrwertige Logik. Eine Einführung in Theorie und Anwendungen.* Akademie–Verlag Berlin, Berlin, GDR, 1989.

[6] Sara Negri and Jan von Plato. *From Kripke Models to Algebraic Counter–valuations*, pages 246–261. 1998.

Theorem Proving for Temporal Logics of Knowledge or Belief

Cláudia Nalon

Centre for Agent Research and Development
Department of Computing and Mathematics
Manchester Metropolitan University
Manchester M1 5GD, UK

Email: C.Nalon@doc.mmu.ac.uk

June, 2000

Temporal logics have been investigated in computer science for over twenty years. They were introduced in [12] as a tool for the specification of reactive systems and, since then, they have been shown to be useful in a variety of applications (e.g. temporal databases, model checking). Nevertheless, for some applications, a temporal component is not enough to describe the properties of the system. To deal with properties of distributed and multi-agent systems, temporal logic is often augmented with modal operators of either knowledge or belief. For description of particular systems, it is also necessary to restrict attention to a class of models. For instance, by adding the axiom $K\bigcirc\phi \Rightarrow \bigcirc K\phi$, synchronous systems with perfect recall can be described. Axioms involving operators from both logics are known as *interaction* axioms.

Within the CARD, a proof method, based on the principle of resolution [11], for temporal logics of knowledge and belief has been proposed [3, 6] and a prototype implementation has been developed. Interactions between knowledge and time, their properties, and issues of complexity have been discussed in [10, 4, 7]. A resolution-based method for a temporal logic of knowledge for synchronous systems with perfect recall has been described in [1]. It has been shown that by adding new clauses to the normal form, no new resolution rules are required and the theorem prover for temporal logics of knowledge (without interactions) can be used with a small number of changes. Current work involves the investigation of other useful interactions, such as perfect recall (alone) [10] and no learning with synchrony ($\bigcirc K\phi \Rightarrow K\bigcirc\phi$) [7]. Interactions between belief and time, as those described in [5] and [9], will also be investigated.

Adding interaction axioms increases (sometimes dramatically) the complexity of the logic. For instance, the complexity of validity for the single agent case for temporal logic of knowledge (without interactions) is PSPACE. If the synchrony and perfect recall axiom is added, complexity is double-exponential time [8]. So, the development of strategies to guide the search for a proof is essential. Successful strategies, such as set of support, which is applied in both classical [13] and temporal [2] logics, will be considered when developing strategies for interacting logics of time and knowledge or belief.

References

[1] C.Dixon and M.Fisher. *Clausal Resolution for Logics of Time and Knowledge with Synchrony and Perfect Recall.* Submitted, 2000.

[2] C.Dixon and M.Fisher. *The Set of Support Strategy in Temporal Resolution.* In Proceedings of TIME-98 the Fifth International Workshop on Temporal Representation and Reasoning, Sanibel Island, Florida, IEEE Computer Society Press, May, 1998.

[3] C.Dixon, M.Fisher, and M. Wooldridge. *Resolution for Temporal Logics of Knowledge.* Journal of Logic and Computation, volume 8, number 3, 1998.

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

[5] M. Fisher and M. Wooldridge. *On the Formal Specifications and Verification of Multi-Agent Systems.* International Journal of Cooperative Information Systems, 6(1), January, 1997.

[6] M. Fisher, M. Wooldridge and C. Dixon. *A Resolution-Based Proof Method for Temporal Logics of Knowledge and Belief.* In Proceedings of the International Conference on Formal and Applied Practical Reasoning (FAPR-96), Bonn, Germany, June, 1996.

[7] J.Y.Halpern, R. van der Meyden, and M.Y. Vardi. *Complete Axiomatizations for Reasoning About Knowledge and Time* Submitted for publication, 1997.

[8] J.Y.Halpern and M.Y. Vardi. *The Complexity of Reasoning about Knowledge and Time. I Lower Bounds.* Journal of Computer and Systems Sciences, 38:195-237, 1989.

[9] J.J.C.Meyer and W. van der Hoek. *Epistemic Logic for Computer Science and Artificial Intelligence*, vol. 41 Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 1995.

[10] R. van der Meyden. *Axioms for Knowledge and Time in Distributed Systems with Perfect Recall.* In Proceedings of the Ninth IEEE Symposium on Logic in Computer Science, pages 448-457, 1994.

[11] J.A.Robinson. *A Machine-Oriented Logic Bases on The Resolution Principle.* ACM Journal, 12(1):23-41, January, 1965.

[12] A. Pnueli *The Temporal Logic of Programs.* In *Proceedings of the 18th Symposium on the Foundations of Computer Science*, Providence, November, 1977.

[13] L.Wos, D.Carson, and G. Robinson. *Efficiency and Completeness of the Set of Support Strategy in Theorem Proving.* ACM Journal, 12:536-541, October, 1965.

Application of simplification theories

Mauricio Osorio[1], Juan Carlos Nieves[1], Gabriel Cervantes[2]

[1] Universidad de las Americas
CENTIA
Sta. Catarina Martir,
Cholula, Puebla
72820 Mexico
josorio@mail.udlap.mx
[2] Benemerita Universidad Autonoma de Puebla
Escuela de Ciencias de la Electronica
75579 Puebla, Mexico
gcervantes@kim.ece.buap.mx

1 Abstract

In this abstract we present different applications of "simplification of theories". By simplification of theories we understand a set of relations defined over a class of theories with a fixed language. The only two general properties that this relations respect are: First, that they are polynomial time computable. Second, if a theory P_1 is related to P under a transformation (that is P_1 is obtained from P using a transformation) and m is a model of P_1, then m is also a model for P.

We discuss applications in three different fields in applied logic: First order theory proving, Well behaved semantics and Answer set programming.

In first order theory proving, given a consistent first order theory T and an atom a, we may be interested in the derivability of a by T, that is, $T \models a$? Using OTTER the problem is traduced as showing that $T \cup \{\neg a\}$ is inconsistent. Unfortunately, this may cause a loop in the process (using OTTER, a well known theorem proving system) when $T \cup \{\neg a\}$ is consistent. In general terms we propose the use of"simplification of theories" that help us to construct a model for $Cl\,\{(T \cup \{\neg a\}\}$, where Cl denotes the clausal form of the given theory. If we succeed in finding such a model, then $T \cup \{\neg a\}$ has a model and therefore $T \not\models a$. We however need some (strong) conditions on T to be able to apply this method.

