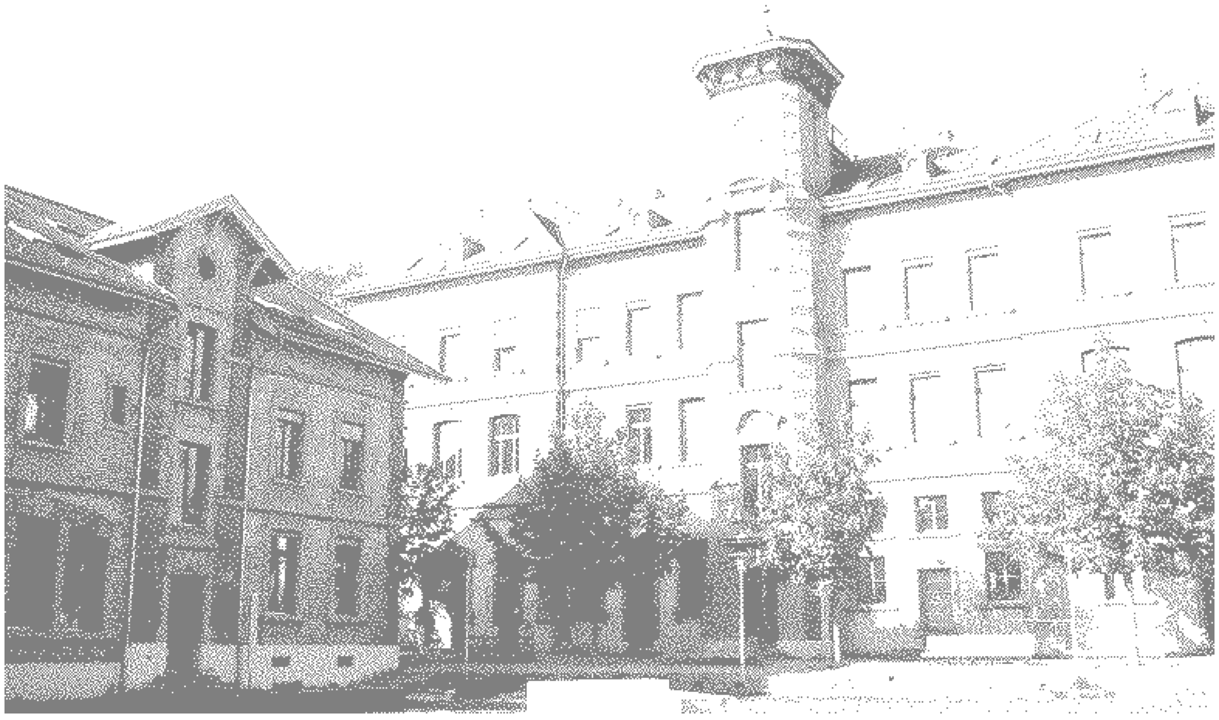
b
UNIVERSITÄT
BERN

TABLEAUX 2011
Workshops, Tutorials, and Short Papers

Martin Giese and Roman Kuznets (editors)

Technical Report IAM-11-002, 4–8 July 2011

Institut für Informatik und angewandte Mathematik, www.iam.unibe.ch



Contents

Part I FTP 2011—International Workshop on First-Order Theorem Proving

Preface 2
Committees and Organizers 3
Table of Contents 5
Author Index 66

Part II Gentzen Systems and Beyond 2011, International Workshop

Preface 68
Nested Sequents and Prefixed Tableaux
Melvin Fitting 69
Constructive Realization in Justification Logics via Nested Sequents
Remo Goetschi 70
Definite and Indefinite Descriptions
Norbert Gratzl 71
A Tentative Atomic Calculus for Natural Deduction
Tom Gundersen and Michel Parigot 72
A Symmetric Natural Deduction
Michel Parigot 73
Sequent Calculus for Justifications
Yury Savateev 74
Some Remarks on Nested Sequent Systems for Modal Logics
Lutz Straßburger 75
On the Correspondence between Display Postulates and Deep Inference in Nested Sequent Calculi for Tense Logics
Alwen Tiu 76

Part III Tutorials

Tableaux(-like) Methods for the Satisfiability Problems of Temporal Logics
Martin Lange 78

Introduction to Proof Nets
Lutz Straßburger 79

Part IV Short Papers

Dialogue Games for Classical Logic
Jesse Alama, Aleks Knoks, and Sara L. Uckelman 82
A Dynamic Programming Algorithm for Prime Implicates
Andrew Matusiewicz 87

Part I
FTP 2011—International Workshop on First-Order Theorem Proving

Preface

This booklet contains the research papers presented at the International Workshop on First-Order Theorem Proving held in Bern, Switzerland, July 4–5, 2011. The workshop was the eighth in a series of international workshops held since 1997, intended to focus effort on First-Order Theorem Proving as a core theme of Automated Deduction, and to provide a forum for presentation of recent work and discussion of research in progress. Previous editions of FTP took place in Schloss Hagenberg, Austria (1997); Vienna, Austria (1998); St Andrews, Scotland (2000); Valencia, Spain (2003); Koblenz, Germany (2005); Liverpool, UK (2007); and Oslo, Norway (2009).

FTP 2011 was held together with the 20th International Conference on Automated Reasoning with Analytic Tableaux and Related Methods (Tableaux 2011). On July 5, 2011, there was a joint session with Tableaux 2011, with Maria Paola Bonacina as joint invited speaker.

The technical program of FTP 2011 consisted of two invited talks, one by Felix Klaedtke on "Monitoring First-order Temporal Properties," and one by Maria Paola Bonacina "On interpolation in decision procedures" (joint with Tableaux 2011), three regular papers, one system description, and two presentation-only contributions. In addition to the regular papers and the system description, this booklet includes short abstracts of the presentation-only contributions and the invited talk by Felix Klaedtke. The paper belonging to the invited talk of Maria Paola Bonacina is included in the Tableaux 2011 proceedings (LNCS vol. 6793, Springer Verlag)

We wish to sincerely thank all the authors who submitted their work for consideration. And we would like to thank the Program Committee members and other referees for their great effort and professional work in the review and selection process. Their names are listed on the following pages.

We are also particularly grateful to the organisers of Tableaux 2011 for both the practical organisation and their help in attracting funding from the Swiss National Science Foundation. Last, but not least, we thank the FTP steering committee, and in particular Ullrich Hustadt for supporting the FTP workshop series.

July 2011

Martin Giese
Program Chair

Program Chair

Martin Giese	University of Oslo, Norway

Program Committee

Roger Antonsen	University of Oslo, Norway
Peter Baumgartner	NICTA, Canberra, Australia
Bernhard Beckert	Karlsruhe Institute of Technology, Germany
Maria Paola Bonacina	Università degli Studi di Verona, Italy
Hans De Nivelle	University of Wroclaw, Poland
Ullrich Hustadt	University of Liverpool, UK
Reiner Hähnle	Chalmers Universityr, Göteborg, Sweden
Katsumi Inoue	National Institute of Informatics, Japan
Tudor Jebelean	RISC, Linz, Austria
Konstantin Korovin	Manchester University, UK
Jens Otten	University of Potsdam, Germany
Nicolas Peltier	CNRS - LIG, Grenoble, France
David Plaisted	University of North Carolina-Chapel Hill, USA
Andre Platzer	Carnegie Mellon University, USA
Silvio Ranise	FBK-Irst, Trento, Italy
Michael Rusinowitch	LORIA - INRIA Lorraine, France
Renate A. Schmidt	University of Manchester, UK
Viorica Sofronie-Stokkermans	MPI, Saarbrücken, Germany

Additional Reviewers

Rajeev Goré	Australian National University, Canberra, Australia
Vladimir Klebanov	Karlsruhe Institute of Technology, Germany
David Renshaw	Carnegie Mellon University, USA

Local organisation

Richard McKinley Roman Kuznets

FTP Steering Committee

Table of Contents

Monitoring First-order Temporal Properties 6
Felix Klaedtke

QMaxSAT version 0.3 & 0.4 7
Xuanye An, Miyuki Koshimura, Hiroshi Fujita and Ryuzo Hasegawa

Generating Schemata of Resolution Proofs 16
Vincent Aravantinos and Nicolas Peltier

Unification in a Theory of Blind Signatures 31
Serdar Erbatur, Christopher Lynch and Paliath Narendran

Incremental Variable Splitting 45
Christian Mahesh Hansen, Martin Giese, Arild Waaler and Roger Antonsen

Implementing and Evaluating Theorem Provers for First-Order Modal Logics 47
Thomas Raths and Jens Otten

Modular Termination and Combinability for Superposition Modulo Counter Arithmetic 64
Christophe Ringeissen and Valerio Senni

Monitoring First-order Temporal Properties

Felix Klaedtke

Computer Science Department, ETH Zurich, Switzerland

Abstract

In security and compliance, it is often necessary to ensure that agents and systems comply to complex policies. An example of such a policy from financial reporting is the requirement that every transaction t of a customer c, who has within the last 30 days been involved in a suspicious transaction t', must be reported as suspicious within 2 days. In this talk, I will give an overview of our approach to automated compliance checking. In particular, I will present our monitoring algorithm for checking properties specified in a fragment of metric first-order temporal logic. I will also report on case studies in security and compliance monitoring and use these to evaluate both the suitability of this fragment for expressing complex, realistic policies and the efficiency of the monitoring algorithm.

Joint work with David Basin, Matúš Harvan, Samuel Müller, and Eugen Zălinescu.

References

[1] D. Basin, M. Harvan, F. Klaedtke, and E. Zălinescu. Monitoring usage-control policies in distributed systems. In *Proceedings of the 18th International Symposium on Temporal Representation and Reasoning (TIME)*. IEEE Computer Society, 2011. To appear.

[2] D. Basin, M. Harvan, F. Klaedtke, and E. Zălinescu. MONPOLY: Monitoring usage-control policies. Submitted for publication, June 2011.

[3] D. Basin, F. Klaedtke, and S. Müller. Monitoring security policies with metric first-order temporal logic. In *Proceedings of the 15th ACM Symposium on Access Control Models and Technologies (SACMAT)*, pages 23–33. ACM Press, 2010.

[4] D. Basin, F. Klaedtke, S. Müller, and B. Pfitzmann. Runtime monitoring of metric first-order temporal properties. In *Proceedings of the 28th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, volume 2 of *Leibiz International Proceedings in Informatics (LIPIcs)*, pages 49–60. Schloss Dagstuhl - Leibniz Center for Informatics, 2008.

QMaxSAT version 0.3 & 0.4

Xuanye An Miyuki Koshimura Hiroshi Fujita
Ryuzo Hasegawa

Kyushu University
744 Motooka, Nishi-ku, Fukuoka, 819-0395 Japan
xuanye@ar.inf.kyushu-u.ac.jp {koshi,fujita,hasegawa}@inf.kyushu-u.ac.jp

Abstract

QMaxSAT is a partial Max-SAT solver obtained by adapting a SAT solver MiniSat for dealing with Boolean cardinality constraints. The version 0.1 was placed first in the industrial subcategory and second in the crafted subcategory of partial Max-SAT category of the 2010 MaxSAT evaluation. This paper presents new versions 0.3 and 0.4. The version 0.3 searches a solution by a binary method while version 0.1 does by a linear method. The version 0.4 alternates linear search and binary search. We give some experimental results by solving instances taken from the 2010 MaxSAT evaluation.

1 Introduction

Maximum Satisfiability (Max-SAT) is one of optimization counterparts of Boolean satisfiability (SAT). The objective is to find an assignment that maximizes the number of satisfied clauses [11, 10]. This is equivalent to finding an assignment that minimizes the number of unsatisfied clauses. There are two approaches to solve Max-SAT: approximation and exact algorithms. This paper considers exact solutions.

The exact solvers can be classified into two approaches. The one implements a branch and bound scheme and applies several techniques tailored to Max-SAT [16, 9, 12]. Another makes use of a state-of-the-art SAT solver as an inference engine [1, 13, 5]. We call this approach *SAT-based* approach.

QMaxSAT follows SAT-based approach. QMaxSAT is a partial Max-SAT (PMS) solver which uses CNF encoding of Boolean cardinality constraints. The old version 0.1 is obtained by adapting a CDCL (Conflict Driven Clause Learning) [14] based SAT solver MiniSat [8]. It was placed first in the industrial subcategory and second in the crafted subcategory of the PMS category of the 2010 MaxSAT Evaluation.

PMS is placed between SAT and Max-SAT. Unlike SAT requiring all clauses to be satisfied, PMS requires that clauses, called *soft*, are satisfied as many as possible, while other clauses, called *hard*, have to be satisfied. The objective of PMS is to find an assignment that satisfies all hard clauses and maximizes the number of satisfied soft clauses.

For unsatisfiable SAT instances, usual SAT solvers tell nothing but "unsatisfiable." In order to get more information from unsatisfiable instances with

SAT solvers, new variables called *blocking* are introduced. Roughly speaking, the number of satisfied blocking variables corresponds to that of unsatisfied soft clauses. Thus, the objective of QMaxSAT is to find an assignment that minimizes the number of satisfied blocking variables. Such an assignment can readily be found by iterative calls to a SAT solver with the cardinality constraints on blocking variables. QMaxSAT uses Bailleux's CNF encoding [3] for the Boolean cardinality constraint.

This paper presents new versions 0.3 and 0.4. The difference between the new versions and the previous versions 0.1 and 0.2 is how the cardinality constraints is managed. The new versions deal with both lower and upper bounds of the Boolean cardinality while the previous versions deal with only the upper bound. In this way, the new versions realize a binary search for a minimal assignment while the previous versions perform a linear search. Thus, the new versions might solve PMS instances faster than the previous versions do. The version 0.3 performs only binary search while version 0.4 alternates binary search and linear search.

2 Version 0.1 and 0.2

Let $C = H \cup S$ be a PMS instance consisting of a set H of hard clauses and a set S of n soft clauses $S_1, \ldots, S_n$, i.e. $S = \{S_1, \ldots, S_n\}$. We construct a new clause set $C^b = H \cup S^b$ by adding a new blocking variable b_i to S_i $(1 \leq i \leq n)$, namely, $S^b = \{S_1 \vee b_1, \ldots, S_n \vee b_n\}$. Thus, finding a PMS solution of C is reduced to find the minimal integer k satisfying C^b and $\sum_{i=1}^{n} b_i \leq k$, that is, minimize the number of satisfied blocking variables while satisfying C^b.

Let ϕ be a PMS instance augumented with blocking variables. First run the solver on ϕ to get an initial model and count the number k of satisfied blocking variables in the model, then add the constraint saying that the number of satisfied blocking variables have to be less than k, and run the solver again. If the problem is unsatisfied, k is the optimum solution. If not, the process is repeated with the new smaller solution. This process is essentially the same as the one in solvers for pseudo-Boolean optimization [17] and an algorithm for MAXONES [4].

The versions 0.1 and 0.2 realize the above process by introducing a CNF encoding of Boolean cardinality constraints on blocking variables. There are several works on encoding Boolean cadinality constraints into CNF formulas [3, 18, 15]. QMaxSAT uses Bailleux's encoding.

In this encoding, we introduce n new variables v_i for n blocking variables $b_i (1 \leq i \leq n)$. Then, we make a CNF formula $C(b_1, \cdots, b_n, v_1, \cdots, v_n)$ saying:

1. If m blocking variables are assigned 1, the first m variables $v_i (1 \leq i \leq m)$ are going to become assigned 1.
2. If m blocking variables are assigned 0, the last m variables $v_{n-i+1} (1 \leq i \leq m)$ are going to become assigned 0.

The encoding needs $O(n \cdot log\ n)$ auxiliary variables and $O(n^2)$ clauses. With the encoding, we encode the constraint $l \leq \sum_{i=1}^{n} b_i \leq k$ by setting the first l variables $v_i (1 \leq i \leq l)$ to 1, and the last $n - k$ variables $v_i (k < i \leq n)$ to 0.

Algorithm 1 shows the procedure of version 0.1. The function `solve(A)` denotes the core part of the SAT solver which returns `false` when a SAT instance

Algorithm 1 QMaxSAT version 0.1

```
1: $\mathtt{A} = \mathtt{C}^\mathtt{b}$; {$\mathtt{C}^\mathtt{b}$ : PMS instance augmented with blocking variables}
2: $\mathtt{sat} = \mathtt{false}$; $\mathtt{first} = \mathtt{true}$;
3: while (solve(A)) do
4:   Let $M$ be a model of A;
5:   "count the number $k$ of blocking variables assigned 1 in $M$";
6:   $\mathtt{sat} = \mathtt{true}$;
7:   if (first) then
8:     $\mathtt{first} = \mathtt{false}$;
9:     $\mathtt{A} = \mathtt{A} \wedge \mathtt{C}(\mathtt{b_1}, \cdots, \mathtt{b_n}, \mathtt{v_1}, \cdots, \mathtt{v_n})$; {augment the constraint to A}
10:  end if
11:  for $i = k$ to $n$ do
12:    $\mathtt{v_i} = 0$;
13:  end for{add the constraint $\sum_{i=1}^{n} b_i < k$}
14: end while
15: if (sat) then
16:   return the latest model $M$;
17: else
18:   return *unsatisfiable*;
19: end if
```

A is unsatisfiable and true when A is satisfiable. In the latter case, a model M of A is obtained through an array from which we count the number k of blocking variables assigned 1 in M.

After we obtain the first model of C^b, we build a CNF formula $C(b_1, \cdots, b_n, v_1, \cdots, v_n)$ which encodes the cardinality constraints (line 9). For every model obtained through solve(A), we introduce extra constraints (line 11,12,13). The k decreases as the procedure progresses. Thus, we reach a fix point at which we obtain a PMS solution (line 16). If C_b is unsatisfiable with no cardinality constraint, we conclude C has no PMS solution, that is, C is unsatisfiable (line 18).

A drawback of version 0.1 arises from the number of clauses for encoding Boolean cardinality constraints. Assuming that there are tens of thousands of soft clauses, the encoding needs hundreds of millions clauses. In our experience, the clauses cannot be held in 4GB memory when the number of soft clauses is greater than nine thousands.

Let k_1 be the number of blocking variables assigned 1 in the first model M in the procedure (see Algorithm 1). We can eliminate clauses in $C(b_1, \cdots, b_n, v_1, \cdots, v_n)$ dealing with integers greater than k_1. This elimination reduces the number of clauses for the encoding from $O(n^2)$ to $O(n \cdot k_1)$. The version 0.2 adopts the elimination.

3 Version 0.3

Both versions 0.1 and 0.2 count the number of satisfied blocking variables in each model obtained by the SAT solver. These numbers decrease linearly as the process proceeds. In this sense, both versions perform a linear search. The version 0.3, on the other hand, performs a binary search by dealing with both lower and upper bounds of the number of satisfied blocking variables. Note that the previous versions manage only the upper bound.

Let ϕ be a PMS instance augumented with blocking variables. First initialize the lower bound to 0 and the upper bound to n, the number of soft clauses. Second run the solver on ϕ to get an initial model and count the number k of blocking variables satisfied in the model. Next, add the constraint saying that the number of blocking variables satisfied has to be less than $\lceil k/2 \rceil$, and run the solver again. If the problem is unsatisfiable, we conclude that the number of satisfied blocking variables is at least $\lceil k/2 \rceil$, that is, the lower bound is updated to $\lceil k/2 \rceil$ while the upper bound is unchanged. On the other hand, if the problem is satisfiable, the upper bound is updated to the new k, the number of blocking variables satisfied in the new model. The process is continued until the lower bound and the upper bound become equal. In this way, the difference between the upper bound and the lower bound becomes less than a half of the previous one after every call of the solver. Thus, version 0.3 realizes a binary search.

Algorithm 2 QMaxSAT version 0.3

```
1: $\mathtt{A} = \mathtt{C^b}$; {$\mathtt{C^b}$ : PMS instance augmented with blocking variables}
2: $\mathtt{sat} = \mathtt{false}$; $\mathtt{first} = \mathtt{true}$;
3: $\mathtt{LB} = 0$; $\mathtt{UB} = \mathtt{n}$; $\mathtt{MP} = \lceil \mathtt{UB}/2 \rceil$; {$\mathtt{n}$: the number of soft clauses}
4: $\mathtt{Ass} = \top$; {assume nothing}
5: **while** $\mathtt{LB} < \mathtt{UB}$ **do**
6:   **if** solve(A,Ass) **then**
7:     Let $M$ be a model of A;
8:     "count the number $k$ of blocking variables assigned 1 in $M$";
9:     $\mathtt{sat} = \mathtt{true}$;
10:    $\mathtt{UB} = \mathtt{k}$; $\mathtt{MP} = \mathtt{LB} + \lceil (\mathtt{UB} - \mathtt{LB})/2 \rceil$;
11:    **if** (first) **then**
12:      $\mathtt{first} = \mathtt{false}$;
13:      $\mathtt{A} = \mathtt{A} \wedge \mathtt{C}(\mathtt{b_1}, \cdots, \mathtt{b_n}, \mathtt{v_1}, \cdots, \mathtt{v_n})$; {augment the constraint to A}
14:    **end if**
15:  **else**
16:    **if** (!sat) **then**
17:      return *unsatisfiable*;
18:    **end if**
19:    $\mathtt{LB} = \mathtt{MP}$; $\mathtt{MP} = \mathtt{LB} + \lceil (\mathtt{UB} - \mathtt{LB})/2 \rceil$;
20:  **end if**
21:  **for** $i = 1$ to LB **do**
22:    $\mathtt{v_i} = 1$;
23:  **end for**{$LB \leq \sum_{i=1}^{n} b_i$}
24:  **for** $i = \mathtt{UB}$ to $n$ **do**
25:    $\mathtt{v_i} = 0$;
26:  **end for**{$\sum_{i=1}^{n} b_i < UB$}
27:  $\mathtt{Ass} = \wedge_{\mathtt{i=MP}}^{\mathtt{UB}} (\mathtt{v_i} = 0)$; {assume $\sum_{i=1}^{n} b_i < MP$}
28: **end while**
29: return the latest model $M$;
```

Algorithm 2 shows the procedure of version 0.3. We introduce new programming variables LB, UB, and MP: LB and UB keep the lower bound and the upper bound, respectively. MP is updated whenever LB or UB is updated for keeping the middle of LB and UB (lines 3, 10, 19). The function solve(A,Ass) denotes the core part of the SAT solver which returns false when a SAT instance A is unsatisfiable and true when A is satisfiable under an assumption Ass. We require that the solver deals with a problem under an assumption which is a conjunction of unit clauses. The MiniSat meets the requirement.

For every model obtained through solve(A,Ass), the upper bound is updated (line 10). If solve(A,Ass) returns *false*, the lower bound is updated to MP (line 19) because there is no model under the assumption $LB \leq \sum_{i=1}^{n} b_i < MP$. After the lower bound or the upper bound is updated, we introduce extra constraints reflecting the update (lines 21 $\sim$ 26). Moreover, a new assumption is created (line 27).

4 Version 0.4

Generally speaking, binary search is better than linear search. Practically, however, there are instances for which linear search is better than binary search.

Let k_1 be the same number mentioned in the last paragraph of Section 2. Assuming k_1 is the PMS solution. We would conclude that k_1 is the solution after the next SAT solver call with version 0.1 and 0.2. However, we would reach the solution after $\log k_1$ solver calls with version 0.3. Thus, version 0.3 is not so good when k_1 is near to the solution.

The version 0.4 alternates binary search and linear search in order to benefit from both searches. It is realized by modifying Algorithm 2 as follows:

- Inserting the statement “BIN = 0;” before the line 4.
- Inserting the following statement before the line 19:
  “**if** BIN == 0 **then** **return** the latest model M **end if**;”
- Inserting the statement “BIN = 1-BIN;” before line 27.
- Replacing the statement at line 27 with
  “**if** BIN == 0 **then** Ass $= \top$ **else** Ass $= \wedge_{\mathtt{i=MP}}^{\mathtt{UB}}(\mathtt{v_i} = 0)$ **end if**”.

Here, a new programming variable BIN is introduced for indicating the searching mode: 0 for linear mode and 1 for binary mode. The value of BIN alternates between 0 and 1.

5 Experimental Results

We implemented QMaxSAT based on MiniSat 2.0. Tables 1 and 2 summarize the results obtained by running the four versions of QMaxSAT on the PMS instances of the fifth Max-SAT evaluation (Max-SAT 2010). The instances are divided into three different categories: random, crafted, and industrial.

The second column shows the number of instances of the corresponding category. The third, fourth, fifth and sixth columns show the average cpu time in seconds for instances solved by version 0.1, 0.2, 0.3 and 0.4, respectively. The number in parentheses indicates the number of instances solved. The experiments are done on Core i5-750 (4-core 2.66GHz) machine with 4GB memory. We used 1800 seconds as timeout.

Table 1: Comparison of four versions for PMS problems from Max-SAT 2010 (Random Category)

Solver	**#Ins.**	**V.0.1**	**V.0.2**	**V.0.3**	**V.0.4**
min2sat/v160c800l2/	30	0.00 (0)	0.00 (0)	0.00 (0)	0.00 (0)
min2sat/v260c1040l2/	30	0.00 (0)	0.00 (0)	0.00 (0)	0.00 (0)
min3sat/c70v350l3/	30	0.00 (0)	0.00 (0)	0.00 (0)	0.00 (0)
min3sat/c80v400l3/	30	0.00 (0)	0.00 (0)	0.00 (0)	0.00 (0)
pmax2sat/hi/	30	0.00 (1)	0.00 (1)	0.00 (1)	0.00 (1)
pmax2sat/me/	30	0.00 (0)	0.00 (0)	0.00 (0)	0.00 (0)
pmax3sat/hi/	30	0.00 (0)	0.00 (0)	644.80 (1)	1106.63 (1)
pmax3sat/lo/	30	46.54 (30)	31.49 (30)	12.14 (30)	16.37 (30)
All	240	45.04 (31)	30.48 (31)	31.54 (32)	49.93 (32)

For the random category, we succeed in descreasing the average cpu time for solving instances in the `pmax3sat/lo` subcategory with versions 0.3 and 0.4. The numbers of blocking variables satisfied in PMS solutions of the `pmax3sat/lo` subcategory are small, or less than 10. Binary search works well for solving such problems.

For the crafted category, version 0.4 is the best solver. It succeeds in solving 298 instances. There are 3 instances solved by version 0.2 but unsolved by version 0.3 while there are 2 instances solved by version 0.3 but unsolved by version 0.2. The version 0.4 succeeds in solving these 5 instances. From this viewpoint one may say that version 0.4 benefits from binary and linear searches.

For the industrial category, there seems to be no progress from version 0.2 to 0.3 or 0.4 while there is a little progress from version 0.1 to 0.2. There are 7 instances solved by version 0.2 but unsolved by version 0.3 while there are 8 instances solved by version 0.3 but unsolved by version 0.2. The version 0.4 succeeds in solving 9 instances of these 15 instances. On the other hand, it fails to solve 5 instances which both versions 0.2 and 0.3 solve. All these experiments make it clear that alternation of binary and linear searches does not always work well.

Table 3 shows the average number of SAT solver calls for instances solved in each category. We except unsatisfiable[*] PMS instances for calculation. The number in parentheses indicates the average number of SAT solver calls returning "unsatisfiable".

The last SAT solver call for each instance usually returns "unsatisfiable". Thus, there is at least one SAT solver call returning "unsatisfiable". However, some numbers in parentheses are less than one. When we find out a model which satisfies all soft clauses, we conclude that the model is a PMS solution without further SAT solver calls. There is no SAT solver call returning "unsatisfiable" for such instances. There are 20 and 7 such instances in the random category and the crafted category, respectively.

We succeed in decreasing the number of SAT solver calls with binary search. For example, it decreases from about eleven to about four for the random category. The decrease does not enhance QMaxSAT 's performance. The main

[*]There is only one unsatisfiable instance in our experiments. It is in the random category.

Table 2: Comparison of four versions for PMS problems from Max-SAT 2010 (Crafted and Industrial Categories)

(Crafted Category)

Solver	**#Ins.**	**V.0.1**	**V.0.2**	**V.0.3**	**V.0.4**
JobShop/	4	14.84 (4)	14.80 (4)	12.52 (4)	14.66 (4)
MAXCLIQUE/RANDOM/	96	65.27 (80)	64.75 (80)	85.76 (79)	91.03 (80)
MAXCLIQUE/STRUCTURED/	62	103.37 (26)	104.16 (26)	65.70 (25)	152.64 (27)
MAXONE/3SAT/	80	173.20 (78)	169.75 (78)	207.65 (80)	181.89 (80)
MAXONE/STRUCTURED/	60	11.72 (60)	10.60 (60)	32.76 (60)	31.46 (60)
PSEUDO/miplib/	4	1.52 (4)	1.66 (4)	2.40 (4)	1.78 (4)
frb/	25	206.89 (24)	182.04 (25)	95.09 (24)	142.67 (25)
min-enc/kbtree/	54	152.43 (18)	88.67 (18)	76.04 (18)	99.07 (18)
All	385	101.69 (294)	94.84 (295)	104.44 (294)	111.61 (298)

(Industrial Category)

Solver	**#Ins.**	**V.0.1**	**V.0.2**	**V.0.3**	**V.0.4**
CircuitTraceCompaction/	4	39.90 (4)	39.94 (4)	35.82 (4)	38.59 (4)
HaplotypeAssembly/	6	688.41 (2)	531.76 (5)	478.91 (5)	500.94 (5)
PROTEIN_INS/	12	498.61 (8)	849.78 (9)	734.91 (7)	116.47 (4)
bcp-fir/	59	46.01 (49)	18.72 (54)	17.79 (54)	48.34 (55)
bcp-hipp-yRa1/SU/	38	88.26 (32)	78.93 (31)	87.53 (31)	111.03 (32)
bcp-hipp-yRa1/simp/	17	50.65 (16)	43.56 (16)	55.09 (16)	45.54 (16)
bcp-msp/	64	18.28 (26)	15.02 (26)	30.45 (26)	24.69 (26)
bcp-mtg/	40	0.10 (40)	0.10 (40)	0.11 (40)	0.10 (40)
bcp-syn/	74	102.23 (34)	49.62 (34)	19.10 (33)	72.19 (34)
pbo-mqc/nencdr/	84	49.21 (77)	91.94 (80)	62.27 (80)	72.61 (81)
pbo-mqc/nlogencdr/	84	21.88 (76)	26.78 (77)	64.42 (81)	87.17 (80)
pbo-routing/	15	8.72 (15)	6.72 (15)	0.83 (15)	4.80 (15)
All	497	55.28 (379)	67.06 (391)	60.91 (392)	66.77 (392)

Table 3: The number of SAT solver calls

	V.0.1	V.0.2	V.0.3	V.0.4
Random	11.20 (0.33)	11.17 (0.33)	3.48 (0.83)	4.26 (0.55)
Crafted	10.30 (0.98)	11.14 (0.98)	8.11 (5.56)	8.56 (3.93)
Industrial	11.47 (1.00)	11.79 (1.00)	7.46 (4.50)	6.98 (2.65)

reason is that the number of SAT solver calls returning "unsatisfiable" increases while that of SAT solver calls returning "satisfiable" decreases. Generally speaking, proving the existence of models is easier than proving the nonexistence of them.

6 Conclusion

We have presented new versions 0.3 and 0.4 of QMaxSAT, a PMS solver based on a SAT solver MiniSat. The version 0.3 performs only binary search while version 0.4 alternates binary search and linear search. Experimental results show that version 0.4 is the best solver among four versions. It succeeds in solving the most instances taken from the PMS category of 2010 MaxSAT Evaluation.

In our experience, QMaxSAT obviously slows down when the number n of soft clauses and k_1[†] are greater than a few thousands. This is a major drawback of QMaxSAT and comes from the size $O(n \cdot k_1)$ of CNF-encoded Boolean cardinality constraints. In order to eliminate the drawback, we will replacing the current encoding with even more compact encodings [2, 7]. These encodings guarantee a space complexity of $O(n \cdot log^2 k_1)$.

We also plan to extend QMaxSAT to solve weighted Max-SAT problems.

Acknowledgment

This work was supported by JSPS KAKENHI(20240003).

References

[1] Carlos Ansótegui, María Luisa Bonet, and Jordi Levy. Solving (Weighted) Partial MaxSAT through Satisfiability Testing. In *Proc. of SAT 2009*, pages 427–440, 2009.

[2] Roberto Asín, Robert Nieuwenhuis, Albert Oliveras, and Enric Rodríguez-Carbonell. Cardinality Networks and Their Applications. In *Proc. of SAT 2009*, pages 167–180, 2009.

[†] k_1 is the same number mentioned in the last paragraph of Section 2.

[3] Olivier Bailleux and Yacine Boufkhad. Efficient CNF Encoding of Boolean Cardinality Constraints. In *Proc. of CP 2003*, pages 108–122, 2003.

[4] Olivier Bailleux and Yacine Boufkhad. Full CNF Encoding : The Counting Constraints Case. In *Proc. of SAT 2004*, pages 263–268, 2004.

[5] Daniel Le Berre and Anne Parrain. The Sat4j library, release 2.2. *JSAT*, 7:59–64, 2010.

[6] Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors. *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, 2009.

[7] Michael Codish and Moshe Zazon-Ivry. Pairwise Cardinality Networks. In *Proc. of LPAR-16*, pages 154–172, 2010.

[8] Niklas Eén and Niklas Sörensson. An Extensible SAT-solver. In *Proc. of SAT 2003*, pages 502–518, 2003.

[9] Federico Heras, Javier Larrosa, and Albert Oliveras. MiniMaxSat: An Efficient Weighted Max-SAT Solver. *J. of Artificial Intelligence Research*, 31:1–32, 2008.

[10] Katsutoshi Hirayama and Makoto Yokoo. *-SAT: Extentions of SAT. *J. of JSAI*, 25(1):105–113, 2010. in Japanese.

[11] Chu Min Li and Felip Manyà. *MaxSAT, Hard and Soft Constraints*, chapter 19, pages 613–631. Volume 185 of Biere et al. [6], 2009.

[12] Han Lin, Kaile Su, and Chu-Min Li. Within-Problem Learning for Efficient Lower Bound Computation in Max-SAT Solving. In *Proc. of AAAI-08*, pages 351–356, 2008.

[13] Vasco Manquinho, Joao Marques-Silva, and Jordi Planes. Algorithms for Weighted Boolean Optimization. In *Proc. of SAT 2009*, pages 495–508, 2009.

[14] Joao Marques-Silva, Ines Lynce, and Sharad Malik. *Conflict-Driven Clause Learning SAT Solvers*, chapter 4, pages 131–153. Volume 185 of Biere et al. [6], 2009.

[15] Joao Marques-Silvar and Inês Lynce. Towards Robust CNF Encodings of Cardinality Constraints. In *Proc. of CP 2007*, pages 483–497, 2007.

[16] Knot Pipatsrisawat and Adnan Darwiche. Clone: Solving Weighted Max-SAT in a Reduced Search Space. In *Proc. of AI 2007*, pages 223–233, 2007.

[17] Olivier Roussel and Vasco Manquinho. *Pseudo-Boolean and Cardinality Constraints*, chapter 22, pages 695–733. Volume 185 of Biere et al. [6], 2009.

[18] Carsten Sinz. Towards an Optimal CNF Encoding of Boolean Cardinality Constraints. In *Proc. of CP 2005*, pages 827–831, 2005.

Generating Schemata of Resolution Proofs[*]

Vincent Aravantinos and Nicolas Peltier
CNRS, LIG/TU Wien

Abstract

Two distinct algorithms are presented to extract (schemata of) resolution proofs from closed tableaux for propositional schemata [4]. The first one handles the most efficient version of the tableau calculus but generates very complex derivations (denoted by rather elaborate rewrite systems). The second one has the advantage that much simpler systems can be obtained, however the considered proof procedure is less efficient.

In [2, 4] a tableau calculus (called Stab) is presented for reasoning on schemata of propositional problems. This proof procedure is able to test the validity of logical formulæ built on a set of indexed propositional symbols, using generalized connectives such as $\bigvee_{\mathtt{i}=1}^{\mathtt{n}}$ or $\bigwedge_{\mathtt{i}=1}^{\mathtt{n}}$, where $\mathtt{i}, \mathtt{n}$ are part of the language ($\mathtt{n}$ denotes a parameter, i.e. an existentially quantified variable). A schema is unsatisfiable iff it is unsatisfiable for every value of $\mathtt{n}$. Stab combines the usual expansion rules of propositional logic with some delayed instantiation schemes that perform a case-analysis on the value of the parameter $\mathtt{n}$. Termination is ensured for a specific class of schemata, called *regular*, thanks to a loop detection rule which is able to prune infinite tableaux into finite ones, by encoding a form of mathematical induction (by "descente infinie"). A related algorithm, called Dpll* and based on an extension of the Davis-Putnam-Logemann-Loveland procedure, is presented in [3].

In the present work, we show that resolution proofs can be automatically extracted from the closed tableaux constructed by Stab or Dpll* on unsatisfiable schemata. More precisely, we present an algorithm that, given a closed tableau $\mathcal{T}$ for a schema $\phi_{\mathtt{n}}$, returns a schema of a refutation of $\phi_{\mathtt{n}}$ in the resolution calculus [9]. In the usual propositional case, it is well-known that algorithms exist to extract resolution proofs from closed tableaux constructed either by the usual structural rules [11, 13] or by the DPLL algorithm [7, 6]. The resolution proofs are used in various applications, for instance for certification [14], for abstraction-refinement [10] or for explanations generation [8]. The present paper extends these techniques to propositional schemata. Beside the previously mentioned applications, this turned out to be particularly important in the context of the ASAP project [1] in which schemata calculi are applied to

[*]This work has been partly funded by the project ASAP of the French *Agence Nationale de la Recherche* (ANR-09-BLAN-0407-01).

the formalisation and analysis of mathematical proofs via cut-elimination. Indeed, the algorithm used for cut-elimination, called CERES [5], explicitly relies on the existence of a resolution proof of the so-called *characteristic clause set* extracted from the initial proof. The cut-free proof is reconstructed from this refutation, by replacing the clauses occurring in this set by some "projections" of the original proof. While Stab and Dpll* are able to detect the unsatisfiability of characteristic clause sets, as such this is completely useless since actually it is known that those sets are *always* unsatisfiable (see Proposition 3.2 in [5]). It is thus essential to be able to generate *explicitly* a representation of the resolution proof. This is precisely the aim of the present paper. Since the initial formula depends on a parameter $\mathtt{n}$, its proof will also depend on $\mathtt{n}$ (except in very particular and trivial cases), i.e. it must be a *schema of resolution proof* (which will be encoded by recursive definitions).