We turn now to discuss our applications over well behaved semantics. Of the major semantics proposed for logic programs with negation as failure, the well founded semantics has proved to have appealing and enduring features. It has its advantages and drawbacks. WFS is defined for a larger class of programs and admits an efficient computation, but it has been argued that WFS is by design overly careful in deciding about the falsity of some atoms, leaving them undefined. Extensions of WFS has then been proposed. Simplification of theories (program transformation in this case) has been used to characterize several

semantics as well as to define new semantics that extend WFS. This line of research was started by Dix. [1] Given a logic program P and a certain set of transformations T we apply these transformations to the program P and find a reduced program P_1, that we call the normalform of program P. Then the set of derivables literals is more simple. Of course this set of transformations must be confluent and terminating in order to guarantee a unique final program. We already have proposed some transformations rules that respect this last condition and thus we combine methods from rewriting with logic programming technology and we get a powerful framework for investigating the semantics of logic programs. Most of the well-known semantics are induced by confluent systems.

In a recent book [2] the authors (Brewka, Dix and Konolige) introduced the notion of well-behaved semantics, and aim at a classification of well-behaved semantics according to other, clearly formulated declarative properties. In particular the mentioned book presents three conjectures (conjectures 7.20, 7.21 and 7.22) stating that there are not well-behaved semantics satisfying certain properties other than the currently known semantics. Recently Osorio and Dix show that the first conjecture is false. Also Nieves and Cervantes show that the second conjecture is also false. In both cases the proofs were based on our discussed approach. We have supported reasons for expecting that the third conjecture is also false, and our current research is in this direction.

Our final application is in answer set programming. Under the stable-model semantics, a program P specifies a family of subsets of the Herbrand universe, determined by the collection of its stable models. Each of these subsets represents a possible answer to the problem encoded by P.

Dloop is a program transformation rule (for disjunctive programs) introduced recently. We have proved that the stable semantics is invariant under Dloop. Francois Fages has shown that for tight normal programs, the supported models semantics is equivalent to stable models semantics. We have generalized the result by Fages from normal programs to disjunctive programs. We define a reduction system, sys that includes Dloop and some other well known transformation rules that are correct with respect to the stable semantics. It tuns out that sometimes sys can transform a non tight program into a tight program. We illustrate how can we apply our results to compute stable models efficently. We have several open lines of research about this last issue.

References

1. Stefan Brass and Jürgen Dix. Characterizations of the Disjunctive Well-founded Semantics: Confluent Calculi and Iterated GCWA. *Journal of Automated Reasoning*, 20(1):143–165, 1998. (Extended abstract appeared in: Characterizing D-WFS: Confluence and Iterated GCWA. *Logics in Artificial Intelligence, JELIA '96*, pages 268–283, 1996. Springer, LNCS 1126.).
2. Gerd Brewka, Jürgen Dix, and Kurt Konolige. *Nonmonotonic Reasoning: An Overview.* CSLI Lecture Notes 73. CSLI Publications, Stanford, CA, 1997.

A deductive decision procedure for a restricted FTL

Regimantas Pliuškevičius

Institute of Mathematics and Informatics
Akademijos 4, Vilnius 2600, LITHUANIA,
e-mail: regis@ktl.mii.lt

Manchester Metropolitan University
Manchester M1 5GD, UK
R.Pliuskevicius@doc.mmu.ac.uk

The aim of this report is to present a new kind deductive procedure Sat for a restricted first-order linear temporal logic (FTL, in short). The proposed procedure Sat is a degenerate case of an ω-decidable-like procedure Sat_ω (see [2]) for restricted FTL. Different from ω-decidable procedure Sat_ω, the procedure Sat is decidable.

For simplicity, we assume that all the predicate symbols are flexible (i.e., change their value in time), but all the variables are rigid (i.e., with time-independent meanings). Besides, all predicate symbols have the same arity (for example all predicate symbols are 2-place only). We consider only skolemized formulas. We can consider occurrences of the "next" operator $\bigcirc$ only occuring in the formula $\bigcirc^k E$ (where E is an *elementary* formula i.e., an expression of the shape $P(t_1, \dots, t_n)$, where P is a predicate symbol, t_i is a variable or a constant). For the sake of simplicity, we "eliminate" the "next" operator and the formula $\bigcirc^k E$ is abbreviated as E^k (i.e., as an elementary formula with the index k, which is called an *atomic* formula). Let us define the objects of consideration for Sat.

Definition 1 (kernel formulas, TD-sequents, induction-free TD-sequents). *The formulas of the form $\Box\forall\bar{x}(E(\bar{x}) \supset R(\bar{x}) \wedge P^l(\bar{b}))$ is a kernel formula, if $l > 0$, $E(\bar{x})$ is an elementary formula (called the premise of the kernel formula) $R(\bar{x})$ is an elementary formula (called isolated conclusion); $P^l(\bar{b})$ is an atomic formula (called constant conclusion, C-conclusion, in short); $\bar{x} = x_1, \dots, x_n$; $\bar{b} = b_1, \dots, b_n$; b_i is a free variable or a constant, $1 \leqslant i \leqslant n$, $n \geqslant 1$.*

A sequent S is TD-sequent, if $S = \Sigma, \Pi^1, \Box\Omega \to \Box^0 A$, where $\Sigma = 0$ or consists of elementary formulas; $\Pi^1 = \varnothing$ or consists of atomic formulas of the shape E^l $(l > 0)$; Σ, Π^1 is called parametrical formulas; $\Box\Omega$ consists of kernel formulas; $\Box^0 \in \{\varnothing, \Box\}$; $A = \exists\bar{y} \bigvee_{i=1}^{m} E_i(\bar{y})$, E_i is an atomic formula. If $\Box^0 = \varnothing$ then S is induction-free TD-sequent. Each TD-sequent must satisfy the following conditions:

(1) Non-repeating condition :

if $\Box\forall\bar{x}(E_i(\bar{x}) \supset R_i(\bar{x}) \wedge P_i^l(\bar{b})) \in \Box\Omega$ and $\Box\forall\bar{y}(E_j(\bar{y}) \supset R_j(\bar{y}) \wedge P_j^k(\bar{c})) \in \Box\Omega$ then $\forall i, j$ $E_i \neq E_j$ and $P_i \neq P_j$, if $i \neq j$.

(2) Saturation condition:

(a) for each elementary formula $Q(b)$ from Σ there must be the unique kernel formula $\Box\forall x(Q(\bar{x}) \supset R(\bar{x}) \wedge P^l(\bar{b}))$ from $\Box\Omega$; (b) for each atomic formula $P^k(\bar{b})$ from Σ, Π^1 there must be unique kernel formula $\Box\forall\bar{x}(M(\bar{x}) \supset R_1(\bar{x}) \wedge P^l(\bar{b}))$ from $\Box\Omega$ and $k < l$; (c) if $P^k(\bar{b})$, $P^l(\bar{c}) \in \Sigma, \Pi^1$, then $k \neq l$ and if $P^k(\bar{b}) \in \Sigma, \Pi^1$, then $P^k(\bar{c}) \notin \Sigma, \Pi^1$; (d) if $\Sigma, \Pi^1 = P(\bar{b})$, then $\Box\Omega = \Box\forall\bar{x}(P(\bar{x}) \supset R(\bar{x}) \wedge P^l(\bar{b}))$, $l \geq 1$; (e) let $\Gamma(\Delta)$ be the set of all predicate symbols from Σ, Π, Ω (from A, correspondingly), then $\Delta \subseteq \Gamma$.