The rest of the paper is structured as follows. In Section 1 we introduce the basic notions and notations used throughout our work, in particular the logic of *propositional schemata* (syntax and semantics). In Section 2 we define a tableau-based proof procedure for this logic. This calculus simulates both Stab and Dpll* (for the specific class of schemata considered in the present paper). In Section 3 we provide an algorithm to extract resolution proofs from closed tableaux. Similarly to the formulæ themselves, the constructed derivations are represented by rewrite systems. In Section 4 we introduce a second algorithm which generates simpler derivations but that requires that one of the closure rules defined in Section 2 (the so-called *Loop Detection* rule) be replaced by a less powerful rule, called the *Global Loop Detection* rule. Section 5 briefly concludes our work. Due to space restrictions, the proofs are omitted. They can be found at `http://arxiv.org/abs/1106.2692`.

1 Propositional schemata

The definitions used in the present paper differ from the previous ones, but the considered logic is equivalent to the class of regular schemata considered in [2] (it is thus strictly less expressive than general schemata, for which the satisfiability problem is undecidable). We consider three disjoint sets of symbols: a set of *arithmetic variables* $\mathcal{V}$, a set of *propositional variables* Ω and a set of *defined symbols* Υ. Let $\prec$ be a total well-founded ordering on the symbols in Υ. An *index expression* is either a natural number or of the form $\mathtt{n} + k$, where $\mathtt{n}$ is an arithmetic variable and k is a natural number. Let I be a set of index expressions. The set $\mathcal{F}(I)$ of *formulæ built on I* is inductively defined as follows: if $p \in \Omega \cup \Upsilon$ and $\alpha \in I$ then $p_\alpha \in \mathcal{F}(I)$; $\top, \bot \in \mathcal{F}(I)$; and if $\phi, \psi \in \mathcal{F}(I)$ then $\neg\phi$, $\phi \vee \psi$, $\phi \wedge \psi$, $\phi \Rightarrow \psi$ and $\phi \Leftrightarrow \psi$ are in $\mathcal{F}(I)$.

Definition 1 We assume that each element $\upsilon \in \Upsilon$ is mapped to two rewrite rules ρ^1_υ and ρ^0_υ that are respectively of the form $\upsilon_{\mathtt{i}+1} \to \phi$ (inductive case) and $\upsilon_0 \to \psi$ (base case), where $\phi \in \mathcal{F}(\{\mathtt{i}+1, \mathtt{i}, 0\})$, $\psi \in \mathcal{F}(\{0\})$ and:

1. For every atom τ_α occurring in ϕ such that $\tau \in \Upsilon$ we have either $\tau \prec \upsilon$ and $\alpha \in \{\mathtt{i}+1, \mathtt{i}, 0\}$ or $\tau = \upsilon$ and $\alpha \in \{0, \mathtt{i}\}$.

2. For every atom τ_α occurring in ψ such that $\tau \in \Upsilon$ we have $\tau \prec \upsilon$ and $\alpha = 0$. $\diamond$

We denote by $\mathcal{R}$ the rewrite system: $\{\rho^1_\upsilon, \rho^0_\upsilon \mid \upsilon \in \Upsilon\}$.

The rules ρ^1_υ and ρ^0_υ are provided by the user, they encode the semantics of the defined symbols. Conditions 1 and 2 in Definition 1 ensure that $\mathcal{R}$ is convergent. For every formula ϕ, we denote by $\phi\downarrow_{\mathcal{R}}$ the unique normal form of ϕ.

A *schema* (*of parameter* $\mathtt{n}$) is an element of $\mathcal{F}(\{0, \mathtt{n}, \mathtt{n}+1\})$. We denote by $\phi\{\mathtt{n} \leftarrow k\}$ the formula obtained from ϕ by replacing every occurrence of $\mathtt{n}$ by k. Obviously for any schema ϕ, $\phi\{\mathtt{n} \leftarrow k\} \in \mathcal{F}(\{0, k, k+1\})$. A *propositional formula* is a formula $\phi \in \mathcal{F}(\mathbb{N})$ containing no defined symbols. Notice that if $\phi \in \mathcal{F}(\mathbb{N})$ then $\phi\downarrow_{\mathcal{R}}$ is a propositional formula.

An *interpretation* is a function mapping every arithmetic variable $\mathtt{n}$ to a natural number and every atom of the form p_k (where $k \in \mathbb{N}$) to a truth value true or false. An interpretation I *validates* a propositional formula ϕ iff one of the following conditions holds: ϕ is of the form p_k and $I(p_k) =$ true; ϕ is of the form $\neg\psi$ and I does not validate ψ; or ϕ is of the form $\psi_1 \vee \psi_2$ (resp. $\psi_1 \wedge \psi_2$) and I validates ψ_1 or ψ_2 (resp. ψ_1 and ψ_2). *I validates a schema ϕ* (written $I \models \phi$) iff I validates $\phi\{\mathtt{n} \leftarrow I(\mathtt{n})\}\downarrow_{\mathcal{R}}$. We write $\phi \models \psi$ if every interpretation I validating ϕ also validates ψ and $\phi \equiv \psi$ if $\phi \models \psi$ and $\psi \models \phi$.

Example 2 The schema $p_0 \wedge \bigwedge_{\mathtt{i}=1}^{\mathtt{n}} (p_{\mathtt{i}-1} \Rightarrow p_{\mathtt{i}}) \wedge \neg p_{\mathtt{n}}$ is encoded by $p_0 \wedge \upsilon_{\mathtt{n}} \wedge \neg p_{\mathtt{n}}$, where υ is defined by the rules: $\upsilon_{\mathtt{i}+1} \rightarrow (\neg p_{\mathtt{i}} \vee p_{\mathtt{i}+1}) \wedge \upsilon_{\mathtt{i}}$ and $\upsilon_0 \rightarrow \top$.

The schema $\bigvee_{\mathtt{i}=1}^{\mathtt{n}} p_{\mathtt{i}} \wedge \bigwedge_{\mathtt{i}=1}^{\mathtt{n}} \neg p_{\mathtt{i}}$ is encoded by $\tau_{\mathtt{n}} \wedge \tau'_{\mathtt{n}}$, where τ and τ' are defined by the rules: $\tau_{\mathtt{i}+1} \rightarrow p_{\mathtt{i}+1} \vee \tau_{\mathtt{i}}$, $\tau_0 \rightarrow \bot$, $\tau'_{\mathtt{i}+1} \rightarrow \neg p_{\mathtt{i}+1} \wedge \tau'_{\mathtt{i}}$ and $\tau'_0 \rightarrow \top$.

Both schemata are obviously unsatisfiable.

The schema $(p_{\mathtt{n}} \Leftrightarrow (p_{\mathtt{n}-1} \Leftrightarrow (\ldots (p_1 \Leftrightarrow p_0) \ldots)))$ is defined by $\upsilon'_{\mathtt{n}}$, where: $\upsilon'_{\mathtt{i}+1} \rightarrow (p_{\mathtt{i}+1} \Leftrightarrow \upsilon'_{\mathtt{i}})$ and $\upsilon'_0 \rightarrow p_0$. ♣

2 Proof procedure

In this section we define the proof procedure used to decide the validity of propositional schemata. We assume for simplicity that the considered schemata are in negative normal form and that the defined symbols occur only positively[1].

The procedure is similar to the one presented in [2] and based on propositional block tableaux [12]. It constructs a tree labeled by finite sets of schemata, using *expansion rules* of the form: $\dfrac{\Phi}{\Psi_1 | \ldots | \Psi_k}$, meaning that a leaf whose label is of the form $\Phi \cup \Phi'$ (and does not already contain $\bot$) is expanded by adding k children labeled by $\Phi' \cup \Psi_1, \ldots, \Phi' \cup \Psi_k$ respectively. If α is a node in $\mathcal{T}$, then $\mathcal{T}(\alpha)$ denotes the label of α. The expansion rules are defined as follows:

[1] If a defined symbol υ occurs negatively then it is easy to replace every literal of the form $\neg\upsilon_\alpha$ by an atom $\overline{\upsilon}_\alpha$ where $\overline{\upsilon}$ denotes the complementary of υ. The rewrite rules for $\overline{\upsilon}$ are obtained by negating the right-hand side of the rules of υ, e.g. the atom $\overline{\upsilon}$ corresponding to the symbol υ in Example 2 is defined by the rewrite rules $\overline{\upsilon}_{\mathtt{i}+1} \rightarrow (p_{\mathtt{i}} \wedge \neg p_{\mathtt{i}+1}) \vee \overline{\upsilon}_{\mathtt{i}}$ and $\overline{\upsilon}_0 \rightarrow \bot$.

$$\text{Normalisation:} \quad \frac{\upsilon_\alpha}{\upsilon_\alpha\!\downarrow_{\mathcal{R}}} \quad \text{if } \upsilon_\alpha \text{ is reducible w.r.t. } \mathcal{R}$$

$$\begin{array}{ccc} \vee\text{-Decomposition} & \wedge\text{-Decomposition} & \text{Closure} \\ \dfrac{\phi \vee \psi}{\phi \mid \psi} & \dfrac{\phi \wedge \psi}{\phi, \psi} & \dfrac{\phi, \neg\phi}{\perp} \end{array}$$

$$\text{Purity rule:} \quad \frac{p_{\mathtt{n}+k}}{\top} \quad \frac{\neg p_{\mathtt{n}+k}}{\top} \quad \text{if } k > 0 \text{ and the previous rules do not apply}$$

Note that the notion of pure literal is much simpler here than in [2]. This is due to the fact that no constant index distinct from 0 and no index of the form $\mathtt{i} + k$ where $k > 1$ are allowed.

A node that is irreducible w.r.t. all the previous rules is called a *layer.* The *Loop Detection rule* applies to nodes containing previously generated layers:

$$\text{Loop Detection:} \quad \frac{\Phi}{\perp} \quad \text{if a non leaf layer labeled by } \Phi \text{ exists in the tree}$$

Note that the layer does not necessarily occur in the same branch as the one on which the rule is applied. The essential point is that the set of schemata Φ has already been considered somewhere – consequently if it has a model then an open branch necessarily exists elsewhere in the tree.

Finally, the last rule performs a case analysis on $\mathtt{n}$ (in this particular rule, Φ denotes the whole label of the considered node):

$$\text{Explosion:} \quad \frac{\Phi}{\Phi\{\mathtt{n} \leftarrow 0\} \mid \Phi\{\mathtt{n} \leftarrow \mathtt{n}+1\}} \quad \begin{array}{l}\text{if no other rule applies} \\ \text{and } \mathtt{n} \text{ occurs in } \Phi\end{array}$$

A tableau is *closed* if the labels of all leaves contain $\perp$.

Theorem 3 *The tableau expansion rules are terminating, i.e. there is no infinite sequence $(\mathcal{T}_i)_{i\in\mathbb{N}}$ such that for every $i \in \mathbb{N}$, $\mathcal{T}_{i+1}$ is obtained from $\mathcal{T}_i$ by applying one of the previous rules.*

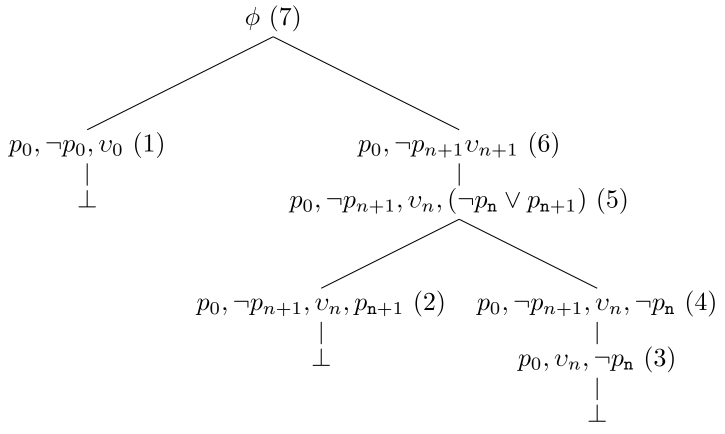
The next theorem states that the calculus is correct:

Theorem 4 *If $\mathcal{T}$ contains an irreducible leaf not containing $\perp$, then the label of the root of $\mathcal{T}$ is satisfiable.*

We will *not* prove the converse (namely that the root of every closed tableau is unsatisfiable), because this is subsumed by Theorem 12 in Section 3 (ensuring the existence of a resolution proof for every instance of the root schema).

Example 5 The schema $\phi : p_0 \wedge \neg p_n \wedge \upsilon_n$, where υ is defined as in Example 2 is unsatisfiable. For instance, $\phi\{\mathtt{n} \leftarrow 2\}$ is $p_0 \wedge \neg p_2 \wedge (\neg p_0 \vee p_1) \wedge (\neg p_1 \vee p_2)$. The reader can check that the expansion rules construct the following tableau.

The root is actually a layer, hence the Explosion rule is applied on it. The node (3) is deduced by the Purity rule and closed by applying the Loop Detection rule (with the root). The other rule applications are straightforward.



ϕ (7)
$p_0, \neg p_0, \upsilon_0$ (1)
$\bot$
$p_0, \neg p_{n+1} \upsilon_{n+1}$ (6)
$p_0, \neg p_{n+1}, \upsilon_n, (\neg p_{\mathtt{n}} \vee p_{\mathtt{n}+1})$ (5)
$p_0, \neg p_{n+1}, \upsilon_n, p_{\mathtt{n}+1}$ (2)
$\bot$
$p_0, \neg p_{n+1}, \upsilon_n, \neg p_{\mathtt{n}}$ (4)
$p_0, \upsilon_n, \neg p_{\mathtt{n}}$ (3)
$\bot$

♣

The DPLL* procedure in [3] can be simulated by the previous expansion rules, simply by adding for each propositional symbol $p \in \Omega$, a defined symbol υ^p with two rules: $\upsilon^p_{\mathtt{i}+1} \to ((p_{\mathtt{i}} \vee \neg p_{\mathtt{i}}) \wedge \upsilon^p_{\mathtt{i}})$ and $\upsilon^p_0 \to \top$. Then the case splitting rule of the DPLL procedure on a variable p corresponds to an application of the $\wedge$-rule on $\upsilon^p_{\mathtt{n}+1}\downarrow_{\mathcal{R}}$ (yielding $p_{\mathtt{n}} \vee \neg p_{\mathtt{n}}$) followed by an application of the $\vee$-rule on $p_{\mathtt{n}} \vee \neg p_{\mathtt{n}}$. The propagation rule is then simulated by combining the $\vee$-rule and the closure rule[2].

3 Constructing resolution proofs

3.1 Propositional resolution calculus

We first briefly recall the notion of resolution inference (in propositional logic). A *literal* is either an atom p_k or the negation of an atom $\neg p_k$ (where $p \in \Omega$ and $k \in \mathbb{N}$). A *clause* is a (possibly empty) disjunction (or set) of literals. A *derivation* from a set of clauses S is a finite sequence $C_1, \dots, C_m$ such that for every $i \in [1, m]$, C_i is either an element of S or obtained from $C_1, \dots, C_{m-1}$ by applying the resolution rule, defined as follows: $\dfrac{p_k \vee X \quad \neg p_k \vee Y}{X \vee Y}$

A *refutation* is a derivation containing $\bot$ (the empty clause). For any formula ϕ, Δ is a *derivation from* ϕ if it is a derivation from a clausal form of ϕ.

It is well-known [9] that every unsatisfiable set of (propositional) clauses has a refutation. In the context of propositional schemata, this means that every *instance* $\phi\{\mathtt{n} \leftarrow k\}\downarrow_{\mathcal{R}}$ of an unsatisfiable propositional schema ϕ of parameter $\mathtt{n}$ has a refutation Δ_k (which in general depends on k). The problem is then to construct a representation of the sequence of refutations $\Delta_0, \Delta_1, \dots, \Delta_k, \dots$

[2]This "trick" does not actually simulate the full procedure in [3], because the latter handles schemata that are more complex than the ones considered in the present paper, possibly containing nested iterations.

This sequence may be seen as a *schema of refutation* which (similarly to the semantics of the defined symbols) will be denoted by a *system of rewrite rules*. From now, we assume that the considered schema is in conjunctive normal form (i.e. it contains no conjunctions inside disjunctions, even if these conjunctions are "hidden" in the inductive definitions of the defined symbols, e.g. the schema $p_{\mathtt{n}} \vee \upsilon_{\mathtt{n}}$, where υ is defined as in Example 2 is forbidden).

3.2 A language for representing refutations

Additional definitions are needed to provide suitable formal languages for denoting such schemata of derivations. Let $\mathcal{D}$ and $\mathcal{X}$ be two disjoint sets of symbols (disjoint from $\mathcal{V}$, Ω and Υ). The symbols in $\mathcal{D}$ are the Δ-*symbols* and the ones in $\mathcal{X}$ are the Δ-*variables*. The symbols in $\mathcal{X}$ are intended to be instantiated by schemata, whereas the symbols $d \in \mathcal{D}$ will denote schemata of refutations, defined by induction (and possibly depending on an additional argument Δ denoting a formula). We assume that $\prec$ is extended into a well-founded ordering on $\mathcal{D}$.

Formally, the set of Δ-expressions is inductively defined as follows:

- All schemata and all Δ-variables are Δ-expressions.
- If $d \in \mathcal{D}$, α is an index expression and Δ is a Δ-expression, then d_α and $d_\alpha(\Delta)$ are Δ-expressions.
- If Δ and Γ are Δ-expressions then $\Delta \vee \Gamma$, $\Delta \wedge \Gamma$ and $\Delta \cdot \Gamma$ are Δ-expressions.

The expression $\Delta \cdot \Gamma$ is to be interpreted as the concatenation of two sequences Δ and Γ. Note that Δ-expressions can represent uniformly schemata of clauses, schemata of clause sets, or schemata of derivations (i.e. schemata of sequences of clauses). For the sake of conciseness and simplicity, the previous definition does *not* ensure that the constructions are well-typed, e.g. we can consider Δ-expressions of the form $\Delta \vee \Gamma$ where Δ and Γ are two *sequences of clauses* (which obviously does not make sense: Δ and Γ should rather be *clauses*). But in the forthcoming definitions we will ensure that all the considered Δ-expressions are well-typed.

Example 6 Let $d \in \mathcal{D}$. Then $(p_2 \vee q_0) \cdot d_2(q_0) \cdot \neg q_0 \cdot \bot$ is a Δ-expression. ♣

A Δ-expression is *ground* if it contains no index variable and no Δ-variable. In order to interpret (ground) Δ-expressions, the value of the Δ-symbols is specified using a rewrite system, exactly as schemata can be transformed into propositional formulæ by interpreting the defined symbols (using the rewrite system $\mathcal{R}$). The rewrite systems used in this section are more complicated than in the previous one, since the symbols in $\mathcal{D}$ may have an additional argument.