(3) Periodic condition:

$\Box\Omega = \Box\forall\bar{x}_1(E(\bar{x}_1) \supset R_1(\bar{x}_1) \wedge E_1^{l_1}(\bar{b}_1)), \Box\forall\bar{x}_2(E_1(\bar{x}_2) \supset R_2(\bar{x}_1) \wedge E_2^{l_2}(\bar{b}_2)), \dots, \Box\forall\bar{x}_n(E_{n-1}(\bar{x}_n) \supset R_n(\bar{x}_n) \wedge E_n(\bar{b}_n))$ and $E_n = E$.

So, TD-sequents do not satisfy, in general, the monodic condition from [1].

To define the separation rules $(ISIF)$ and (GIS) (see below) let us define the following operation $(+)$.

Definition 2 (operation $+$). *Let $S = \Sigma, \Pi^1, \Box\Omega \to \Box^0 A$ be a TD-sequent, and $E(\bar{b})$ be any elementary formula from Σ. Then $(E(\bar{b}))^+ := P^{n-1}(\bar{b}_i)$, where $P^n(\bar{b}_i)$ is a C-conclusion of a kernel formula $\Box\forall\bar{x}(E(\bar{x}) \supset R(\bar{x}) \wedge P^n(\bar{b}_i))$ from $\Box\Omega$.*

Let us define the infinitary calculus $G_{L\omega}$ with the help of which the proposed decision procedure is founded. Derivations in the calculus $G^*_{L\omega}$ are constructed in the bottom-up manner in the form of an infinite tree.

Definition 3 (calculi $G^*_{L\omega}$, G^*). *The calculus $G^*_{L\omega}$ is defined by the following postulates.*

The axiom $(\exists) : \Gamma, E_i(b_1, \dots, b_m) \to \exists y_1 \dots y_m \bigvee_{j=1}^{n} E_j(y_1, \dots, y_m)$ $(m \leqslant n,\ m \geqslant 0,\ 1 \leqslant i \leqslant n)$.

The rules consist of the ω-type rule:

$$\frac{\Gamma \to A; \Gamma \to A^1; \dots; \Gamma \to A^k; \dots}{\Gamma \to \Box A}\ (\to \Box_\omega)$$

and the following (loop-free) integrated separation induction-free rule:

$$\frac{(\Sigma)^+, \Pi, \Box\Omega \to B^{k-1}}{\Sigma, \Pi^1, \Box\Omega \to B^k}\ \textit{(ISIF)}, \qquad k > 0,$$

where $\Sigma = \varnothing$ or consists of elementary formulas; $\Pi^1 = \varnothing$ or consists of atomic formulas of the shape E^l $(l > 0)$; $\Box\Omega$ consists of kernel formulas; $B = \exists y_1, \dots, y_n \bigvee_{i=1}^{m} E_i(\bar{y}_i)$ $(m \leqslant n)$, where operation $(+)$ is the same as in Definition 2.

The calculus G^* is obtained from $G^*_{L\omega}$ by dropping the ω-type rule $(\to \Box_\omega)$.

Theorem 1. *(a) The calculus $G_{L\omega}$ is sound and complete for TD-sequents. (b) The calculus G^* is a decision procedure.*

Let us define the generalized integrated separation rule (GIS) which is the main tool of the proposed deductive procedure Sat and which is applied to any non-induction-free TD-sequent.

Definition 4 (generalized integrated separation rule: (GIS), successful application of (GIS)). *Let $S = \Sigma, \Pi^1, \Box\Omega \to \Box B$ be a TD-sequent. Let $(\Sigma)^+$ mean the same as in definition of (ISIF), then the generalized integrated separation rule (GIS) is as follows:*

$$\frac{\Sigma, \Pi^1, \Box\Omega \to B;\ (\Sigma)^+, \Pi, \Box\Omega \to \Box B}{\Sigma, \Pi^1, \Box\Omega \to \Box B}\ \textit{(GIS).}$$

If the left premise of (GIS), i.e., the sequent $S_1 = \Sigma, \Pi^1, \Box\Omega \to B$ is such that $G^* \vdash S_1$ we say that bottom-up application of (GIS) is successful.

Now we are going to define the basic part of Sat – the k-th resolvent (in symbols: $Re^k(S)$).

Definition 5 (similarity index, k-th resolvent: $Re^k(S)$, parametrical part of $Re^k(S)$). *Let $S = \Sigma, \Pi^1, \Box\Omega \to \Box B$ be a TD-sequent and $p_1, \dots, p_n$ indices of kernel C-conclusion formulas of S, then $p(S) = \sum_{i=1}^{n} p_i$ is similarity index of S. Let S be a TD-sequent, then the k-th resolvent of a TD-sequent S (in symbols: $Re^k(S)$) is defined in the following way: $Re^0(S) = S$. Let $Re^k(S) = S_k = \Sigma, \Pi^1, \Box\Omega \to \Box B$ then $Re^{k+1}(S)$ is defined in the following way.*

1. Let us bottom-up apply the rule (GIS) to S_k and S_{k1}, S_{k2} be the left and right premises of the application of (GIS).

2. If $G^* \nvdash S_{k1}$, then $Re^{k+1}(S) = \bot$ (false) and the calculation of $Re^{k+1}(S)$ is stopped.

3. Let $G^* \vdash S_{k1}$ (i.e., the bottom-up application of (GIS) is successful), then $Re^{k+1}(S) = S_{k2} = (\Sigma)^+, \Pi, \Box\Omega \to \Box B$; $(\Sigma)^+, \Pi$ is parametrical part of $Re^{k+1}(S)$.

4. If $Re^{k+1}(S) = S_{k2}$ and $k + 1 = p(S)$, then the calculation of $Re^{k+1}(S)$ is finished.

Analogously as in [2] we get the following

Lemma 1. *(a) Let S be a TD-sequent and all bottom-up applications of (GIS) in constructing $Re^k(S)$ are successful and $p = p(S)$ be similarity index of S. Then $Re^p(S) = S$.*

(b) The problem of calculation of $Re^p(S)$ is decidable.

Definition 6 (deductive procedure Sat, TD-sequent derivable by the help of Sat). *The deductive procedure Sat consists of decision procedure $Re^k(S)$. TD-sequent S is derivable by the help of Sat (in symbols: $Sat \vdash S$) if $Re^p(S) = S$, $p = p(S)$, $p(S)$ is the similarity index of S.*

Now let us introduce the "invariant calculus" IN.