A Δ-*substitution* is a function mapping every arithmetic variable to an index expression and every Δ-variable to a Δ-expression. If Δ is a Δ-expression and σ is a Δ-substitution, then $\Delta\sigma$ denotes the Δ-expression obtained from Δ by replacing every variable $x \in \mathcal{V} \cup \mathcal{X}$ by $\sigma(x)$.

Definition 7 A $\mathcal{D}$-*system* is a set of *rewrite rules* of the form $\Delta \to \Gamma$, where Δ, Γ are two Δ-expressions such that every arithmetic variable and every Δ-

variable occurring in Γ also occurs in Δ. A $\mathcal{D}$-system is *propositional* if it contains no Δ-variables (it may contain arithmetic variables).

Given two Δ-expressions Δ and Γ and a $\mathcal{D}$-system $\mathfrak{R}$, we write $\Delta \to_{\mathfrak{R}} \Gamma$ if there exists a rule $\Delta' \to \Gamma'$ in $\mathfrak{R}$ and a Δ-substitution σ such that Γ is obtained from Δ by replacing an occurrence of an expression $\Delta'\sigma$ by $\Gamma'\sigma$. $\diamond$

For matching, the associativity and commutativity of logical symbols are not taken into account in general, *except for conjunctions occurring at the root level* (this rather unusual convention is needed to ensure confluence without having to bother on the order of the schemata at the root level). For instance the rule $d(p \wedge ((r \wedge q) \vee \neg r)) \to p$ does not apply on $d(p \wedge (\neg r \vee (r \wedge q)))$ nor on $d(p \wedge ((q \wedge r) \vee \neg r))$, but it applies on $d(((r \wedge q) \vee \neg r) \wedge p)$. Similarly, $d(p \wedge q) \to p$ applies on $d(p)$ by assuming $q = \top$.

Example 8 Consider the following rewrite system (Z is a Δ-variable).

$$\{d_{\mathtt{i}+1}(Z) \to (\neg p_{\mathtt{i}+1} \vee p_{\mathtt{i}}) \cdot (p_{\mathtt{i}} \vee Z) \cdot d_{\mathtt{i}}(Z), \quad d_0(Z) \to \neg p_0 \cdot Z\}$$

The reader can check that it reduces the Δ-expression of Example 6 to:

$$(p_2 \vee q_0) \cdot (\neg p_2 \vee p_1) \cdot (p_1 \vee q_0) \cdot (\neg p_1 \vee p_0) \cdot (p_0 \vee q_0) \cdot \neg p_0 \cdot q_0 \cdot \neg q_0 \cdot \bot$$

This last expression is a refutation. ♣

3.3 From closed tableaux to resolution proofs

Let $\mathcal{T}$ be a closed tableau of a schema ϕ. The general idea is to construct, from $\mathcal{T}$, a $\mathcal{D}$-system $\mathfrak{R}(\mathcal{T})$ representing a schema of refutation for ϕ. Obviously, $\mathfrak{R}(\mathcal{T})$ represents an inductive proof of the assertion: "for every $\mathtt{n} \in \mathbb{N}$, the corresponding instance of ϕ is unsatisfiable". Ideally, we would just refute the base case, and then build a refutation of ϕ at $\mathtt{n}+1$ from a refutation of ϕ at $\mathtt{n}$. However, as often in inductive reasoning, we need to generalize the conjecture in order to refute it properly. This is done as follows: recall that our aim is to construct a refutation of ϕ, i.e. a derivation of $\bot$ from ϕ; instead, however, $\mathfrak{R}(\mathcal{T})$ will describe how to build a derivation of X from $\phi \vee X$, *for any* X (formally, X will be a Δ-variable). Then, our original goal will be reached by just substituting $\bot$ to X. In practice, we need to generalize even more this reasoning since the construction of $\mathfrak{R}(\mathcal{T})$ is done by mapping *every node* α of $\mathcal{T}$ to some rewrite rules. So, instead of considering only the root schema ϕ, we need to consider all the formulæ $\{\phi_1, \dots, \phi_k\}$ that occur in $T(\alpha)$. And, instead of building a derivation of X from $\phi \vee X$, we build a derivation of $X_1 \vee \dots \vee X_k$ from $(\phi_1 \vee X_1) \wedge \dots \wedge (\phi_k \vee X_k)$, for some Δ-variables $X_1, \dots, X_k$. More precisely we build a derivation of a clause $C \subseteq X_1 \vee \ldots \vee X_k$, since some formulæ $\phi_i \vee X_i$ may be useless. We retrieve our original goal when we just substitute the root of $\mathcal{T}$ to α.

The following definition constructs a $\mathcal{D}$-system $\mathfrak{R}(\mathcal{T})$ and two Δ-symbols ν^α and μ^α such that, if $\mathcal{T}(\alpha) = \{\phi_1, \dots, \phi_k\}$ and U denotes the formula $(\phi_1 \vee X_1) \wedge \dots \wedge (\phi_k \vee X_k)$ then $\mu^\alpha_{\mathtt{n}}(U)$ denotes the above clause C and $\nu^\alpha_{\mathtt{n}}(U)$ denotes a derivation of C from U. This system is constructed by induction on the tableau.

Definition 9 Let $\mathcal{T}$ be a tableau. We map every node α in $\mathcal{T}$ to two Δ-symbols ν^α and μ^α. We assume that the symbols ν^α and μ^α are pairwise distinct. The system of rules $\mathfrak{R}(\mathcal{T})$ is defined by the rules in $\mathcal{R}$ and the following rules, for every node α in $\mathcal{T}$ (we distinguish several cases, according to the rule applied on α):

- If no rule is applied on α: $\nu^\alpha_{\mathtt{n}}((\bot \vee X) \wedge Y) \to X \quad \mu^\alpha_{\mathtt{n}}((\bot \vee X) \wedge Y) \to X$
- If the Normalisation rule is applied on α, using a formula ϕ, yielding a node β:
$$\nu^\alpha_{\mathtt{n}}((\phi \vee X) \wedge Y) \to \nu^\beta_{\mathtt{n}}((\phi{\downarrow_{\mathcal{R}}} \vee X) \wedge Y)$$
$$\mu^\alpha_{\mathtt{n}}((\phi \vee X) \wedge Y) \to \mu^\beta_{\mathtt{n}}((\phi{\downarrow_{\mathcal{R}}} \vee X) \wedge Y)$$
- If the Closure rule is applied on α, using ϕ and $\neg\phi$:
$$\nu^\alpha_{\mathtt{n}}((\phi \vee X) \wedge (\neg\phi \vee Y) \wedge Z) \to (\neg\phi \vee Y) \cdot (\phi \vee X) \cdot (X \vee Y)$$
$$\mu^\alpha_{\mathtt{n}}((\phi \vee X) \wedge (\neg\phi \vee Y) \wedge Z) \to (X \vee Y)$$
- If $\wedge$-Decomposition is applied on α, yielding a child β:
$$\nu^\alpha_{\mathtt{n}}(((\phi_1 \wedge \phi_2) \vee X) \wedge Y) \to \nu^\beta_{\mathtt{n}}((\phi_1 \vee X) \wedge (\phi_2 \vee X) \wedge Y)$$
$$\mu^\alpha_{\mathtt{n}}(((\phi_1 \wedge \phi_2) \vee X) \wedge Y) \to \mu^\beta_{\mathtt{n}}((\phi_1 \vee X) \wedge (\phi_2 \vee X) \wedge Y)$$
- If $\vee$-Decomposition is applied on α using a formula $\phi \vee \psi$ and yielding two children β_1 and β_2:
$$\nu^\alpha_{\mathtt{n}}(((\phi_1 \vee \phi_2) \vee X) \wedge Y) \to \nu^{\beta_1}_{\mathtt{n}}((\phi_1 \vee (\phi_2 \vee X)) \wedge Y) \cdot \nu^{\beta_2}_{\mathtt{n}}(\mu^{\beta_1}_{\mathtt{n}}((\phi_1 \vee (\phi_2 \vee X)) \wedge Y) \wedge Y)$$
$$\mu^\alpha_{\mathtt{n}}(((\phi_1 \vee \phi_2) \vee X) \wedge Y) \to \mu^{\beta_2}_{\mathtt{n}}(\mu^{\beta_1}_{\mathtt{n}}((\phi_1 \vee (\phi_2 \vee X)) \wedge Y) \wedge Y)$$
- If the Purity rule is applied on α, on a formula ϕ, yielding a node β:
$$\nu^\alpha_{\mathtt{n}}((\phi \vee X) \wedge Y) \to \nu^\beta_{\mathtt{n}}(Y) \quad \mu^\alpha_{\mathtt{n}}((\phi \vee X) \wedge Y) \to \mu^\beta_{\mathtt{n}}(Y)$$
- If the Loop Detection rule is applied on α, using a layer β:
$$\nu^\alpha_{\mathtt{n}}(X) \to \nu^\beta_{\mathtt{n}}(X) \quad \mu^\alpha_{\mathtt{n}}(X) \to \mu^\alpha_{\mathtt{n}}(X)$$
- If the Explosion rule is applied on α, yielding two children β_1 and β_2, corresponding to the cases $\mathtt{n} \leftarrow 0$ and $\mathtt{n} \leftarrow \mathtt{n}+1$ respectively:
$$\nu^\alpha_0(X) \to \nu^{\beta_1}_0(X) \quad \nu^\alpha_{\mathtt{n}+1}(X) \to \nu^{\beta_1}_{\mathtt{n}}(X) \quad \mu^\alpha_0(X) \to \mu^{\beta_2}_0(X) \quad \mu^\alpha_{\mathtt{n}+1}(X) \to \mu^{\beta_2}_{\mathtt{n}}(X)$$

$\diamond$

Note that all the symbols ϕ, ϕ_1, ϕ_2 denote meta-variables, and not Δ-variables (hence they cannot be instantiated during rewriting, in contrast to X, Y,...).

Before establishing the properties of $\mathfrak{R}(\mathcal{T})$, we show an example of application:

Example 10 Consider the proof tree of Example 5. The reader can check that $\mathfrak{R}(\mathcal{T})$ contains the following rules:

$$\begin{array}{lcl}
\nu^1_{\mathtt{n}}((p_0 \vee X) \wedge (\neg p_0 \vee Y) \wedge Z) & \rightarrow & (p_0 \vee X) \cdot (\neg p_0 \vee Y) \cdot (X \vee Y) \\
\mu^1_{\mathtt{n}}((p_0 \vee X) \wedge (\neg p_0 \vee Y) \wedge Z) & \rightarrow & X \vee Y \\
\nu^2_{\mathtt{n}}((p_{\mathtt{n}+1} \vee X) \wedge (\neg p_{\mathtt{n}+1} \vee Y) \wedge Z) & \rightarrow & (p_{\mathtt{n}+1} \vee X) \cdot (\neg p_{\mathtt{n}+1} \vee Y) \cdot (X \vee Y) \\
\mu^2_{\mathtt{n}}((p_{\mathtt{n}+1} \vee X) \wedge (\neg p_{\mathtt{n}+1} \vee Y) \wedge Z) & \rightarrow & X \vee Y \\
\nu^3_{\mathtt{n}}(X) & \rightarrow & \nu^7_{\mathtt{n}}(X) \\
\mu^3_{\mathtt{n}}(X) & \rightarrow & \mu^7_{\mathtt{n}}(X) \\
\nu^4_{\mathtt{n}}((\neg p_{\mathtt{n}+1} \vee X) \wedge Y) & \rightarrow & \nu^3_{\mathtt{n}}(Y) \\
\mu^4_{\mathtt{n}}((\neg p_{\mathtt{n}+1} \vee X) \wedge Y) & \rightarrow & \mu^3_{\mathtt{n}}(Y) \\
\nu^5_{\mathtt{n}}(((\neg p_{\mathtt{n}} \vee p_{\mathtt{n}+1}) \vee X) \wedge Y) & \rightarrow & \nu^2_{\mathtt{n}}(((p_{\mathtt{n}+1}) \vee (\neg p_{\mathtt{n}} \vee X)) \wedge Y) \\
 & & \cdot \nu^4_{\mathtt{n}}(\mu^2_{\mathtt{n}}((p_{\mathtt{n}+1} \vee (\neg p_{\mathtt{n}} \vee X)) \wedge Y) \wedge Y) \\
\mu^5_{\mathtt{n}}(((\neg p_{\mathtt{n}} \vee p_{\mathtt{n}+1}) \vee X) \wedge Y) & \rightarrow & \mu^4_{\mathtt{n}}(\mu^2_{\mathtt{n}}((p_{\mathtt{n}+1} \vee (\neg p_{\mathtt{n}} \vee X)) \wedge Y) \wedge Y) \\
\nu^6_{\mathtt{n}}((\upsilon_{\mathtt{n}+1} \vee X) \wedge Y) & \rightarrow & \nu^5_{\mathtt{n}}(((\neg p_{\mathtt{n}} \vee p_{\mathtt{n}+1}) \vee X) \wedge \upsilon_{\mathtt{n}} \wedge Y) \\
\mu^6_{\mathtt{n}}((\upsilon_{\mathtt{n}+1} \vee X) \wedge Y) & \rightarrow & \mu^5_{\mathtt{n}}(((\neg p_{\mathtt{n}} \vee p_{\mathtt{n}+1}) \vee X) \wedge \upsilon_{\mathtt{n}} \wedge Y) \\
\nu^7_0(X) & \rightarrow & \nu^1_0(X) \\
\mu^7_0(X) & \rightarrow & \mu^1_0(X) \\
\nu^7_{\mathtt{n}+1}(X) & \rightarrow & \nu^6_{\mathtt{n}}(X) \\
\mu^7_{\mathtt{n}+1}(X) & \rightarrow & \mu^6_{\mathtt{n}}(X)
\end{array}$$

The Δ-expression $\nu^7_{\mathtt{n}}((p_0 \vee \bot) \wedge (\neg p_{\mathtt{n}} \vee \bot) \wedge (\upsilon_{\mathtt{n}} \vee \bot))$ denotes a refutation of $p_0 \wedge \neg p_{\mathtt{n}} \wedge \upsilon_{\mathtt{n}}$. This rewrite system is complex and hardly readable, fortunately it can be simplified by instantiating the arguments when possible and by *statically* evaluating the derivations that do no depend on the value of the parameter $\mathtt{n}$. For instance the Δ-symbol $\nu^7_{\mathtt{n}}$ is only called on the formula $T_{\mathtt{n}} = (p_0 \vee \bot) \wedge (\neg p_n \vee \bot) \wedge (\upsilon_n \vee \bot)$. Thus the rule $\nu^7_{\mathtt{n}}(X) \rightarrow \nu^1_0(X)$ may be simplified by instantiating X by T_0 and evaluating the right-hand side: $\nu^7_0(T_0) \rightarrow p_0 \cdot \neg p_0 \cdot \bot$

Similarly, the rule $\nu^7_{\mathtt{n}+1}(X) \rightarrow \nu^6_{\mathtt{n}}(X)$ can be replaced by the following rule (in this case only a partial evaluation is possible since some parts of the derivation depend on the value of $\mathtt{n}$): $\nu^7_{\mathtt{n}+1}(T_{\mathtt{n}+1}) \rightarrow (\neg p_{\mathtt{n}} \vee p_{\mathtt{n}+1}) \cdot \neg p_{\mathtt{n}+1} \cdot \neg p_{\mathtt{n}} \cdot \nu^7_{\mathtt{n}}(T_{\mathtt{n}})$

The obtained system (only containing the two previous rules) is obviously much simpler than the original one, in particular it is propositional (no schema variables occur in it). To improve readability, the expression $\nu^7_{\mathtt{n}}(T_{\mathtt{n}})$ could be simply replaced by a fresh symbol $\nu^{7'}_{\mathtt{n}}$ (with no argument). ♣

Lemma 11 *Let $\mathcal{T}$ be a tableau. $\mathfrak{R}(\mathcal{T})$ is convergent.*

For any Δ-expression T, we denote by $T{\downarrow_{\mathfrak{R}(\mathcal{T})}}$ the normal form of T. We now state the soundness of our algorithm.

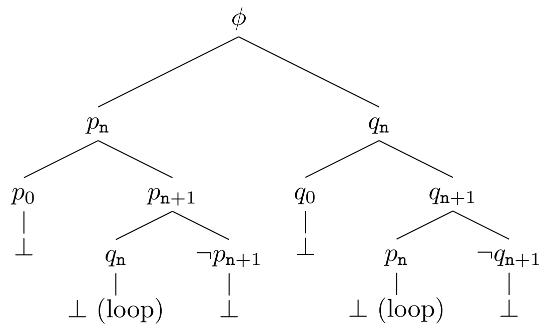
Theorem 12 *Let $\mathcal{T}$ be a closed tableau containing a node α. Let $\mathtt{n}$ be the parameter of $\mathcal{T}(\alpha)$. Let $\mathcal{T}(\alpha) = \{\phi_1, \dots, \phi_n\}$ and let $\Phi = (\phi_1 \vee \bot) \wedge \dots \wedge (\phi_n \vee \bot)$.*

For any $k \in \mathbb{N}$, $\nu_k^\alpha(\Phi\{\mathtt{n} \leftarrow k\})\downarrow_{\mathfrak{R}(\mathcal{T})}$ is a refutation of $\Phi\{\mathtt{n} \leftarrow k\}\downarrow_{\mathcal{R}}$. Thus $\mathcal{T}(\alpha)$ is unsatisfiable.

Note that the size of the rewrite system $\mathfrak{R}(\mathcal{T})$ is clearly linear w.r.t. the one of the tableau $\mathcal{T}$.

The simplification phase used in Example 10 can be applied in a systematic way. However, it is not always sufficient to reduce the rewrite system into a propositional one. Actually, it is not difficult to see that as soon as a node α exists in the tableau on which the $\vee$-Decomposition rule is applied, yielding two branches β_1 and β_2 that are *both* looping on an ascendant of α, then the use of schema variables cannot be avoided.

Example 13 Consider for instance the schema: $\phi : \neg p_0 \wedge \neg q_0 \wedge (p_\mathtt{n} \vee q_\mathtt{n}) \wedge \upsilon_\mathtt{n}$, where υ is defined by the rules: $\upsilon_{\mathtt{i}+1} \to (q_\mathtt{i} \vee \neg p_{\mathtt{i}+1}) \wedge (p_\mathtt{i} \vee \neg q_{\mathtt{i}+1}) \wedge \upsilon_\mathtt{i}$ and $\upsilon_0 \to \top$. The following tableau is constructed:



The corresponding rewrite system (after partial evaluation and simplification) is the following ($\nu_\mathtt{n}^1$ corresponds to the refutation of ϕ):

$$\begin{aligned}
&\nu_\mathtt{n}^1(\neg p_0 \wedge \neg q_0 \wedge (p_\mathtt{n} \vee q_\mathtt{n}) \wedge \upsilon_\mathtt{n}) \to \\
&\qquad \nu_\mathtt{n}^2(\neg p_0 \wedge \neg q_0 \wedge (p_\mathtt{n} \vee q_\mathtt{n}) \wedge \upsilon_\mathtt{n}) \cdot \nu_\mathtt{n}^3(\neg p_0 \wedge \neg q_0 \wedge q_\mathtt{n} \wedge \upsilon_\mathtt{n}) \\
&\nu_0^2(\neg p_0 \wedge (p_0 \vee X) \wedge Y) \to \neg p_0 \cdot (p_0 \vee X) \cdot X \\
&\nu_{\mathtt{n}+1}^2(\neg p_0 \wedge \neg q_0 \wedge (p_{\mathtt{n}+1} \vee X) \wedge \upsilon_{\mathtt{n}+1}) \to \\
&\qquad (p_{\mathtt{n}+1} \vee X) \cdot (q_\mathtt{n} \vee \neg p_{\mathtt{n}+1}) \cdot (q_\mathtt{n} \vee X) \cdot \nu_\mathtt{n}^3(\neg p_0 \wedge \neg q_0 \wedge (q_\mathtt{n} \vee X) \wedge \upsilon_\mathtt{n}) \\
&\nu_0^3(\neg p_0 \wedge (q_0 \vee X) \wedge Y) \to \neg q_0 \cdot (q_0 \vee X) \cdot X \\
&\nu_{\mathtt{n}+1}^3(\neg p_0 \wedge \neg q_0 \wedge (q_{\mathtt{n}+1} \vee X) \wedge \upsilon_{\mathtt{n}+1}) \to \\
&\qquad (q_{\mathtt{n}+1} \vee X) \cdot (p_\mathtt{n} \vee \neg q_{\mathtt{n}+1}) \cdot (p_\mathtt{n} \vee X) \cdot \nu_\mathtt{n}^2(\neg p_0 \wedge \neg q_0 \wedge (p_\mathtt{n} \vee X) \wedge \upsilon_\mathtt{n})
\end{aligned}$$

The system still contains Δ-variables, although some of them have been removed by static evaluation. Note that it could be further simplified (for instance by moving the axioms such as $\neg p_0$ outside the inductive definitions), but the use of Δ-variables cannot be avoided. ♣

We now focus on an alternative approach that has the advantage that only propositional rewrite systems are generated.

4 Globally looping tableaux

Compared to the previous approach, the second algorithm generates much simpler rewrite systems, but it has the drawback that a more restrictive version of the Loop Detection rule must be used to prune the tableaux into finite ones. At a very high and informal level: in the first approach, we were building mutually inductive proofs of several lemmata, whereas, in the second approach, we manage to have one single invariant proved by a single induction.

We first need to introduce some additional terminology. A node α is *of rank* k in a tableau $\mathcal{T}$ of root β if there are *exactly* k applications of the Explosion rule between β and α (including β, but not α). $\text{Leaves}(\mathcal{T}, \alpha)$ denotes the set of non-closed leaves below α in $\mathcal{T}$, $\text{Layers}(\mathcal{T}, k)$ denotes the set of layers of rank k in $\mathcal{T}$ and $\text{Layers}(\mathcal{T}, k, \alpha)$ denotes the set of layers of rank k in $\mathcal{T}$ that occur below α. For any set of formulæ Φ, we denote by $\bigwedge \Phi$ the conjunction $\bigwedge_{\phi \in \Phi} \phi$. If $\mathcal{T}$ is a tableau and N is a set of nodes in $\mathcal{T}$, then $\mathcal{T}[N]$ denotes the disjunction $\bigvee_{\alpha \in N} \bigwedge \mathcal{T}(\alpha)$. We write $\text{cnf}(\phi)$ for a (subsumption-minimal) clausal form of $\phi{\downarrow_{\mathcal{R}}}$.

Definition 14 A tableau $\mathcal{T}$ is *globally looping* (w.r.t. two natural numbers k and n) iff the following conditions hold:

1. $n < k$.
2. $\mathcal{T}[\text{Layers}(\mathcal{T}, k)] = \mathcal{T}[\text{Layers}(\mathcal{T}, n)]$ (modulo AC and idempotence).
3. All non-closed leaves in $\mathcal{T}$ are of a rank greater or equal to k.

Then the *Global Loop Detection rule* closes every node in $\text{Layers}(\mathcal{T}, k)$. $\diamond$

By definition, after the Global Loop Detection rule is applied, all branches containing the parameter `n` are closed and the construction of the tableau is over (since no leaf can be expanded anymore). Note that the Global Loop Detection rule can be simulated by several applications of the Loop Detection rule introduced in Section 2. Indeed, assume that a pair of natural numbers (k, n) satisfying the conditions of Definition 14 exists. Then, by Condition 2, for every layer α of rank k, there exists a layer β of rank n such that $\mathcal{T}(\alpha) = \mathcal{T}(\beta)$. Thus the Loop Detection rule applies on α (w.l.o.g. we assume that the layers of rank n are constructed before those of rank k in all parallel branches, which is possible since $n < k$). However, it is easy to see that the converse does not hold: the Global Loop Detection rule is *strictly less general* than the looping rule. It is, however, powerful enough to ensure termination, provided that a fair strategy is used to expand the tableau, as stated by the following theorem:

Theorem 15 *Let $(\mathcal{T}_i)_{i \in \mathbb{N}}$ be an infinite sequence of tableaux such that, for every $i \in \mathbb{N}$, $\mathcal{T}_{i+1}$ is obtained from $\mathcal{T}_i$ by applying one of the Expansion rules of Section 2, other than the Loop Detection rule. Assume, moreover, that for every $k \in \mathbb{N}$, there exists $n \in \mathbb{N}$ such that every non-closed leaf in $\mathcal{T}_n$ is of a rank greater than k (i.e. no branch is indefinitely "frozen", the rank of the leaves increases indefinitely). There exists $n \in \mathbb{N}$ such that $\mathcal{T}_n$ is globally looping.*

We now show that from every tableau $\mathcal{T}$, one can extract a resolution derivation from the root of $\mathcal{T}$ of the disjunction of the leaves of $\mathcal{T}$. We first restrict ourselves to tableaux built without the Explosion and Loop Detection rules. We focus on such tableaux because they correspond to the subtrees that are found "between" two layers in an tableau built without restriction on the rules. More precisely, take a layer α of some rank m in a tableau $\mathcal{T}$ (built without restriction on the rules). Then the subtree of $\mathcal{T}$ of root α and whose leaves are the layers of rank $m+1$ below α is indeed a tree built without Explosion nor Loop Detection (by definition of a layer).

We first build derivations for such subtrees, those derivations will then be used as the base elements for building the final schema of refutation. For such a tree $\mathcal{T}$ and a node α of $\mathcal{T}$, the next definition introduces $\Delta(\mathcal{T},\alpha)$, which is intended to be a derivation of $\mathrm{cnf}(\mathcal{T}[\mathrm{Leaves}(\mathcal{T},\alpha)])$ from $\mathrm{cnf}(\mathcal{T}(\alpha))$.

Definition 16 Let $\mathcal{T}$ be a tableau constructed using the Expansion rules, *except* the Explosion and Loop Detection rules. Let α be a node in $\mathcal{T}$. We define a derivation $\Delta(\mathcal{T},\alpha)$ inductively, according to the rule that is applied on α:

- If α is a leaf, then $\Delta(\mathcal{T},\alpha)$ is defined as the sequence of clauses in $\mathrm{cnf}(\mathcal{T}(\alpha))$.
- If the Closure rule is applied on α, using two formulæ ϕ and $\neg\phi$, then $\Delta(\mathcal{T},\alpha) \overset{\mathrm{def}}{=} \phi \cdot \neg\phi \cdot \bot$ (notice that since the formulæ are in NNF, ϕ must be an atom).
- If the Normalisation, Purity or $\wedge$-Decomposition rule is applied on α, yielding a node β then $\Delta(\mathcal{T},\alpha) \overset{\mathrm{def}}{=} \Delta(\mathcal{T},\beta)$.
- Finally, assume that the $\vee$-Decomposition rule is applied on α yielding two nodes β_1 and β_2. Let Φ_1 and Φ_2 be the clausal forms of ϕ_1 and ϕ_2 respectively. For any $C \in \Phi_2$, let $\Lambda'(C)$ be the derivation obtained from $\Delta(\mathcal{T},\beta_1)$ by replacing every occurrence of a clause $D \in \Phi_1$ by $D \vee C$ (and by adding the disjunction $\vee C$ to every descendant of D).

  For any clause C' in $\mathrm{cnf}(\mathcal{T}[\mathrm{Leaves}(\mathcal{T},\beta_1)])$, we construct a derivation $\Lambda''(C')$ from $\Delta(\mathcal{T},\beta_2)$ by replacing every occurrence of a clause $D \in \Phi_2$ by $D \vee C'$ (and by adding the disjunction $\vee C'$ to every descendant of D). Then $\Delta(\mathcal{T},\alpha)$ is the concatenation of all the derivations $\Lambda'(C)$ and $\Lambda''(C')$ (with $C \in \Phi_2$ and $C' \in \mathrm{cnf}(\mathcal{T}[\mathrm{Leaves}(\mathcal{T},\beta_1)])$).

Only the case of disjunction is non-trivial. Informally, it does nothing more than building, for two sets of clauses S_1 and S_2, a derivation of $\mathrm{cnf}(S_1 \vee S_2)$ from two derivations of S_1 and S_2.

Thus the function $\mathcal{T}(\alpha) \to \Delta(\mathcal{T},\alpha)$ allows us to build derivations from subtrees of a whole tableau. Intuitively, the next step is to put together those derivations according to the positions of the corresponding subtrees in the main tableau. Consider a rank m in a tableau $\mathcal{T}$. One can apply the function Δ to all the (parallel) subtrees whose root is a layer of rank m. Then we can do the same at rank $m+1$, append every resulting derivation to the derivation obtained from the parent tree, and go on at rank $m+2$, etc. This intuitively gives the structure of a rewrite system where n decreases each time we go to the next rank. However this gives us a tree-like structure (to every derivation corresponding to a subtree $\mathcal{U}$ we append the derivations corresponding to all the leaves of $\mathcal{U}$, and go on with the trees below those leaves) similar to the rewrite systems presented in Section 3. Instead we would like a more linear structure. So we will consider *at once* all the layers of a given rank and get only one derivation corresponding to those nodes. For this, we need a way to apply Δ to all the subtrees at once. This is actually done by building a new tableau from the subtrees.

Let $\mathcal{T}$ be a tableau of root α. Assume that $\mathcal{T}$ is globally looping w.r.t. n and k, with $n < k$. Let $m < k$. We denote by $\mathcal{U}(\mathcal{T}, m)$ a tableau whose root is labeled by a formula $\mathcal{T}[\text{Layers}(\mathcal{T}, m)]$ (note that we take *all the layers* of rank m at a time), and obtained by applying the $\vee$ and $\wedge$-Decomposition and Closure rules (and only these rules) until irreducibility. By definition, since the root formula of $\mathcal{U}(\mathcal{T}, m)$ is the disjunction of the labels of the layers in $\text{Layers}(\mathcal{T}, m)$, every non-closed leaf β of $\mathcal{U}(\mathcal{T}, m)$ is labeled by a set of formulæ of the form $\mathcal{T}(\gamma_\beta)$, where $\gamma_\beta \in \text{Layers}(\mathcal{T}, m)$. Furthermore, for every $\gamma \in \text{Layers}(\mathcal{T}, m)$, there exists a leaf β of $\mathcal{U}(\mathcal{T}, m)$ such that $\gamma_\beta = \gamma$. Since $m < k$ and since by Definition 14 the leaves of $\mathcal{T}$ must be of a rank greater or equal to k, the node γ_β cannot be a leaf of $\mathcal{T}$. This implies that some rule is applied on γ_β. But the only rule that is applicable on a layer (beside the Global Loop Detection rule that cannot be applied on layers of a rank distinct from k) is the Explosion rule. Hence $\mathcal{T}$ necessarily contains two subtableaux, written $\mathcal{T}_\beta^0$ and $\mathcal{T}_\beta^1$, of roots $\mathcal{T}(\gamma_\beta)\{\mathtt{n} \leftarrow 0\}$ and $\mathcal{T}(\gamma_\beta)\{\mathtt{n} \leftarrow \mathtt{n}+1\}$ respectively. Then $\mathcal{V}^0(\mathcal{T}, m)$ and $\mathcal{V}^1(\mathcal{T}, m)$ denote respectively the tableaux obtained from $\mathcal{U}(\mathcal{T}, m)\{\mathtt{n} \leftarrow 0\}$ and $\mathcal{U}(\mathcal{T}, m)\{\mathtt{n} \leftarrow \mathtt{n}+1\}$ by:

- Replacing every leaf β by $\mathcal{T}_\beta^0$ and $\mathcal{T}_\beta^1$ respectively.
- Removing, in the obtained tableau, all applications of the Explosion rule[3] (and all the nodes that occur below such an application).

By applying the above function $\Delta(\mathcal{T}, \alpha)$ on the two tableaux $\mathcal{V}^1(\mathcal{T}, m)$ and $\mathcal{V}^0(\mathcal{T}, m)$, we define the following derivations (where α denotes the root of $\mathcal{V}^1(\mathcal{T}, m)$ and $\mathcal{V}^0(\mathcal{T}, m)$):

$$\Lambda^1(\mathcal{T}, m) \stackrel{\text{def}}{=} \Delta(\mathcal{V}^1(\mathcal{T}, m), \alpha) \qquad \Lambda^0(\mathcal{T}, m) \stackrel{\text{def}}{=} \Delta(\mathcal{V}^0(\mathcal{T}, m), \alpha)$$

Let $\mathcal{T}$ be a tableau that is globally looping w.r.t. two numbers $n < k$. We associate to each natural number $m < k$ a symbol γ^m. Let $\mathfrak{R}^\star(\mathcal{T})$ the system containing the following rules. Note that $\mathcal{V}^0(\mathcal{T}, m)$ and $\mathcal{V}^1(\mathcal{T}, m)$ are defined only w.r.t. the rank m, but not w.r.t. a particular node. Thus, contrarily to the transformation of Section 3, there is not one derivation per node, but rather one derivation per rank.

$$\gamma_0^m \to \Lambda^0(\mathcal{T}, m) \qquad \gamma_{\mathtt{n}+1}^m \to \Lambda^1(\mathcal{T}, m) \cdot \gamma_{\mathtt{n}}^{m+1} \text{ (if } m+1 < k) \qquad \gamma_{\mathtt{n}+1}^{k-1} \to \Lambda^1(\mathcal{T}, k) \cdot \gamma_{\mathtt{n}}^{n}$$

[3]Note that, although no application of the Explosion rule occurs in $\mathcal{U}(\mathcal{T}, m)$, some applications of this rule may occur in $\mathcal{T}_\beta^1$.

Intuitively, we are appending the derivations, rank after rank, until we reach the rank k where the Global Loop Detection applies. In this case we get back at the rank of looping n. Thus we can see the use of grouping the derivations by rank (instead of node) as it allows to benefit from the simplified form of looping induced by the Global Loop Detection rule. In the end, the resulting rewrite system is indeed much simpler.

Proposition 17 $\mathfrak{R}^\star(\mathcal{T})$ *is convergent.*

Note that, by definition, $\mathfrak{R}^\star(\mathcal{T})$ is always propositional (unlike $\mathfrak{R}(\mathcal{T})$).

Theorem 18 *Let $\mathcal{T}$ be a tableau of root α that is globally looping w.r.t. two numbers n, k, with $n < k$. Let $m < k$. For all $i \in \mathbb{N}$, $\gamma_i^m{\downarrow}_{\mathfrak{R}^\star(\mathcal{T})}$ is a refutation of $cnf(\mathcal{T}[Layers(\mathcal{T}, m)])\{\mathtt{n} \leftarrow i\}{\downarrow}_{\mathcal{R}}$. Thus in particular, if α is a layer, $\gamma_i^0{\downarrow}_{\mathfrak{R}^\star(\mathcal{T})}$ is a refutation of $\mathcal{T}(\alpha)\{\mathtt{n} \leftarrow i\}{\downarrow}_{\mathcal{R}}$.*

When α is not a layer, the rewrite system is easily adapted by prepending the derivation obtained by applying Δ to the subtree of $\mathcal{T}$ whose leaves are the layers of rank 0.

Example 19 Consider the tableau of Example 13. This tableau is actually globally looping. The following rewrite system is constructed (after partial evaluation and simplification):

$$\begin{array}{lll}
\gamma_0 & \rightarrow & p_0 \vee q_0 \cdot \neg p_0 \cdot q_0 \cdot \neg q_0 \cdot \bot \\
\gamma_{\mathtt{n}+1} & \rightarrow & (p_{\mathtt{n}+1} \vee q_{\mathtt{n}+1}) \cdot (q_{\mathtt{n}} \vee \neg p_{\mathtt{n}+1}) \cdot (q_{\mathtt{n}} \vee q_{\mathtt{n}+1}) \cdot (p_{\mathtt{n}} \vee \neg q_{\mathtt{n}+1}) \cdot (q_{\mathtt{n}} \vee p_{\mathtt{n}}) \cdot \gamma_{\mathtt{n}}
\end{array}$$

Compared with the system produced by the previous method (see Example 13), these rules are obviously simpler (no schema variable are needed, and only linear recursion is used). Furthermore, it is easy to check that they generate much shorter derivations. ♣

5 Conclusion

Two distinct algorithms have been designed for extracting schemata of resolution proofs from closed tableaux. This work is motivated by the fact that such refutations are needed for some natural applications of schemata calculus (unsatisfiability detection is not always sufficient). In particular, the explicit generation of the proofs (even in the form of proof schemata) makes possible the certification of the results produced by the provers. The first algorithm tackles the tableau calculus in its full generality, but it yields very complex representations of the derivations (which will make them less usable in practice, in particular they are not very informative for a human user). The second one uses a less powerful calculus, but it generates schemata of refutations in a much simpler format (propositional rewrite systems are obtained).

There is thus a natural trade-off between the two presented methods: none of them is uniformly superior to the other. The choice between the two algorithms should be made according to the considered applications, and/or to the form of the constructed tableaux. In some cases, as shown by the examples in Section 3, the first approach generates a propositional rewrite system. In this case it should of course be preferred. Future work includes the implementation of the two methods and the precise evaluation of the complexity of the second algorithm. One could also wonder whether a polynomial algorithm generating propositional derivations exists for the general case. We conjecture that the use of Δ-variables cannot be avoided in general.

References

[1] ASAP: About schemata and proofs. ANR-FWF Research project, 2010–2013. http://membres-lig.imag.fr/peltier/ASAP/.

[2] V. Aravantinos, R. Caferra, and N. Peltier. A schemata calculus for propositional logic. In *TABLEAUX 09 (International Conference on Automated Reasoning with Analytic Tableaux and Related Methods)*, volume 5607 of *LNCS*, pages 32–46. Springer, 2009.

[3] V. Aravantinos, R. Caferra, and N. Peltier. A Decidable Class of Nested Iterated Schemata. In *IJCAR 2010 (International Joint Conference on Automated Reasoning)*, LNCS, pages 293–308. Springer, 2010.

[4] V. Aravantinos, R. Caferra, and N. Peltier. Decidability and undecidability results for propositional schemata. *Journal of Artificial Intelligence Research*, 40:599–656, 2011.