Definition 7 (invariant calculus IN). *The calculus IN is obtained from the calculus $G^*_{L\omega}$ in the following way: (1) adding the logical rules $(\to \wedge)$, $(\wedge \to)$, $(\vee \to)$, $(\to \vee)$; (2) adding the axiom $\Gamma, \Box A \to \Box A^1$; (3) replacing the rule $(\to \Box_\omega)$ by the following rule*

$$\frac{\Gamma \to R;\ R \to R^1;\ R \to A}{\Sigma, \Pi^1, \Box\Omega \to \Box A}\ (\to \Box),$$

where $\Gamma = \Sigma, \Pi^1, \Box\Omega$ and invariant formula has the following shape $R = \bigvee_{i=1}^{n} \Gamma_i^{\wedge} \wedge \Box\Omega$, where $\Box\Omega$ is the kernel of the given TD-sequent $S = \Sigma, \Pi^1, \Box\Omega \to \Box A$; Γ_k is the parametrical part of k-th $Re^k(S)$, $n = p(S)$, i.e., the similarity index of S; $\Gamma_i^{\wedge}$ is the conjunction formulas from Γ_i.

Theorem 2. *Let S be TD-sequent then $G_{L\omega} \vdash S \iff Sat \vdash S \iff IN \vdash S$.*

From Theorems 1, 2 and Lemma 2(b) we get

Theorem 3. *The calculi Sat and IN are sound and complete and decidable for TD-sequents.*

Example 1. Let $S = E(b_2), \Box\Omega \to \Box\exists y(E(y) \vee P^1(y) \vee P(y))$, where $\Box\Omega = \Box\forall x(E(x) \supset R_1(x) \wedge P^2(b_1))$, $\Box\forall y(P(y) \supset R_2(y) \wedge E^1(b_2))$. Then the similarity index $p(S) = 2 + 1 = 3$. It is easy to verify that each calculation of $Re^k(S)$ is successful. Therefore the calculation $Re^k(S)$ stops when $k = 3$ and $Re^3(S) = S$. Hence $Sat \vdash S$. It is easy to verify that $Re^1(S) = P^1(b_1), \Box\Omega \to \Box A$; $Re^2(S) = P(b_1), \Box\Omega \to \Box A$; $Re^3(S) = E(b_2)$, $\Box\Omega \to \Box A$. Therefore $R = (P^1(b_1) \vee P(b_1) \vee E(b_2)) \wedge \Box\Omega$. It is easy to verify that $IN \vdash S$.

References

1. Hodkinson I., Wolter F., Zakharyaschev M.: Decidable fragments of first-order temporal logics. (To appear in: *Annals of Pure and Applied Logic*).
2. Pliuškevičius R.: On an ω-decidable deductive procedure for non-Horn sequents of a restricted FTL. (To appear in: *Proceedings of First International Conference on Computational Logic*).

Run-time optimisations for reasoning with intensional logics

Allan Ramsay
Dept of Language Engineering, UMIST, PO Box 88, Manchester M60 1QD, UK
Allan.Ramsay@umist.ac.uk

Abstract. Most optimisation techniques for theorem provers for first-order logic rely on static analysis of the problem statement. For intensional logics, such as static analysis cannot be relied on, since it is impossible to predict what literals may be introduced by the intensional rules. The current paper shows how to use a dynamic (run-time) version of one well-known static optimisation, and considers its relationship to the use of 'relevance checking' in Satchmo.

1 A constructive intensional logic

We have shown elsewhere [8, 3] how to extend [6]'s theorem prover Satchmo to cope with [9]'s property theory. Property theory is a highly intensional logic which, roughly speaking, allows you to perform unconstrained quantification over propositions and properties, but places constraints on the conditions under which the Tarski biconditional

$$(\lambda x P).t \leftrightarrow P_{t/x}$$

holds. This language has numerous potential applications: I use it primarily for reasoning about natural language semantics, because that's what I'm interested in, but [10] uses a closely related language for reasoning about programs, and it can also be used as the basis of a theory of knowledge and belief which does not fall foul of the problems associated with modal treatments of these topics (logical omniscience, logical blindness).

Property theory is, clearly, highly intractable in theory – it is undecidable, rather than just semi-decidable, and incomplete (in the same sense that recursive arithmetic is incomplete, and for the same reasons). In the domain where I want to apply it, however, the problems that arise tend not be all that hard: it may be true that natural language semantics can only be captured properly if you use some language of this kind (as, for instance, situation semantics [2] uses non-well-founded set theory [1]), but most of the time you do not have to solve pathological problems in order to understand what some has said (unless, of course, they say *'I am now lying'*).

In ([8]), I showed that you could adapt Satchmo to work with property theory roughly as follows:

(i) rules with unspecified formulae in their antecedents are used only when those antecedents are ground. This means that a rule like $\forall X \forall P((believe(X,P) \& P) \rightarrow know(X,P)$[1] is turned into `know(X, P) :- believe(X, P), nonvar(P), P`.

(ii) rules with unspecified formulae in their consequents are used in the forward-chaining part of the algorithm, as though they were an extreme case of formulae with disjunctive consequents. As an example, the rule $\forall X \forall P(know(X,P) \rightarrow P$ turns into `know(X, P), nonvar(P) ==> P`, to be used by the Satchmo rule that asserts a component of the consequent of any sequent where the antecedent is currently provable but the consequent is not. The `nonvar(P)` test here is strictly redundant, since any sequent which failed to ground the consequent would be necessarily inconsistent (since it would entail *everything*, including $\perp$).

[1] I am not proposing this as a serious characterisation of the relation between knowledge and belief, just using it to illustrate the approach.

2 Optimisations

The problem with this inference engine is that although it is obtained by extending a fairly simple first-order theorem prover, it is not possible to use any optimisations which rely on a static analysis of the problem. You cannot, for instance, use [4]'s 'pure literal deletion' (where you remove a clause if it contains a positive (negative) instance of a literal which has no negative(positive) occurrences), because you do not know that the literal you are considering will remain pure. I propose a dynamic version of pure literal deletion, which makes sequents containing pure literals *temporarily* unavailable. At the point in Satchmo where a split sequent has led to the introduction of a new fact, clauses which might be *im*purified by the new fact are inspected, and reinstated if appropriate (this can lead to a cascade of reinstatements, in exact contrast to the way that pure literal deletion itself can lead to gangrene).

There are two ways to implement dynamic pure literal deletion of this kind. The weak version involves looking for antecedent literals that fail to appear in the consequent of any Horn sequent, the strong one deals with antecedent literals that fail to appear in the consequent of any sequent at all. It turns out that you cannot use the strong version with the relevance checking algorithms described in ([7, 5]), but for for the problems we are concerned with the strong version turns out to be more effective than the weak one + relevance checking in any case – Fig. 1 shows a typical set of results for a reasonably complex example (times in seconds).

	+groundedness	-groundedness
Unoptimised	1.62	1.98
Strong purification	0.38	0.42
Relevance checking	1.55	2.04
Weak purification	0.54	0.59
Relevance + weak pur.	0.52	0.59

Fig. 1. Relative effects of optimisations

References

1. P Aczel and R Lunnon. Universes and parameters. In J Barwise, J M Gawron, G Plotkin, and S Tutiya, editors, *Situation Theory and its Applications II.* CSLI Publications, 1991.
2. J Barwise and J Perry. *Situations and Attitudes.* Bradford Books, Cambridge, MA, 1983.
3. M Cryan and A M Ramsay. A normal form for Property Theory. In *Fourteenth Conference on Automated Deduction*, Townsville, Australia, 1997.
4. R Kowalski. A proof procedure using connection graphs. *JACM*, 22(4):572–595, 1975.
5. D W Loveland. Near-horn Prolog and beyond. *Journal of Automated Reasoning*, 7:1–26, 1991.
6. R Manthey and F Bry. Satchmo: a theorem prover in Prolog. In *Proc. 9th Inter. Conf. on Automated Deduction (CADE-9), LNAI 310*, pages 415–434, 1988.
7. A M Ramsay. Generating relevant models. *Journal of Automated Reasoning*, 7:359–368, 1991.
8. A M Ramsay. A theorem prover for an intensional logic. *Journal of Automated Reasoning*, 14:237–255, 1995.
9. R Turner. A theory of properties. *Journal of Symbolic Logic*, 52(2):455–472, 1987.
10. R Turner. *Constructive Foundations for Functional Languages.* McGraw Hill, London, 1991.

System description: Vampire 1.0

Alexandre Riazanov and Andrei Voronkov

University of Manchester
{riazanov,voronkov}@cs.man.ac.uk

1 General description

Vampire is an automatic theorem prover for first-order classical logic. It implements the calculi of ordered binary resolution, hyperresolution and superposition for handling equality. The splitting rule is simulated by introducing new predicate symbols. The search process is based on a saturation algorithm similar to the one of Otter [4], enhanced with the Limited Resource Strategy briefly described in Section 3. For pruning the search space we exploit a number of standard redundancy criteria and simplification techniques: subsumption, tautology deletion, subsumption resolution and rewriting by ordered unit equalities. The only term ordering used in Vampire at the moment is a special version of the Knuth-Bendix ordering which allows efficient approximation algorithms for solving ordering constraints. One of the main goals of the project is creating an experimental environment for developing and evaluating efficient implementation techniques for first-order theorem proving.

2 Term indexing in Vampire

Even in the presence of good redundancy criteria, the main operations in a typical resolution-based theorem prover — unification, matching, subsumption — involve searching in huge sets of terms and clauses. The standard way of solving this problem is to maintain special datastructures for *term indexing* (see [2], [5]). In Vampire we use basically three kinds of indexes. A version of *discrimination trees* [3] is used for the unification indexes. Code trees [7] are used for forward subsumption and matching. A novel indexing technique based on path indexing [6] and database style joins is used for backward subsumption. Our main approach to designing efficient indexing schemes is based on the notion of *compilation.* To perform retrieval from an index efficiently we specialize the retrieval algorithm for every particular query clause or term in order to utilise specific properties of the query. For example, compilation for indexed unification makes use of information on occurences of variables and ground subterms in the query in order to identify redundant occurence checks. The specialized version is represented as a code for an abstract machine and then interpreted. Moreover, in our indexing for forward subsumption we also compile the indexed objects and combine their codes into a structure called a *code tree.*

3 Other features

In order to solve complex real-life problems in limited time, the completeness of search procedures used in provers is often compromised for the sake of efficiency.

The limited resource strategy implemented in our system is intended to do this in an intelligent way. Vampire tries to estimate what generated clauses can be processed by the given time limit. This estimation makes use of statistics on the speed of processing retained clauses of different complexity and establishes a limit on the complexity of retained clauses. Generated clauses whose complexity exceeds this limit are discarded.

Another distinguishing feature of Vampire is an efficient and flexible implementation of splitting (see for example [1]). When a clause can be split into the components $C_1, \ldots, C_n$ with disjoint sets of variables, we introduce n new propositional variables $p_1, \ldots, p_n$, that can be regarded as names for the components, and replace the clause by $n+1$ new clauses $C_1 \vee p_1 \ \ , \ldots, \ \ C_n \vee p_n$, $\neg p_1 \vee \ldots \vee \neg p_n$. The literals p_i in the new clauses C_i, p_i are made minimal in order to ensure that they are not selected before all the other literals are cut by resolution. A special index is maintained to avoid giving different names to components that are variants of each other.

4 Work in progress

At the moment we are interested in finding optimisations for our term indexing techniques which could allow us to treat efficiently symbols with special properties such as commutativity of functions and symmetry of predicates. Also, some attempts are being made to integrate checking of ordering constraints into the indexing structures.

5 Implementation and availability

The system is implemented in C++ and can be compiled by 2.91 or newer versions of gcc. Currently it runs on Solaris, Linux and Free BSD. It is available from the authors free of charge, for details check
`http://www.cs.man.ac.uk/~riazanov/Vampire/`.

References

1. Weidenbach C. Spass: combining superposition, sorts and splitting. In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning.* Elsevier Science and MIT Press, 2000. To appear.
2. P. Graf. *Term Indexing*, volume 1053 of *Lecture Notes in Computer Science.* Springer Verlag, 1996.
3. William W. McCune. Experiments with discrimination-tree indexing and path indexing for term retrieval. *Journal of Automated Reasoning*, 9(2):147–167, 1992.
4. W.W. McCune. OTTER 3.0 reference manual and guide. Technical Report ANL-94/6, Argonne National Laboratory, January 1994.
5. I.V. Ramakrishnan, R. Sekar, and A. Voronkov. Term indexing. In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning.* Elsevier Science and MIT Press, 2000. To appear.
6. M. Stickel. The path indexing method for indexing terms. Technical Report 473, Artificial Intelligence Center, SRI International, Menlo Park, CA, October 1989.
7. A. Voronkov. The anatomy of Vampire: Implementing bottom-up procedures with code trees. *Journal of Automated Reasoning*, 15(2):237–265, 1995.

A decision procedure for term algebras with queues

Tatiana Rybina[*] Andrei Voronkov[‡]

July 14, 2000

Abstract

In software verification it is often required to prove statements about heterogeneous domains containing elements of various sorts, such as counters, stacks, lists, trees and queues. Any domain with counters, stacks, lists, and trees (but not queues) can be easily seen a special case of the term algebra, and hence a decision procedure for term algebras can be applied to decide the first-order theory of such a domain.

We prove that the first-order theory of term algebras extended with queues is decidable by presenting a quantifier-elimination procedure for this theory.