[5] M. Baaz and A. Leitsch. Cut-elimination and Redundancy-elimination by Resolution. *Journal of Symbolic Computation*, 29(2):149–176, 2000.

[6] O. Bar-Ilan, O. Fuhrmann, S. Hoory, O. Shacham, and O. Strichman. Linear-time reductions of resolution proofs. In *Proceedings of HVC'08*.

[7] P. Beame, H. Kautz, and A. Sabharwal. Towards understanding and harnessing the potential of clause learning. *Journal of Artificial Intelligence Research*, 22:319–351, 2004.

[8] A. Cimatti, A. Griggio, and R. Sebastiani. A simple and flexible way of computing small unsatisfiable cores in SAT modulo theories. In *SAT*, LNCS, pages 334–339. Springer, 2007.

[9] A. Leitsch. *The resolution calculus*. Springer. Texts in Theoretical Computer Science, 1997.

[10] K. L. McMillan and N. Amla. Automatic abstraction without counterexamples. In *Proceedings of TACAS'03*, pages 2–17. Springer-Verlag, 2003.

[11] G. Mints. Proof theory in the USSR 1925-1969. *J. Symb. Log.*, 56(2):385–424, 1991.

[12] R. M. Smullyan. *First-Order Logic.* Springer, 1968.

[13] A. Wolf. Optimization and translation of tableau-proofs into resolution. *Journal of Information Processing and Cybernetics*, 30(5/6):311–325, 1994.

[14] L. Zhang and S. Malik. Validating SAT Solvers Using an Independent Resolution-Based Checker: Practical Implementations and Other Applications. In *DATE*, pages 10880–10885. IEEE Computer Society, 2003.

Unification in Blind Signatures

Serdar Erbatur* ^{1}, Christopher Lynch † ^{2}, and Paliath Narendran‡ ^{1}

^{1} University at Albany–SUNY {se,dran}@cs.albany.edu
^{2} Clarkson University clynch@clarkson.edu

Abstract

Blind signatures are signature schemes that keep the content confidential and have applications in modern cryptography for electronic voting and digital cash schemes. We study three unification problems based on an equational theory for blind signatures. This theory consists of two axioms, namely

$$\begin{array}{rcll} U(S(B(m,x)),x) & = & S(m) & (\mathcal{E}_1) \\ m * B(n,r) & = & B(m*n,r) & (\mathcal{E}_2) \end{array}$$

derived from its implementation with RSA. First, the unification problem modulo $\mathcal{E}_1$ is shown to be NP-complete and of type finitary. An algorithm based on deduction rules is given. Second, unification in $\mathcal{E}_2$ is shown to be decidable and of type unitary. Likewise, we give an algorithm which returns a unique unifier if there exists one and provide necessary failure rule mechanisms to detect function clash, occur-check and infinite splitting. Finally, the combination of unification problems $\mathcal{E}_1$ and $\mathcal{E}_2$ turns out to be decidable. The result follows from techniques of equational term rewriting systems and unification in the subtheories $\mathcal{E}_1$ and $\mathcal{E}_2$. Consequently, these results are useful for symbolic analysis of protocols deploying blind signatures.

1 Introduction

In formal cryptographic protocol analysis, messages are represented as terms, where the functions in the terms represent actions that can be performed on messages, such as encrypting a message with some key, hashing a message, or calculating the exclusive OR of two messages.

A protocol is described formally by the actions of a principal, who will receive a message, and then send out another message based on the message received. An attack can be represented by the intruder learning some secret. A tool for cryptographic protocol analysis can then work its way back from the goal to initial facts, to see if a sequence of actions which leads to the intruder learning

*Partially supported by the NSF grants CNS-0831209 and CNS-0905286
†Partially supported by the NSF grants CNS-0831305 and CNS-0905378
‡Partially supported by the NSF grants CNS-0831209 and CNS-0905286

the secret message is possible. At each stage of this search, unification must be performed between terms representing messages sent and terms representing messages that are expected to be received. These terms may contain variables representing unknowns.

Traditionally, cryptographic protocol analysis tools work in the free algebra, which gives no meaning to the function symbols, and terms can only be equal if they are syntactically the same. However, tools such as the Maude NPA [7] can give a deeper analysis. Equational properties of terms can be given, which take into account the meaning of a function symbol. Then unification can be performed modulo the equational theory.

For example, consider the case of blind signatures. There is a blinding function B, which blinds a message m with a given key x. We can represent this by the term $B(m, x)$. There is an unblinding function U which performs the inverse of unblinding for some key. There is also a signing function. Blind signatures have the property that if a message m is blinded with some key x, then signed, and then unblinded with x, the signed message will emerge. These are called blind signatures, because the signer could not tell what was being signed. Blind signatures are used in electronic voting and digital cash [5]. The properties we have just described can be represented by the following equation:

$$U(S(B(m,x)),x) = S(m)$$

Blind RSA signatures [4] are created by multiplying the message m by a random number r raised to a public key e. Therefore, multiplying an RSA-blinded message by another number (message) is equivalent to multiplying the two messages and then blinding them.

The two properties mentioned above are represented by the following axioms:

$$U(S(B(m,x)),x) = S(m) \tag{1}$$

$$m * B(n,r) = B(m*n,r) \tag{2}$$

We denote axiom (1) as equational theory $\mathcal{E}_1$ and axiom (2) as $\mathcal{E}_2$.

Performing unification modulo $\mathcal{E}_1$ and $\mathcal{E}_2$ will allow a cryptographic analysis tool to give a deeper analysis of a cryptographic protocol which uses blinding. Therefore, in this paper we give unification algorithms for $\mathcal{E}_1$, $\mathcal{E}_2$ and the theory consisting of both of them.

The algorithms given are based on inference rules originally given in [14], which gave a unification algorithm for one-sided distributivity. We give an algorithm to generate a complete set of unifiers for each of these theories. The algorithm for $\mathcal{E}_1$ runs in nondeterministic polynomial time and gives a finite complete set of unifiers. The algorithm for $\mathcal{E}_2$ gives a single most general unifier, and the algorithm for the combination of the two also gives a finite complete set of unifiers.

We describe the algorithm for $\mathcal{E}_1$ in Section 3, for $\mathcal{E}_2$ in Section 4, and for the combination in Section 5.

2 Preliminaries

We introduce some basic definitions here. The reader is referred to the survey [2] for more details.

Let $\mathcal{S} = \{s_1 =^?_E t_1, \ldots, s_m =^?_E t_m\}$ be an E-unification problem. An *E-unifier* for $\mathcal{S}$ is a substitution σ such that $\sigma(s_i) =_E \sigma(t_i)$ for all $1 \leq i \leq m$. That is, *equality modulo E* , $=_E$, in $\mathcal{S}$ is satisfied if we apply σ to every equation. The set of all E-unifiers of $\mathcal{S}$ is denoted by $U_E(\mathcal{S})$. It is said that σ is *more general modulo E* than θ on a set of variables V, denoted as $\sigma \leq_E \theta$, if and only if there is a substitution τ such that $\sigma\tau(x) =_E \theta(x)$ for all $x \in V$. A *complete set of E-unifiers* of $\mathcal{S}$ is a set Σ of substitutions such that every $\theta \in \Sigma$ is an E-unifier and for every E-unifier θ, there is a substitution $\sigma \in \Sigma$ where $\sigma \leq_E \theta$ holds. A complete set of E-unifiers Σ of a unification problem $\mathcal{S}$ is said to be *minimal* if and only if for any two E-unifiers σ and θ in Σ, $\sigma \leq_E \theta$ implies that $\sigma =_E \theta$.

An E-unification problem $\mathcal{S}$ is of type *unitary*, if the minimal complete set of E-unifiers of $\mathcal{S}$ has size one. $\mathcal{S}$ is *finitary (infinitary)* if the minimal complete set of E-unifiers of it is finite (infinite). We note that the minimal set of unifiers might be empty even when the problem is unifiable. We say $\mathcal{S}$ is of type *zero* in that case. An equational theory E is *unitary* if the maximal type of an E-unification problem is unitary. Similarly, E is *finitary* if E-unification problems have at most type finitary. If there exists a problem of type infinitary on E and no problem of type nullary, then E is *infinitary.* E has type *zero* (or E is *nullary*) if it has a problem of type zero.

A set of equations (i.e., a unification problem) is said to be in *dag-solved form* (or *d-solved form*) if and only if they can be arranged as a list

$$x_1 =^? t_1, \ \ldots, \ x_n =^? t_n$$

where (a) each left-hand side x_i is a distinct variable, and (b) $\forall\, 1 \leq i \leq j \leq n$: x_i does not occur in t_j ([8]). It is not hard to see that a unification problem in *dag*-solved form has a unique most general unifier which can be obtained in a straightforward way [8]. If a set of equations $\mathcal{EQ}$ is in dag-solved form, we say that $\mathcal{EQ}$ is *solved.*

A rewrite rule is an ordered pair (l, r) of terms such that the variables in r also appear in l. It is often written as $l \rightarrow r$. A rewrite system $\mathcal{R}$ is set of rewrite rules (l, r). Let $\mathcal{R}$ be a rewrite system and E a set of equations. We define *extended rewriting* with $\mathcal{R}$ modulo E, expressed as

$$s \rightarrow_{E \backslash \mathcal{R}} t,$$

if and only if there exist a rule $l \rightarrow r$ in $\mathcal{R}$ and a position p in s such that $s|_p \leftrightarrow^*_E \sigma(l)$, $t = s\left[\sigma(r)\right]_p$ for some substitution σ. See [9] and [3] for detailed expositions of equational rewriting.

A rewrite rule $l \rightarrow r$ is *optimally reducing*[1] if and only if for any substitution θ for which $\theta(r)$ is R-reducible, there is a *proper* subterm s of l such that $\theta(s)$ is R-reducible. A rewrite system R is *optimally reducing* iff every rule in it is optimally reducing modulo R.

3 Unification in $\mathcal{E}_1$

We show that $\mathcal{E}_1$-unification is NP-Complete and give a nondeterministic algorithm for it.

[1]For term rewriting systems this notion was first introduced in [12], and has been generalized in [6].

To show NP-Hardness, the NP-complete problem monotone **1-in-3 3SAT** will be polynomially reduced to $\mathcal{E}_1$-unifiability.

The definition of monotone **1-in-3 3SAT** is as follows:

Given: A set of clauses $C = \{c_1, \ldots, c_n\}$ where each clause has exactly three propositional variables.

Question: Is there a satisfying assignment such that exactly one variable is set to *true* in each clause?

Let $C = c_1 \wedge \cdots \wedge c_m$ be an instance of the **1-in-3 3SAT** problem and $V = \{u_1, \ldots, u_n\}$ be variables occurring in C, i.e., $V = Var(C)$. We show how to construct an instance S of the $\mathcal{E}_1$-unification problem from C such that S is unifiable if and only if C is satisfiable.

First of all, we define ground terms a_1, a_2, a_3 as follows:

$$\begin{aligned} a_1 &= B(B(1,0),0) \\ a_2 &= B(B(0,1),0) \\ a_3 &= B(B(0,0),1) \end{aligned}$$

For any clause $c_i = (u_{i_1}, u_{i_2}, u_{i_3})$, $u_{i_j} \in V$, $i = 1, \ldots, m$ and $j = 1, 2, 3$, we introduce a term $T_i = B(B(u_{i_1}, u_{i_2}), u_{i_3})$.

In addition, auxiliary variables x_i, y_i, z_i and a constant m are created. The equation constructed for c_i is

$$U(S(B(m, U(S(B(m, y_i)), a_3))), U(S(B(m, a_1)), T_i)) =^?_{\mathcal{E}_1} \\ U(S(B(m, x_i)), U(S(B(m, z_i)), a_2)).$$

We note that separate variables x_i, y_i, z_i, which are also pairwise distinct, are created for each clause c_i. To follow the results more easily, we define $t_{i,1}$, $t_{i,2}$ and $t_{i,3}$.

$$\begin{aligned} t_{i,1} &= U(S(B(m, a_1)), T_i) \\ t_{i,2} &= U(S(B(m, y_i)), a_3) \\ t_{i,3} &= U(S(B(m, z_i)), a_2) \end{aligned}$$

Therefore, we can now rewrite the equation into the following form:

$$U(S(B(m, t_{i,2})), t_{i,1}) =^?_{\mathcal{E}_1} U(S(B(m, x_i)), t_{i,3})$$

Obviously, we in general obtain a set of equations rather than one equation from a given **1-in-3 3SAT** instance C. Let us denote this set by S.

Lemma 3.1. *S is unifiable if and only if C is satisfiable.*

Proof. If C is satisfiable, S is unified trivially. Each clause is assigned to one of (1,0,0) or (0,1,0) or (0,0,1). We simply unify corresponding a_k $(k = 1, 2, 3)$ with T_i's in each equation. For instance, if $T_i =^?_{\mathcal{E}_1} a_1$, then $t_{i,1} =^?_{\mathcal{E}_1} S(m)$ and the solution follows from assigning a_3 to y_i and $t_{i,3}$ to x_i. Similar for $T_i =^?_{\mathcal{E}_1} a_2$ and $T_i =^?_{\mathcal{E}_1} a_3$.

Conversely, let S be unified by following the settings above. For each clause c_i in C, it is straightforward to verify that the equation is satisfied if and only if $t_{i,1} =^?_{\mathcal{E}_1} t_{i,2}$ or $t_{i,1} =^?_{\mathcal{E}_1} t_{i,3}$. One of these equations is satisfied when and only when exactly one of $T_i =^?_{\mathcal{E}_1} a_1$ or $T_i =^?_{\mathcal{E}_1} a_2$ or $T_i =^?_{\mathcal{E}_1} a_3$ is unified. By definition, there is only one variable u_{i_j} in T_i assigned to 1 in the solution. We can set the corresponding variable to *true* in each clause of C to build a 1-in-3 satisfying assignment. □

Thus, we finally obtain

Theorem 3.2. *$\mathcal{E}_1$-unification is NP-complete.*

Proof. NP-hardness follows from the previous lemma. Membership in NP follows from the fact that the term rewriting system

$$U(S(B(m,x)),x) \;\rightarrow\; S(m)$$

is optimally reducing and convergent. $\square$

Since unification modulo convergent, optimally reducing term rewriting systems is finitary[2] we get

Theorem 3.3. *$\mathcal{E}_1$-unification is finitary, and there is an algorithm for computing a complete set of $\mathcal{E}_1$-unifiers.*

Proof. An alternative proof would be to observe that $\mathcal{E}_1$ is saturated under ordered paramodulation and then use the result in [10] or [13]. $\square$

However, we also show this in the next section by devising a new algorithm.

3.1 Algorithm

In this section we outline a nondeterministic algorithm for the general $\mathcal{E}_1$-unification problem which we plan to implement. In addition, this algorithm returns a complete set of unifiers for a given problem. We assume, without loss of generality, that each equation is in one of the following standard forms:

1. $X =^? V$
2. $X =^? U(V,Y)$
3. $X =^? B(V,Y)$
4. $X =^? S(V)$
5. $X =^? f(V_1,\ldots,V_n)$

In this setting X, V, $V_1,\ldots,V_n$ and Y are variables and f is an uninterpreted function symbol with arity n.

Transformation rules are created based on the equation forms we specified. Note that rules (h1) and (h2) are nondeterministic and applied "most lazily." The goal is to transform the given set of equations to dag-solved form.

[2]Strictly speaking, this is not shown in [12]. But it is not hard to show, see [6].

(a) $$\frac{\{X =^? V\} \uplus \mathcal{EQ}}{\{X =^? V\} \cup [V/X](\mathcal{EQ})} \qquad \text{if } X \text{ occurs in } \mathcal{EQ}$$

(b) $$\frac{\mathcal{EQ} \uplus \{X =^? B(V,Y),\ X =^? B(W,T)\}}{\mathcal{EQ} \cup \{X =^? B(V,Y),\ V =^? W,\ Y =^? T\}}$$

(c) $$\frac{\mathcal{EQ} \uplus \{X =^? S(V),\ X =^? S(W)\}}{\mathcal{EQ} \cup \{X =^? S(V),\ V =^? W\}}$$

(d) $$\frac{\mathcal{EQ} \uplus \{X =^? U(V,Y),\ V =^? S(W'),\ W' =^? B(W,Y))\}}{\mathcal{EQ} \cup \{X =^? S(W),\ V =^? S(W'),\ W' =^? B(W,Y))\}}$$

(e) $$\frac{\mathcal{EQ} \uplus \{X =^? U(V,Y),\ X =^? S(W)\}}{\mathcal{EQ} \cup \{X =^? S(W),\ V =^? S(W'),\ W' =^? B(W,Y))\}}$$

(f) $$\frac{\mathcal{EQ} \uplus \{X =^? U(V,Y),\ X =^? U(W,Y)\}}{\mathcal{EQ} \cup \{V =^? W,\ X =^? U(W,Y)\}}$$

(g) $$\frac{\mathcal{EQ} \uplus \{X =^? U(V,Y),\ X =^? U(V,T)\}}{\mathcal{EQ} \cup \{X =^? U(V,Y),\ Y =^? T\}}$$

(h1) $$\begin{array}{c} \mathcal{EQ} \uplus \{X =^? U(Y,Z)\} \\ \vdots \ \text{if } \mathcal{EQ} \uplus \{X =^? U(Y,Z)\} \text{ is not solved} \\ \mathcal{EQ} \cup \{Y =^? S(Y'),\ Y' =^? B(M,Z),\ X =^? S(M)\} \end{array}$$

(h2) $$\frac{\mathcal{EQ} \uplus \{X =^? U(V,W),\ X =^? U(Y,Z)\}}{\mathcal{EQ} \cup \{X =^? U(Y,Z),\ V =^? Y,\ W =^? Z\}}$$

Variables Y', M in rule (h1) and W' in rule (e) are fresh variables.

For uninterpreted function symbols, we have

(i) $$\frac{\mathcal{EQ} \uplus \{X =^? f(V_1,\ldots,V_n),\ X =^? f(W_1,\ldots,W_n)\}}{\mathcal{EQ} \cup \{X =^? f(V_1,\ldots,V_n),\ V_1 =^? W_1,\ \ldots,\ V_n =^? W_n\}}$$

We use rule (a) to eliminate a variable V from the rest of the system. By rules (b), (c), (f), (g) and (i), we remove function symbols from the problem, i.e., narrow the equations. Right after applying those rules, we apply rule (a) to the resulting equations for variable elimination. The soundness of rules (b) – (h2) follows from axiom (1).