Term algebras with queues

Sorts. We assume a finite set of *basic sort.* We define *sort* as follows: (i) every basic sort is a sort; (ii) if α is a sort, then $queue(\alpha)$ is a sort, called a *queue sort.*

We call a *function type* any expression of the form $\alpha_1 \times \ldots \times \alpha_n \rightarrow \beta$, where $\alpha_1, \ldots, \alpha_n, \beta$ are sorts, $n \geq 0$. When $n = 0$, we will write β instead of $\rightarrow \beta$. A *signature* $\mathcal{S}$ is a pair consisting of a finite set of function symbols, and a function : mapping every function symbol f to a function type $\alpha_1 \times \ldots \times \alpha_n \rightarrow \beta$, called the *type* of f, where β is a basic sort. The number n is called the *arity* of f. Function symbols of arity 0 are called *constants.*

Structure $\mathcal{Q}(\mathcal{S})$. Every signature $\mathcal{S}$ defines a unique structure $\mathcal{Q}(\mathcal{S})$, called a *term algebra with queues* as follows.

The *universe* of $\mathcal{Q}(\mathcal{S})$ is defined inductively as follows.

1. For every constant $a : \beta$, a is an element of $\mathcal{Q}(\mathcal{S})$ of the sort β.

2. For every elements $e_1, \ldots, e_n$ of the same sort α, where $n \geq 0$, the sequence $e_1 \ldots e_n$ is an element of $\mathcal{Q}(\mathcal{S})$ of the sort $queue(\alpha)$. Elements of any sort $queue(\alpha)$ will be called *queues.* The empty sequence is called the *empty queue*, and denoted ε_α.

3. If $f : \alpha_1 \times \ldots \times \alpha_n \rightarrow \beta$ and $e_1, \ldots, e_n$ are elements of $\mathcal{Q}(\mathcal{S})$ of the sorts $\alpha_1, \ldots, \alpha_n$, respectively, then $f(e_1, \ldots, e_n)$ is an element of $\mathcal{Q}(\mathcal{S})$ of the sort β.

We consider two elements equal only if they have the same sort and coincide as expressions. Using different signatures we obtain different domains, in which sorts can be finite or infinite. For simplicity, we assume that each basic sort has a nonempty domain.

[*]Partially supported by a grant from EPSRC (Rainbow project).

[†]Partially supported by grants from EPSRC and the Faculty of Science and Engineering.

[‡]This abstract is based on [Rybina and Voronkov 2000].

The *language of* $\mathcal{Q}(\mathcal{S})$, denoted $\mathcal{L}_{\mathcal{Q}(\mathcal{S})}$, is a first-order language using, in addition to function symbols in $\mathcal{S}$, the equality relation symbol $=$ and the following symbols: (i) for every sort α, the constant $\varepsilon_\alpha : queue(\alpha)$; (ii) for every sort α, the function symbols $ladd_\alpha : \alpha \times queue(\alpha) \to queue(\alpha)$ and $radd_\alpha : \alpha \times queue(\alpha) \to queue(\alpha)$.

Semantics of $\mathcal{Q}(\mathcal{S})$. For every function symbol $f : \alpha_1 \times \ldots \times \alpha_n \to \beta$ of $\mathcal{S}$, the interpretation $\underline{f}$ of this symbol in $\mathcal{Q}(\mathcal{S})$ is defined as follows: for all elements $e_1, \ldots, e_n$ of the sorts $\alpha_1, \ldots, \alpha_n$ we have $\underline{f}(e_1, \ldots, e_n) = f(e_1, \ldots, e_n)$. Likewise, the interpretation $\underline{\varepsilon_\alpha}$ of the constant ε_α is the empty queue of the sort $queue(\alpha)$. The function symbols $\underline{ladd_\alpha}$ and $\underline{radd_\alpha}$ of the function symbols $ladd_\alpha$ and $radd_\alpha$ are interpreted as the left- and right- addition of an element to a queue.

Quantifier elimination

Similar to Kunen [1987] and Belegradek [1988], we extend the language $\mathcal{L}_{\mathcal{Q}(\mathcal{S})}$ to a new language, denoted $\mathcal{L'}_{\mathcal{Q}(\mathcal{S})}$, by introducing the following collection of decomposition function symbols and relation symbols.

1. A unary relation symbol $\mathbf{Is}_f : \beta$ for each function symbol $f : \alpha_1 \times \ldots \times \alpha_n \to \beta$, $n \geq 1$ of $\mathcal{S}$. This symbols is interpreted as follows: $\mathbf{Is}_f(a)$ is true if and only if a has the form $f(\ldots)$.
2. A unary function symbol $f_i : \beta \to \alpha_i$, for each function symbol $f : \alpha_1 \times \ldots \times \alpha_n \to \beta$, $n \geq 1$ of $\mathcal{S}$ and each $i = 1, \ldots, n$. We have $f_i(f(t_1, \ldots, t_n)) = t_i$.
3. unary function symbols

$$\begin{array}{ll} lhead_\alpha : queue(\alpha) \to \alpha & rhead_\alpha : queue(\alpha) \to \alpha \\ ltail_\alpha : queue(\alpha) \to queue(\alpha) & rtail_\alpha : queue(\alpha) \to queue(\alpha) \end{array}$$

   for each sort α. For every queue q, $lhead(q)$ gives the leftmost element of q, and $ltail(q)$ is the queue obtained by removing this element, and similar for *rhead* and *rtail*.

Theorem 1 (Quantifier Elimination) *The first-order theory of term algebras with queues in the language* $\mathcal{L'}_{\mathcal{Q}(\mathcal{S})}$ *admits quantifier elimination.*

Corollary 2 (Decidability) *The first-order theory of any term algebra with queues is decidable.*

Corollary 3 (Words) *The first-order theory of words with the operations of the left and right multiplications by a letter is decidable.*

The monadic second-order theory of this structure is known to be undecidable.

References

Belegradek O. [1988], Model theory of locally free algebras (in Russian), *in* 'Model Theory and its Applications', Vol. 8 of *Trudy Instituta Matematiki*, Nauka, Novosibirsk, pp. 3–24.

Kunen K. [1987], 'Negation in logic programming', *Journal of Logic Programming* **4**, 289–308.

Rybina T. and Voronkov A. [2000], A decision procedure for term algebras with queues, *in* 'Proc. 15th Annual IEEE Symp. on Logic in Computer Science', Santa Barbara, California, pp. 279–290.

Description Logics and Knowledge Discovery of Data

Stefan Schlobach[*]

Introduction. Description Logics (DL) are knowledge representation formalisms which have been applied in a number of application areas over the last decade. Knowledge Discovery on the other hand is becoming more and more popular given the enormous amount of available data and has considerably increased the interest in machine learning techniques for data mining purposes. We have tried to make this machinery available to DL reasoning systems, which enables easy integration of predefined knowledge and data for the Knowledge Discovery process. The core of this new approach, i.e. the formal definition of a learning paradigm in Description Logics and a general framework to calculate *learned concepts* was presented in [Sch00].

Knowledge Discovery from Data. Fayyad et al. define Knowledge Discovery in Databases (KDD) as *the nontrivial process of identifying valid, novel, potentially useful, and ultimately understandable patterns in data [FPSS96].* The KDD process consists of three steps. First, data is preprocessed, i.e., interesting and typical (and hopefully crisp and un-noisy) data is chosen. The second step is the data mining process. Data Mining means the application of data analysis and learning algorithms to produce enumerations of patterns over the data. Finally the learned results have to be evaluated and turned into a useful (e.g. human-readable) form.

Description Logics. Description Logics (DL) are set description languages that have been in use for knowledge representation for more than a decade. The hybrid character of reasoning about concepts and about individuals is nicely matched in the hybrid architecture of DL-systems which usually incorporate separate mechanisms for reasoning with extensional and intensional knowledge. The general ideas of A–Box mining are independent of a particular description logic. For this reason we define the set of languages $\mathcal{DL}$ to which our approach is applicable as the set of description logics containing at least conjunction, atomic negation and universal quantification over roles where subsumption, consistency and instance checking is decidable.

Learning and Mining in Description Logics. The main objective of A–Box mining is to generate concepts from a given A–Box $\mathcal{A}$ that can replace the decisions in the D–Box for classification. We will introduce and define two necessary criteria (called covering and exclusiveness) which define our notion of concepts *learned* from $\mathcal{A}$ and $\mathcal{D}$ (and possibly $\mathcal{T}$). These concepts will be called *generalised decision concepts* (GDCs).

The first condition states that all objects in the A–Box that are instances of a GDC learned from a decision D must be instances of D, i.e., all A–Box instances a of a GDC $\underline{\mathcal{A}}D_i$ must be covered by the corresponding decision D_i: i.e. $a \in_{\mathcal{A}} \underline{\mathcal{A}}D_i \Rightarrow a \in_{\mathcal{A}} D_i$. This condition ensures correctness of the classification through the GDC with respect to the original decision. The second condition denotes the fact that any A–Box object a, which is an instance of $\underline{\mathcal{A}}D_i$, must by unambiguously decidable with the corresponding original decision D_i only: $a \in_{\mathcal{A}} \underline{\mathcal{A}}D_i \Rightarrow a \notin_{\mathcal{A}} D_j$. A *generalised decision concept* (GDC) for a decision D is a concept, which is exclusively $\mathcal{A}$-covered by D.

Simplicity and Optimality. The definition for GDCs gives necessary but not sufficient criteria to justify the quality of learned concepts. Depending on the chosen language, there might be infinitely many GDCs for a decision. There are two further criteria which we introduce informally, namely that GDCs should be "as general as possible" (semantic optimality) and "as simple as possible" (syntactic simplicity).

[*]Dept. of Computer Science, King's College London, Strand, London WC2R 2LS, UK, schlobac@dcs.kcl.ac.uk

Description Logic based KDD Systems. We present a framework for Knowledge Discovery for Description Logics $\mathcal{DL}$ which was investigated in [Sch00] in more detail.

Input:	An A–Box $\mathcal{A}$, T–Box $\mathcal{T}$ and a set of decisions $\{D_1, \ldots, D_n\}$.
Output:	The Set of all evaluated generalised decision concepts for each decision.

- **Preprocessing:** There are three preprocessing steps to be performed that are useful independent of a particular learning or mining method. *A–Box classification* transforms the assertional information about data into related logical concepts. By creating hierarchies for the logical descriptions, *T–Box classification* allows to deal with sparse A–Boxes and missing data. *Separation* of concepts into approximations is a natural mechanism to deal with rough and noisy information.

- **Generalisation:** There are different ways to learn concepts from examples and counterexamples, but mostly a combination of concept expansion and inconsistency check is involved. *Expansion* is the process of choosing superconcepts for relevant logical descriptions, which are inconsistent with logical descriptions (or expansions) of the counterexamples. This part of the framework is in general non constructive.

- **Optimality:** For each particular description logic optimality has to be defined separately. For some languages (e.g. $\mathcal{AL}^{\mathbb{R},=}$) the notions of optimality and simplicity coincide, and sound and complete mechanisms to calculate optimal GDCs can be defined. For other languages and learning methods, the quality of notions for "general and simple" will be more difficult to assess and heuristics have to be incorporated into the learning process.

- **Evaluation of Results:** The evaluation of the results is an integral parts of the KDD process If generalised decision concepts are to become part of the intensional knowledge of the knowledge representation system, they have to be assessed using quantitative accuracy measures and by human experts. Classically, learning results are difficult to read and understand, so that it must be one of the features of a knowledge discovery system to transform the results into a human-readable form. One possible approach is described in [BKM] where Baader et al. investigate algorithms for rewriting concepts using terminologies for several logics in $\mathcal{DL}$.

Implementation. The Group of Logic and Computation at King's College London is currently implementing a family of hybrid systems[1] which enables data driven logic reasoning. Implementations of specialised and efficient algorithms for $\mathcal{AL}^{\mathbb{R}}$, $\mathcal{ALC}$ with concrete domains and $DL_{arc\forall}$ [Ohl99], which allows for an expressive T–Box with a strong arithmetical component and a database–like assertional component are currently under way. The integration of a knowledge discovery facility will be an integral part of the system.

References

[BKM] F. Baader, R. Küster, and R. Molitor. Rewriting concepts using terminologies. Technical Report LTCS-Report 99-12, LuFG Theoretical Computer Science, RWTH Aachen, Germany.

[FPSS96] U. Fayyad, G. Piatetsky-Shapiro, and P. Smyth. Knowledge discovery and data mining toward a unifying framework. In *Proceeding of The Second Int. Conference on Knowledge Discovery and Data Mining*, pages 82–88, 1996.

[Ohl99] H.J. Ohlbach. A theory-resolution style A–Box calculus. In *M4M Methods for Modalities, Workshop Proceedings*. University of Amsterdam, 1999.

[Sch00] S. Schlobach. Assertional mining in description logics. 2000 International Workshop on Description Logics - DL2000, August 2000.

[1] More information about the `Wellington` knowledge representation and reasoning system and its implementation is available from the author or on: `http://www.dcs.kcl.ac.uk/research/groups/logic/wellington`.

Deciding Fluted Logic with Resolution

Renate A. Schmidt

Department of Computer Science, University of Manchester
Manchester M13 9PL, United Kingdom, schmidt@cs.man.ac.uk

Fluted logic is a solvable fragment of first-order logic which arose as a by-product of Quine's predicate functor logic [13, 14]. It is defined as follows. Let $X_i = \{x_1, \ldots, x_i\}$ denote an ordered set of variables. By definition, an *atomic fluted formula* over X_i is an n-ary atom $P(x_l, \ldots, x_i)$, with $l = i - n + 1$ and $n \leq i$. Now, define *fluted formulae* by: (i) any atomic fluted formula over X_i is a fluted formula over X_i, (ii) if φ is a fluted formula over X_{i+1}, then $\exists x_{i+1}\varphi$ and $\forall x_{i+1}\varphi$ are fluted formulae over X_i, and (iii) any Boolean combination of fluted formulae over X_i is a fluted formula over X_i.