Rule (a) is applied most eagerly, followed by the cancellation rules (b), (c), (f), (g) and (i), then (d) and (e) in that order of priority. As mentioned earlier, *the nondeterministic rules (h1) and (h2) have the lowest priority.*

We have the following failure rules:

$$\text{(F1)} \qquad \frac{\mathcal{EQ} \ \uplus \ \{X =^? U(V,Y),\ X =^? B(W,T)\}}{FAIL}$$

$$\text{(F2)} \qquad \frac{\mathcal{EQ} \ \uplus \ \{X =^? B(V,Y),\ X =^? S(W)\}}{FAIL}$$

We also add a failure rule, which is applied when at least one of f and g is an uninterpreted function symbol.

$$\text{(F3)} \qquad \frac{\mathcal{EQ} \ \uplus \ \{X =^? f(V_1,\ldots,V_m),\ X =^? g(W_1,\ldots,W_n)\}}{FAIL} \qquad \text{if } (f \neq g)$$

These rules could be combined into

$$\text{(F4)} \qquad \begin{array}{c} \mathcal{EQ} \ \uplus \ \{X =^? f(V_1,\ldots,V_m),\ X =^? g(W_1,\ldots,W_n)\} \\ \vdots \\ FAIL \end{array} \quad \text{if } (f \neq g) \text{ and } \{f,g\} \neq \{U,S\}$$

Another kind of failure is occur-check which can be implemented as an extended cycle check as done in algorithms for standard unification. (This can be defined similar to the failure rule (F2) in the next section.) In the presence of the nondeterministic rules (h1) and (h2) this is enough to catch all failures. For instance, consider $X =^? U(Y,X)$. If (h1) is not applied at all, this would cause occur-check failure. But once (h1) is applied we get the set of equations $\{Y =^? S(Y'),\ Y' =^? B(M,X),\ X =^? S(M)\}$ which is unifiable.

Termination can be shown by using the following measure

$m(S)$ = (number of occurrences of the symbol U, number of unsolved variables)

The first component decreases in all applications of rules (d) through (h2). Furthermore, it does not increase in rules (a)-(c) and (i). The second component clearly decreases in the case of rule (a); it also decreases for rules (b), (c) and (i), provided that rule (a) is applied immediately afterwards. Since (a) is applied most eagerly, this follows.

Theorem 3.4. *Rules (a)-(i) terminate.*

4 Unification in $\mathcal{E}_2$

We describe an algorithm by using transformation rules, as we did for $\mathcal{E}_1$ in Section 3.1. Furthermore, the algorithm returns a most general unifier if the input equations are unifiable. Without loss of generality, equations will be in one of these forms:

$$X =^? V,\ X =^? B(V,Y),\ X =^? V * Y,\ X =^? f(V_1,\ldots,V_n)$$

(U, V, $V_1,\ldots,V_n$ and Y are variables and f is an uninterpreted function symbol with arity n.)

Both of the function symbols, B and $*$, are cancellative. Note that we do not assume that B or $*$ is commutative or associative.

(a) $$\frac{\{X =^? V\} \ \uplus\ \mathcal{EQ}}{\{X =^? V\} \cup [V/X](\mathcal{EQ})} \qquad \text{if } X \text{ occurs in } \mathcal{EQ}$$

(b) $$\frac{\mathcal{EQ}\ \uplus\ \{X =^? B(V,Y),\ X =^? B(W,T)\}}{\mathcal{EQ}\ \cup\ \{X =^? B(V,Y),\ V =^? W,\ Y =^? T\}}$$

(c) $$\frac{\mathcal{EQ}\ \uplus\ \{X =^? V * Y,\ X =^? W * T\}}{\mathcal{EQ}\ \cup\ \{X =^? V * Y,\ V =^? W,\ Y =^? T\}}$$

(d) $$\frac{\mathcal{EQ}\ \uplus\ \{U =^? B(X,Y),\ U =^? U_1 * U_2\}}{\mathcal{EQ}\ \cup\ \{X =^? U_1 * Z,\ U_2 =^? B(Z,Y),\ U =^? U_1 * U_2\}}$$

Rule (d) (the "splitting rule") introduces a fresh variable Z.

To handle uninterpreted functions, we add the same rules as in the case of $\mathcal{E}_1$.

(e) $$\frac{\mathcal{EQ}\ \uplus\ \{X =^? f(V_1,\ldots,V_n),\ X =^? f(W_1,\ldots,W_n)\}}{\mathcal{EQ}\ \cup\ \{X =^? f(V_1,\ldots,W_n),\ V_1 =^? W_1,\ \ldots,\ V_n =^? W_n\}}$$

A standard failure rule for function clash is:

(F1) $$\begin{array}{c}\mathcal{EQ}\ \uplus\ \{X =^? f(V_1,\ldots,V_m),\ X =^? g(W_1,\ldots,W_n)\}\\ \vdots \\ FAIL\end{array} \quad \text{if } (f \neq g) \text{ and } \{f,g\} \neq \{B,*\}$$

The outline of the algorithm is as follows: As long as rules are applicable, rules (a) and (F1) are applied most eagerly, and the cancellative rules (b), (c) and (e) come next. The splitting rule (d) is applied, if necessary, at the end, i.e., rule (d) has the lowest priority.

The proof of correctness for this algorithm is similar to the one in Tiden-Arnborg [14].

We define the following relations between terms.

- $U \succ_{r_*} V$ iff there is an equation $U = T * V$
- $U \succ_{l_*} V$ iff there is an equation $U = V * T$

- $U \succ_{r_B} V$ iff there is an equation $U = B(T, V)$
- $U \succ_{l_B} V$ iff there is an equation $U = B(V, T)$
- $U \succ_{*} V$ iff $U \succ_{r_*} V$ or $U \succ_{l_*} V$
- $U \succ_{B} V$ iff $U \succ_{r_B} V$ or $U \succ_{l_B} V$
- $U \succ_{f} V$ iff there is an equation $U = f(\ldots, V, \ldots)$, where f is uninterpreted.

Let $\succ\, = \,\succ_{r_*} \cup \succ_{l_*} \cup \succ_{r_B} \cup \succ_{l_B} \cup \succ_f$, i.e., the union of the four relations above. Thus, each of these relations is a subrelation of $\succ$. Alternatively, $\succ\, = \,\succ_* \cup \succ_B \cup \succ_f$.

We define an extended occur-check failure rule using $\succ$.

(F2) $\dfrac{\mathcal{EQ}}{FAIL}$ if $X \succ^+ X$ for some X

Let $\sim_{rp(*)}$, and $\sim_{lp(B)}$ be the reflexive, symmetric and transitive closures for $\succ_{r_*}$ and $\succ_{l_B}$, respectively.

We also define a set of relations $\beta = \{\beta_1, \beta_2\}$ where

- $\beta_1 = \sim_{rp(*)} \circ \succ_B \circ \sim_{rp(*)}$
- $\beta_2 = \sim_{lp(B)} \circ \succ_* \circ \sim_{lp(B)}$.

One can define two interpretations for $\mathcal{E}_2$, namely interpreting B as left and $*$ as right projections. We denote these interpretations as projection functions symbols $rp(*)$, and $lp(B)$. For instance, if we interpret $*$ as right projection by $rp(*)$, then the axiom $m * B(n, r) = B((m * n), r)$ is trivially satisfied. The same holds if we take B as left projection.

Both interpretations give valid models for the theory. That is, if a problem is solvable modulo $\mathcal{E}_2$, it is also solvable modulo any of these interpretations. This fact is used to prove the following lemma.

Lemma 4.1. *If one of β_1 or β_2 is cyclic, then the problem is not solvable.*

Proof. Without loss of generality assume β_1 is not well-founded. If we interpret $*$ with $rp(*)$ (which gives a model for $\mathcal{E}_2$), it is not hard to see that all variables in the same $\sim_{rp(*)}$-equivalence class become equal to each other. Hence the relation $\succ_B$ becomes not well founded on the set of variables. This implies that there is a cycle w.r.t. $\succ_B$ in the interpreted problem (which is a standard unification problem) and hence there is no solution. Thus the result follows, since the interpreted problem is solvable if the original problem is solvable. A similar argument holds for β_2. □

Therefore, we introduce the following failure rule:

(F3) $\dfrac{\mathcal{EQ}}{FAIL}$ if any of the β_i, $i \in \{1, 2\}$, is cyclic

We will illustrate these with an example. Let $U =^? B(X, Y)$ and $U =^? U_1 * U_2$ be two equations. After rule (d) is applied to this pair of equations, we get

$$U =^? U_1 * U_2,\ X =^? U_1 * Z,\ U_2 =^? B(Z, Y)$$

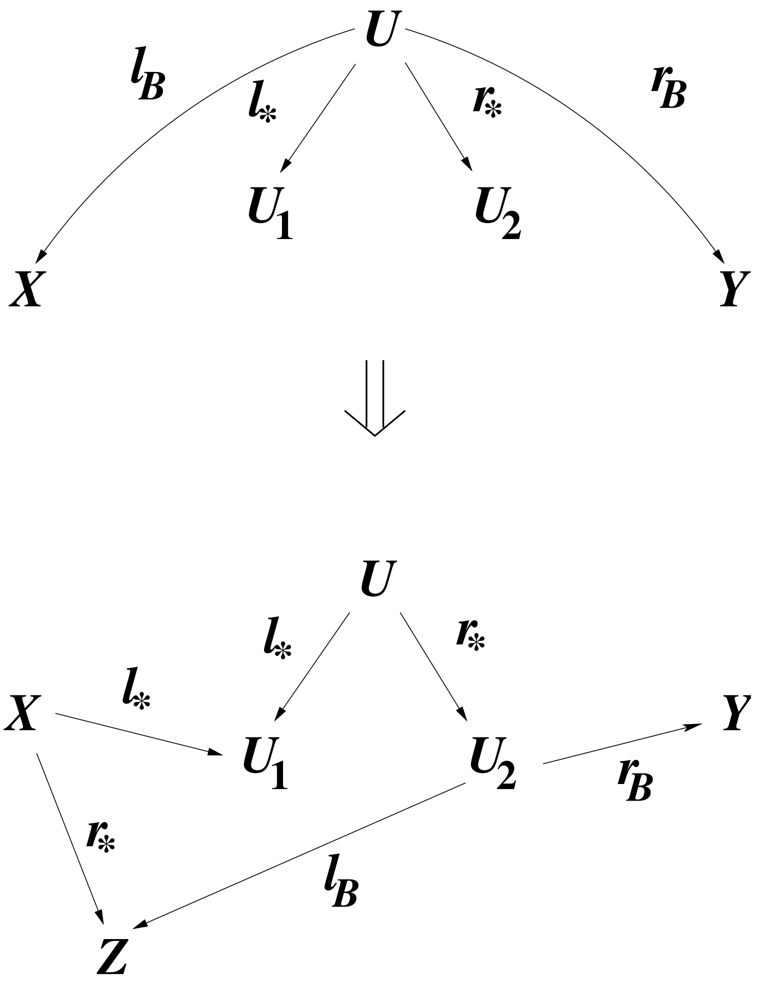


Figure 1: Splitting

where Z is a new variable. Now observe that $Z \sim_{lp(B)} U_2$ and $Z \sim_{rp(*)} X$. Thus every new variable introduced by an application of rule (d) is *below* an already existing variables by $\succ_*$ and $\succ_B$ (see Figure 1) and also *equivalent* to existing variables by one of $\{\sim_{rp(*)},\ \sim_{lp(B)}\}$.

Lemma 4.2. *For each equivalence relation in $\{\sim_{rp(*)}, \sim_{lp(B)}\}$, the number of equivalence classes does not increase with the splitting rule.*

Proof. Trivial, since if we apply rule (d), we see that new variable Z 'joins' the existing $\sim_{rp(*)}$- and $\sim_{lp(B)}$-equivalence classes (see Figure 1). □

The number of equivalence classes modulo any equivalence relation will be less than or equal to the number of initial variables in a given problem.

Lemma 4.3. *If rules (a)-(d) are applied infinitely, then one of the relations β_i $(i = 1, 2)$ is cyclic.*

Proof. The only case we need to look at carefully is when the splitting rule is applied. By Lemma 4.2 new variables will not create new equivalence classes; instead they join already existing equivalence classes. Note that for every new variable X created by the splitting rule there is another variable $Y \succ_* X$ which was created earlier. Thus if splitting goes on indefinitely, then we get arbitrarily long chains of the form

$$X_{i_1} \succ_* X_{i_2} \succ_* \dots$$

But since the number of $\sim_{lp(B)}$- equivalence classes for the problem do not increase, there are indices j and k such that $j < k$ and $X_{i_j} \sim_{lp(B)} X_{i_k}$. (In fact, if n is the number of variables in the original problem, then $j < k \leq n+1$). This will cause β_2 to be cyclic after finitely many steps. □

Theorem 4.4. *The unification problem modulo $\mathcal{E}_2$ is decidable.*

Proof. Let $\mathcal{S}$ be a problem modulo $\mathcal{E}_2$. If $\mathcal{S}$ is unifiable, rules (a)-(e) will return a solution. On the other hand, if $\mathcal{S}$ is not unifiable, then the possible errors are function clash, occur check error and infinite splitting among variables. Rules (F1) and (F2) detect function clash and occur check in finite time. In the case of infinite splitting, the algorithm will encounter the failure rule (F3) which checks if any of β_i relations is cyclic. Thus, our algorithm decides if $\mathcal{S}$ is unifiable (and computes a comlete set of unifiers) □

Theorem 4.5. *Unifiability modulo $\mathcal{E}_2$ is in P.*

Proof. By Lemma 4.2, we know that the number of equivalence classes remains same throughout the algorithm. Let n be the number of variables. It is easy to see that the number of equivalence classes in both $\sim_{lp(B)}$ and $\sim_{rp(*)}$ is at most n. Note that the algorithm terminates if rule (d) terminates. Rule (d) removes an existing l_B-edge between equivalence classes and adds a new one, which is one level *below the old one* with respect to r_* (see Figure 1). By Lemma 4.3, this cannot go on for more than n times without failure. Therefore the result follows. □

In the next section, we show that the cardinality of minimal complete set of unifiers is "one".

4.1 Unification Type of $\mathcal{E}_2$

We know that standard forms can be used to represent any problem modulo $\mathcal{E}_2$. Therefore, the transformation rules which we define form a complete method for the problem. We use this fact indirectly to show that the unification type of $\mathcal{E}_2$ is unitary. We first prove that the transformation steps (a)–(e) "preserve unifiers."

Lemma 4.6. *Let $\mathcal{S}$ be a unification problem in standard form and let $\mathcal{S}'$ be obtained after applying one of (a)–(e). Then*

1. *Every unifier of $\mathcal{S}'$ is a unifier of $\mathcal{S}$.*

2. *For every unifier σ of $\mathcal{S}$ there is a unifier $\hat{\sigma}$ of $\mathcal{S}'$ such that $\sigma \equiv_{Var(S)} \hat{\sigma}$.*

Theorem 4.7. *The unification type of $\mathcal{E}_2$ is unitary, and our algorithm computes a complete set of $\mathcal{E}_2$-unifiers.*

Proof. Let $\mathcal{T}$ be the solved form for $\mathcal{S}$. It is easy to see that $\mathcal{T}$ itself is a sequential unifier, and the corresponding parallel unifier, say σ, is a unifier of $\mathcal{S}$. On the other hand, let θ be a unifier of $\mathcal{S}$. By induction on the number of steps we can show, using Lemma 4.6, that there is a unifier $\hat{\theta}$ of $\mathcal{T}$ such that $\theta \equiv_{Var(S)} \hat{\theta}$. We can now show that $\hat{\theta}$ is an instance of σ. □

5 Unification in $\mathcal{E}_1 \cup \mathcal{E}_2$

We show the decidability of unification modulo the union of $\mathcal{E}_1$ and $\mathcal{E}_2$. This theory has the following convergent system:

$$\begin{aligned} U(S(B(m,x)),x) &\rightarrow S(m) \\ m * B(n,r) &\rightarrow B(m*n,r) \end{aligned}$$

Orienting the second axiom the other way causes the Knuth-Bendix completion procedure to diverge.

Let us consider the term rewriting system $U(S(B(m,x)),x) \rightarrow S(m)$ associated with $\mathcal{E}_1$ and denote it by $\mathcal{R}$. Consequently, we use the equational (or class) rewriting relation $\mathcal{R}/\mathcal{E}_2$ and the extended rewriting relation $\mathcal{E}_2\backslash\mathcal{R}$ which is between $\mathcal{R}$ and $\mathcal{R}/\mathcal{E}_2$, i.e.,

$$\mathcal{R} \subseteq \mathcal{E}_2\backslash\mathcal{R} \subseteq \mathcal{R}/\mathcal{E}_2$$

From Section 3, we know that $\mathcal{R}$ is convergent and optimally reducing. We extend these results to $\mathcal{E}_2\backslash\mathcal{R}$. In other words, we show that $\mathcal{E}_2\backslash\mathcal{R}$ is convergent and optimally reducing modulo $\mathcal{E}_2$.

Termination of $\mathcal{R}/\mathcal{E}_2$ follows easily. We observe that for each term t, its equivalence class $[t]_{=_{\mathcal{E}_2}}$ is finite. Applying the rewrite rule in $\mathcal{R}$ modulo $\mathcal{E}_2$ causes a U symbol to disappear. Hence there is no infinite descending chain $t \rightarrow_{\mathcal{R}/\mathcal{E}_2} t' \rightarrow_{\mathcal{R}/\mathcal{E}_2} \ldots$. Furthermore, termination of $\mathcal{E}_2\backslash\mathcal{R}$ follows.

Recall that $\mathcal{R}$ is $\mathcal{E}_2$-confluent if and only if for every s,t such that $s =_{\mathcal{R}\cup\mathcal{E}_2} t$, there exist s',t' such that $s \rightarrow^*_{\mathcal{E}_2\backslash\mathcal{R}} s'$ and $t \rightarrow^*_{\mathcal{E}_2\backslash\mathcal{R}} t'$, and $s' =_{\mathcal{E}_2} t'$ [11].

Lemma 5.1. *$\mathcal{E}_2\backslash\mathcal{R}$ is convergent modulo $\mathcal{E}_2$.*

Proof. This follows from the critical pair criteria given by Jouannaud and Kirchner [9]. That is, if $\mathcal{R}/\mathcal{E}_2$ is terminating and all $\mathcal{E}_2$-classes are finite, then $\mathcal{E}_2\backslash\mathcal{R}$ is confluent if and only if all critical pairs in $CP_{\mathcal{E}_2}(\mathcal{R},\mathcal{R})$ and $CP_{\mathcal{E}_2}(\mathcal{R},\mathcal{E})$ are joinable. Note that all equivalence classes of $\mathcal{E}_2$ are finite and $\mathcal{R}/\mathcal{E}_2$ is terminating. Furthermore, the subterm ordering modulo $\mathcal{E}_2$, denoted as $\succsim_{\mathcal{E}_2}$, is also well-founded [3] since the equation $m*B(n,r) = B(m*n,r)$ is regular and *size-preserving,* i.e., left and right hand sides of the equation are of the same size. Thus the necessary conditions to apply the result in [9] are satisfied. Consider the complete sets of $\mathcal{E}_2$-critical pairs $CP_{\mathcal{E}_2}(\mathcal{R},\mathcal{R})$. The only non-variable position p in term $l = U(S(B(m,x)),x)$ such that $l|_p$ can be unified with (a variant of) l modulo $\mathcal{E}_2$ is $p = \epsilon$ (i.e., at the root). But this leads to a *trivial* critical pair. It is not hard to see that the set of critical pairs $CP_{\mathcal{E}_2}(\mathcal{R},\mathcal{E}_2)$, obtained from overlapping $\mathcal{R}$ "below" $\mathcal{E}_2$, is empty. Since $\mathcal{E}_2$-unification is decidable and unitary, the result follows. □

We next extend the optimal reducibility of $\mathcal{R}$ to the optimal $\mathcal{E}_2$-reducibility of $\mathcal{R}$. A rewrite rule $l \rightarrow r$ is *optimally E-reducing* if and only if for any substitution θ for which $\theta(r)$ is $E\backslash R$-reducible, there is a *proper* subterm s of l such that $\theta(s)$ is $E\backslash R$-reducible. A rewrite system R is *optimally E-reducing* iff every rule in it is optimally E-reducing modulo R.