The way decidability is obtained in fluted logic is an interesting contrast to other solvable first-order fragments which are more well-known. Fragments considered until the sixties usually involve some form of restriction on quantification. In prefix classes such as the Bernays-Schönfinkel class, and the initially extended Ackermann class, the initially extended Gödel class the quantifier prefixes are restricted, to $\exists^*\forall^*$, $\exists^*\forall\exists^*$ and $\exists^*\forall\forall\exists^*$. In Maslov's class K (more precisely, in the dual class $\overline{K}$) there is a restriction on universal quantification. In the guarded and loosely guarded fragments, which were introduced more recently, quantifiers are restricted to conditional quantifiers of the form $\exists\overline{y}G(\overline{x},\overline{y}) \land \varphi$ or $\forall\overline{y}G(\overline{x},\overline{y}) \to \varphi$, where $G(\overline{x},\overline{y})$ is a guard formula satisfying certain restrictions ($G(\overline{x},\overline{y})$ is an atom in the case of the guarded fragment). Other decidable classes such as the monadic class and FO^2 are defined over predicate symbols with bounded arity. By contrast, in the case of fluted logic decidability is obtained by imposing an ordering on variables and arguments. To our knowledge, of the mentioned logics, fluted logic has thus far not been studied in the context of resolution. For all other logics mentioned above resolution-based decision procedures have been proposed, in some cases even several different refinements exist [1, 2, 5, 9].

Fluted logic is also of interest for its relationship to non-classical logics. Fluted logic may be viewed as a generalisation of propositional modal logic, just as the guarded fragments can. The properties fluted logic is known to share with modal logics are decidability and the finite model property [10–12]. From a modal perspective an advantage of fluted logic over the guarded fragment is that relational atoms may be negated. This means that extended modal logics such as Boolean modal logic [3] and other enriched modal logics, as well as expressive description logics like $\mathcal{ALB}$ (without converse) [8], which cannot be embedded in the guarded fragment, can be embedded in fluted logic. Interestingly, translations of propositional modal formulae by both the relational translation and a variation of the functional translation (described and used in [4, 6]) are fluted formulae.

In [15] we characterise fluted logic by a new class of clauses, called the class of *fluted clauses*. We present a decision procedure for this class which is based on an ordering refinement of resolution and an additional *separation rule*. This is a new inference rule which does dynamic renaming. It replaces a clause $C \lor D$ by two clauses which involve a newly introduced literal. Formally:

$$\frac{N \cup \{C \vee D\}}{N \cup \{\neg q(x_1,\dots,x_n) \vee C,\ q(x_1,\dots,x_n) \vee D\}}$$

provided (N denotes a set of clauses) (i) the clause $C \vee D$ is separable into clauses C and D, that is, $\mathrm{var}(C) \not\subseteq \mathrm{var}(D)$ and $\mathrm{var}(D) \not\subseteq \mathrm{var}(C)$, (ii) $\mathrm{var}(C) \cap \mathrm{var}(D) = \{x_1,\dots,x_n\}$ for $n \geq 0$, and (iii) q does not occur in N, C or D. The rule is sound, in general, and resolution extended by this rule remains complete, if it is not applied infinitely often. Separation is essential for our decision procedure, since it allows us to transform certain problematic fluted clauses into so-called *strongly fluted clauses.* A strongly fluted clause is a fluted clause that contains a literals which includes all the variables of the clause. When inference is restricted to such literals (i) the number of variables in any derivable clause is finitely bounded, in particular, the number of variables does not exceed the number of variables in the original clause set. To show termination, it is usually sufficient [9] to show in addition that (ii) there is a bound on the depth of terms occurring in derived clauses. Because separation introduces new predicate names during the derivation, in our case we also need to show that (iii) there is a bound on the number of applications of the separation rule. Exhibiting (ii) and (iii), along with verifying the deductive closure of the class of (strongly) fluted clauses are the most difficult parts of the termination proof. The difficulty can be attributed to the fact that the depth of terms can grow during the derivation, as is the case for some other solvable clausal classes, for example, those associated with Maslov's dual class $\overline{K}$ [1, 7].

References

1. C. Fermüller, A. Leitsch, T. Tammet, and N. Zamov. *Resolution Method for the Decision Problem*, vol. 679 of *LNCS*. Springer, 1993.
2. C. G. Fermüller, A. Leitsch, U. Hustadt, and T. Tammet. Resolution theorem proving. In *Handbook of Automated Reasoning*. Elsevier, 2000. To appear.
3. G. Gargov and S. Passy. A note on Boolean modal logic. In P. P. Petkov, editor, *Mathematical Logic: Proceedings of the 1988 Heyting Summerschool*, pp. 299–309. Plenum Press, 1990.
4. A. Herzig. A new decidable fragment of first order logic, 1990. In Abstracts of the Third Logical Biennial, Summer School & Conference in Honour of S. C. Kleene, Varna, Bulgaria.
5. U. Hustadt. *Resolution-Based Decision Procedures for Subclasses of First-Order Logic.* PhD thesis, Univ. d. Saarlandes, Saarbrücken, Germany, 1999.
6. U. Hustadt and R. A. Schmidt. An empirical analysis of modal theorem provers. *J. Appl. Non-Classical Logics*, 9(4), 1999.
7. U. Hustadt and R. A. Schmidt. Maslov's class K revisited. In H. Ganzinger, editor, *Automated Deduction—CADE-16*, vol. 1632 of *LNAI*, pp. 172–186. Springer, 1999.
8. U. Hustadt and R. A. Schmidt. Issues of decidability for description logics in the framework of resolution. In *Automated Deduction in Classical and Non-Classical Logics*, vol. 1761 of *LNAI*, pp. 192–206. Springer, 2000.
9. W. H. Joyner Jr. Resolution strategies as decision procedures. *J. ACM*, 23(3):398–417, 1976.
10. W. C. Purdy. Decidability of fluted logic with identity. *Notre Dame J. Formal Logic*, 37(1):84–104, 1996.
11. W. C. Purdy. Fluted formulas and the limits of decidability. *J. Symbolic Logic*, 61(2):608–620, 1996.
12. W. C. Purdy. Quine's 'limits of decision'. *J. Symbolic Logic*, 64(4):1439–1466, 1999.
13. W. V. Quine. Variables explained away. In *Proc. Amer. Philos. Soc.*, vol. 104, pp. 343–347, 1960.
14. W. V. Quine. Algebraic logic and predicate functors. In R. Rudner and I. Scheffler, editors, *Logic and Art: Esssays in Honor of Nelson Goodman.* Bobbs-Merrill, Indianapolis, 1971.
15. R. A. Schmidt and U. Hustadt. A resolution decision procedure for fluted logic. In *Automated Deduction—CADE-17*, vol. 1831 of *LNAI*, pp. 433–448. Springer, 2000.