Lemma 5.2. *$\mathcal{R}$ is optimally $\mathcal{E}_2$-reducing.*

Proof. Straightforward, since for all substitutions θ, $\theta(S(m))$ is reducible if and only if $\theta(m)$ is. □

Furthermore, we use the following fact about $\mathcal{E}_2\backslash\mathcal{R}$. Since the root symbol of the left-hand side of $\mathcal{E}_1$, namely U, is not in $Sig(\mathcal{E}_2)$, if s is in normal form and θ is an irreducible substitution modulo $\mathcal{E}_2\backslash\mathcal{R}$, then $\theta(s)$ can be reduced to its $\mathcal{E}_2\backslash\mathcal{R}$-normal form in $|s|$ steps by the innermost reduction strategy.

As mentioned earlier, it was shown by Narendran et al [12] that every optimally reducing and confluent term rewriting system has a decidable unification problem. We give a similar proof to the one in [12] for showing that $\mathcal{E}_2\backslash\mathcal{R}$ is decidable.

Theorem 5.3. *Unification modulo $\mathcal{E}_1 \cup \mathcal{E}_2$ is decidable and finitary, and there is an algorithm to compute a complete set of $\mathcal{E}_1 \cup \mathcal{E}_2$-unifiers.*

Proof. Let s and t be two terms and θ an irreducible substitution that unifies them. $\theta(s)$ and $\theta(t)$ are reduced to $\mathcal{E}_2\backslash\mathcal{R}$-normal forms by the rule

$$U(S(B(m,x)),x) \;\rightarrow\; S(m)$$

in at most $|s|$ and $|t|$ steps respectively by the innermost reduction strategy. Consider the sequence of positions where the reductions occur. Without performing unification, we instead mimic each reduction step as $s|_p =^?_{\mathcal{E}_2} \eta(U(S(B(m,x)),x))$, where p is a position that a reduction occurs and η an appropriate renaming. Repeat the same for new term $s\left[\eta(S(m))\right]_p$ and so on. Thus, the idea is to transform the problem by mimicking an innermost reduction sequence where the reductions take place at each *original* term position. We obtain at most $|s| + |t| + 1$ equations to be unified. We apply $\mathcal{E}_2$-unification to the resulting equations and see if there is a solution. Since $\mathcal{E}_2$-unification is decidable and unitary, the result follows. □

6 Conclusion

We have given an equational theory based on the RSA implementation of blind signaures and studied three relevant unification problems. We first considered the two axioms $\mathcal{E}_1$ and $\mathcal{E}_2$ as separate theories and finally unification modulo $\mathcal{E}_1 \cup \mathcal{E}_2$, which turned out to be decidable and finitary. The equational theories we consider are only some of the many possible axiomatizations about blind signatures. Future work would include incorporating other equational axioms. Furthermore, we plan to implement the algorithms and integrate them with the Maude-NPA protocol analyzer [7].

References

[1] S. Anantharaman, H. Lin, C. Lynch, P. Narendran, M. Rusinowitch. "Cap Unification: Application to Protocol Security modulo Homomorphic Encryption". In: *Proc. of the* 5*th ACM Symp. on Information, Computer and Communications Security*, ASIACCS'10, pp. 192–203, April 2010.

[2] F. Baader, W. Snyder. "Unification Theory". In: *Handbook of Automated Reasoning*, pp. 440–526, Elsevier Sc. Publishers B.V., 2001.

[3] L. Bachmair. *Canonical Equational Proofs.* Birkhäuser 1991.

[4] D. Chaum. "Security without Identification: Transaction System to Make Big Brother Obsolete". *Communications of the ACM* 28(2): 1030–1044, 1985.

[5] D. Chaum. "Blind signatures for untraceable payments". In: *Advances in Cryptology - Crypto '82* 199–203, 1983.

[6] H. Comon-Lundh, S. Delaune. "The finite variant property: how to get rid of some algebraic properties". In: Proc. of *RTA'05* (J. Giesl, ed.), LNCS 3467, pages 294–307. Springer-Verlag, 2005.

[7] S. Escobar, C. Meadows, J. Meseguer. "Maude-NPA: Cryptographic Protocol Analysis Modulo Equational Properties". In: *Foundations of Security Analysis and Design V, FOSAD 2007/2008/2009 Tutorial Lectures* (A. Aldini, G. Barthe, and R. Gorrieri, eds.) LNCS 5705, pages 1–50.

[8] J.-P. Jouannaud, C. Kirchner. "Solving equations in abstract algebras: a rule-based survey of unification." In: *Computational Logic: Essays in Honor of Alan Robinson*, pp. 360–394, MIT Press, Boston (1991).

[9] J.-P. Jouannaud, H. Kirchner. "Completion of a Set of Rules Modulo a Set of Equations". *SIAM J. Comput.* 15(4): 1155–1194, 1986.

[10] C. Lynch, B. Morawska. "Basic Syntactic Mutation." In: *Proc. of CADE 2002* (A. Voronkov, ed.), LNCS 2392, pages 471–485.

[11] C. Meadows, P. Narendran. "A Unification Algorithm for the Group Diffie-Hellman Protocol". In: *Proc. of WITS 2002* 3(1–2): 14–15, 2002.

[12] P. Narendran, F. Pfenning, R. Statman. "On the Unification Problem for Cartesian Closed Categories". *J. Symbolic Logic* 62(2): 636–647, 1997.

[13] R. Nieuwenhuis. "Basic paramodulation and decidable theories." In: *Proc. 11th IEEE Symposium on Logic in Computer Science (LICS'96)* 473–482.

[14] E. Tiden, S. Arnborg. "Unification Problems with One-sided Distributivity". *Journal of Symbolic Computation* 3(1–2): 183–202, 1987.

Incremental Variable Splitting
(Presentation-only paper)

C. M. Hansen, R. Antonsen, M. Giese, and A. Waaler
Dept. of Informatics, University of Oslo, Norway

In free-variable tableau calculi, free variables are introduced as place-holders when expanding universal formulas in order to postpone the choice of an instantiation, see e.g. [4]. Free variables are instantiated when branches are closed, by unifying potentially complementary formulas on a branch. The expansion of disjunctive formulas splits a proof into several branches, and the same free variable can occur on more than one branch. Usually, occurrences of a free variable on different branches have to be instantiated consistently to ensure soundness.

Antonsen and Waaler [1, 2] have analyzed the dependency between branching and the instantiation of free variables and have arrived at a criterion that, in some cases, permits to instantiate a free variable differently on different branches. Such divergent instantiation is referred to as *variable splitting*.

Unfortunately, the variable splitting technique gives only a global criterion that states when a substitution that closes all branches, possibly instantiating free variables on different branches differently, is *admissible*. No hint is given as to how the existence of such an admissible closing substitution can be ensured during proof search, short of performing an admissibility check for all possible closing substitutions after each proof step. As it stands, the existing work on variable splitting is therefore not suited for direct implementation.

We found that a similar problem of applying a global closure check after each proof step lies at the heart of the *incremental closure* approach proposed by Giese [5, 6]. Incremental closure is mainly designed as a method to avoid the backtracking usually employed when searching for free variable tableau proofs. Instead of globally instantiating free variables when a branch can be closed, incremental closure determines after each proof step whether there is a way to close *all* branches of a proof *simultaneously*. To do this efficiently, the set of closing substitutions for every subderivation is kept track of during proof search, and this information is updated whenever a new complementary pair of literals (i.e. a *connection*) is introduced. Syntactic unification constraints are used to represent sets of substitutions.

The main contribution in this article is to show how the incremental closure approach can be extended to provide an incremental, and therefore tractable evaluation of the global closure criterion of the variable splitting calculus. This is done by recasting the admissibility condition for closing substitutions into a constraint satisfaction problem. The constraint language contains syntactic unification constraints like in [5], and additionally *ordering* and *consistency* constraints to express the admissibility criterion of variable splitting. The resulting mechanism allows to check the existence of an admissible closing substitution incrementally during the construction of a proof.

We present a rule-based algorithm for testing satisfiability of constraints that is an extension of the well-known rule system for syntactic unification of Comon and Kirchner [3].

We have implementeted our approach in a prototypical variable splitting theorem prover. This implementation allows to apply the same strategy for rule applications both with and without variable splitting, which gives us a means of direct comparison between the two approaches. We present experimental results from running the prover on a wide range of problems.

References

[1] Roger Antonsen. *The Method of Variable Splitting.* PhD thesis, Univ. of Oslo, 2008.

[2] Roger Antonsen and Arild Waaler. Liberalized variable splitting. *Journal of Automated Reasoning*, 38:3–30, 2007.

[3] Hubert Comon and Claude Kirchner. Constraint solving on terms. In Hubert Comon, Claude Marché, and Ralf Treinen, editors, *Constraints in computational logics*, volume 2002 of *LNCS*, pages 47–103. Springer, 2001.

[4] Melvin Fitting. *First-order logic and automated theorem proving (2nd ed.).* Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1996.

[5] Martin Giese. Incremental closure of free variable tableaux. In *Proc. IJCAR 2001*, number 2083 in LNCS, pages 545–560. Springer, 2001.

[6] Martin Giese. *Proof Search without Backtracking for Free Variable Tableaux.* PhD thesis, Fakultät für Informatik, Universität Karlsruhe, July 2002.

Evaluating Automated Theorem Provers for First-Order Modal Logics

Thomas Raths[*] Jens Otten

Institut für Informatik, University of Potsdam
August-Bebel-Str. 89, 14482 Potsdam-Babelsberg, Germany
{traths,jeotten}@cs.uni-potsdam.de

Abstract

First-order modal logics have many applications, e.g., in planning, natural language processing, program verification, querying knowledge bases, and modeling communication. This paper gives an overview of several new implementations of theorem provers for first-order modal logics based on different proof calculi. Among these calculi are the standard sequent calculus, a prefixed tableau calculus, an embedding into simple type theory, and an instance-based method. All these theorem provers are tested and evaluated on the QMLTP problem library for first-order modal logic. The results of these test runs are compared and analyzed.

1 Introduction

Modal logics extend classical logic with the modalities ”it is necessarily true that” and ”it is possibly true that” represented by the unary operators $\Box$ and $\Diamond$, respectively. *First-order* modal logics extend propositional modal logics by *domains* specifying sets of objects that are associated with each world, and the standard universal and existential quantifiers [9, 15, 17].

First-order modal logics allow a natural and compact knowledge representation. The subtle combination of the modal operators and first-order logic enables specifications of epistemic, dynamic and temporal aspects, and of infinite sets of objects. For this reason, first-order modal logics have many applications, e.g., in planning, natural language processing, program verification, querying knowledge bases, and modeling communication. In these applications modalities are used to represent incomplete knowledge, programs, or to contrast different sources of information. First-order components, such as variables, functions, predicates and quantifiers enable to describe objects, their properties, types, and abstract information that can be instantiated later.

For example, the planning system PKS [27] constructs conditional plans. It uses modal operators to represent incomplete knowledge, constants and predicates to describe objects and their properties, and variables and functions to generate abstract plans, which are instantiated later, when sufficient knowledge is available. An inference procedure for a restricted quantified modal logic determines whether the plan achieves

[*]This work is partly funded by the German Science Foundation DFG under reference number KR858/9-1.

the goal and the preconditions of the actions hold, and generates the effects of the actions. PKS can be applied to, e.g., dialogue planning [37]. The dialogue system Artimis [33] and the sentence-planner SPUD [40] plan, generate and interpret sentences in a natural language. They use modalities to distinguish beliefs, intentions and actions of the system and the user. First-order logic components represent objects, properties and quantified statements. Variables enable to process abstract instructions that can be instantiated later when more information is available [39]. An inference engine is adapted to plan and interpret the sentences. The systems KIV [31], VSE-II [1] and KeY [8] are advanced tools for program verification and synthesis. Their proof components use dynamic and temporal first-order logic which are closely related to first-order modal logic. The modalities represent programs, whereas functions, variables and quantifiers characterize attributes, types and the creation of objects. Likewise the verification of database update programs [36] and the integration of UML specification [10] can be described in first-order modal logic. A first-order modal logic is also used as query language for description logic knowledge bases [11]. Automated reasoning is required to answer queries and to verify and optimize integrity conditions. Finally, first-order modal logics are used to describe communication and cooperation [12, 23].

All these applications require the use of automated theorem proving (ATP) systems for *first-order* modal logics. Whereas there are some ATP systems available for propositional modal logics, e.g., MSPASS [20] and modleanTAP [4], there are currently only few ATP systems that can deal with the full first-order fragment of modal logics.

The purpose of this paper is to introduce some new ATP systems for first-order modal logics and to evaluate, compare and analyze their performance on a standardized problem set. The reader is assumed to be familiar with the syntax and semantics of first-order modal logics, see, e.g., [15, 17]. If not stated otherwise the standard semantics and the following options regarding first-order terms for all evaluated ATP systems are considered: term designation is rigid, i.e., the terms denote the same object in each world, and terms are assumed to be local, i.e., any ground term denotes an existing object for each world.

This paper is structured as follows. In Section 2 all new and existing ATP systems for first-order modal logics are shortly described. Section 3 provides details about the used problem set and presents comprehensive performance results and comparisons of all described ATP systems. Section 4 concludes with a short summary and a few remarks on future work.

2 Implementing First-Order Modal Theorem Provers

The following (new and existing) ATP systems for first-order modal logics are described in this section: MleanSeP based on the standard sequent calculus, GQML-Prover and MleanTAP based on tableau calculi, M-Leo-II and M-Satallax based on an embedding into simple type theory and the f2p-MSPASS based on an instance-based method. Table 1 gives an overview of these systems.

2.1 Sequent Calculus

MleanSeP implements the standard sequent calculus for several modal logics.[1] It is implemented in Prolog. Proof search is carried out in an analytic way and free-variables are used in combination with a dynamic Skolemization that is calculated during the

[1]MleanSeP can be downloaded at `www.leancop.de/mleansep/programs/mleansep11.pl` .

actual proof search. Together with the occurs-check of the term unification algorithm this ensures that the Eigenvariable condition is respected.

MleanSeP 1.1 can deal with the first-order cumulative and constant domains of the modal logics K, K4, D, D4, S4, and T. To deal with constant domains, the *Barcan formula*[2] is automatically added to the given formula in a preprocessing step.

2.2 Tableau Calculi

GQML-Prover [44] is based on a free-variable tableau calculus using annotated tableau nodes and function symbols. It uses a liberalized δ^+-rule and is implemented in OCaml. GQML-Prover 1.2 can deal with cumulative, constant and varying domains of the modal logics K, K4, D, S4, and T, using rigid or non-rigid terms, and local or non-local terms.

MleanTAP implements a prefixed tableau calculus.[3] The compact system is implemented in Prolog. It uses not only free term variables but also free string variables for the prefixes and a prefix unification procedure. It is based on the ileanTAP system for first-order intuitionistic logic [24]. At first MleanTAP performs a purely classical proof search. After a classical proof is found, the prefixes of those literals that close branches in the (classical) tableau are unified. The existence of a prefix substitution ensures that the given formula is valid in modal logic as well [45]. If no prefix substitution exists backtracking is done in order to find alternative classical proofs (and prefixes). To deal with different modal logics only the prefix unification procedure has to be adapted [22].

MleanTAP 1.1 can deal with the first-order cumulative and constant domains of the modal logics D and S4. By further modifying the prefix unification algorithm MleanTAP can be extended to the modal logics D4, S5, and T.

2.3 Embedding into Simple Type Theory

M-Leo-II 1.2 and M-Satallax 1.4 extend the ATP systems Leo-II 1.2 and Satallax 1.4 to first-order modal logics, respectively. Both provers use an embedding of quantified modal logic into simple type theory [6]. Leo-II [7] and Satallax [2] are ATP systems for typed higher-order logic.[4] Leo-II is based on an extensional higher-order resolution calculus. It cooperates with a first-order ATP system, by default E [34], and applies term sharing and term indexing techniques. It is implemented in OCaml. Satallax uses a complete ground tableau calculus for higher-order logic to generate successively propositional clauses and calls MiniSat repeatedly to test unsatisfiability of these clauses. It can be regarded as an instance-based method for higher-order logic. Satallax is implemented in Steel Bank Common (SBC) Lisp.

Currently the embedding of quantified modal logic into simple type theory works for constant domains only. Thus, M-Leo-II 1.2 and M-Satallax 1.4 can deal with the first-order constant domains of the modal logics K, D, S4, S5, and T.

[2]The Barcan formula scheme has the form $\forall\vec{x}(\Box p(\vec{x}) \Rightarrow \Box\forall\vec{x}p(\vec{x})$ with $\vec{x} = x_1, \ldots, x_n$ for all predicates p with $n \geq 1$.

[3]MleanTAP can be downloaded at `www.leancop.de/mleantap/programs/mleantap11.pl` .

[4]These two higher-order ATP systems were selected as they have the best performance of all currently available systems for higher-order logic [43].

2.4 Instance-Based Method

f2p-MSPASS 3.0 uses an instance-based method to generate ground formulas and the ATP system MSPASS 3.0 for proving formulas in propositional modal logic. Like most instance-based methods, the f2p-MSPASS system consists of two components. The first component, called first2p, takes a first-order modal formula, removes all quantifiers and replaces every variable with a unique constant. The second component, MSPASS [20], takes the resulting (ground) formula and tries to find a proof or a counter model. If a counter model is found first2p adds quantified subformulas to the original formula and instantiates variables with new terms. Afterwards MSPASS is again used to find a proof for the resulting formula. If first2p is unable to add any new instances of subformulas, the original formula is invalid.

first2p is written in Prolog. It does not translate the given formula into any clausal form but preserves its structure throughout the whole proof process. Due to the restrictions of modal logics this instance-based approach does only work for formulas that contain either only existential or only universal quantifiers. MSPASS is an extension of and incorporated into the resolution-based ATP system SPASS. It uses several translation methods into classical logic. By default the standard relational translation from modal logic into classical logic is applied.

f2p-MSPASS 3.0 and can deal with the first-order cumulative and constant domains of the modal logics K, K4, K5, B, D, S4, S5, and T. To deal with constant domains, first2p automatically adds the Barcan formula (see Section 2.1) to the original formula in a preprocessing step.

Table 1: First-order modal ATP systems

ATP system	modal logics	domains	equality	language
MleanSeP 1.1	K,K4,D,S4,T	cumulative, constant	no	Prolog
MleanTAP 1.1	D, S4	cumulative, constant	no	Prolog
GQML-Prover 1.2	K,K4,D,S4,T	cumul., const., vary.	no	OCaml
M-Leo-II 1.2	K,D,S4,S5,T	constant	yes	SBC Lisp
M-Satallax 1.4	K,D,S4,S5,T	constant	yes	OCaml
f2p-MSPASS 3.0	K,K4,K5,B, D,S4,S5,T	cumulative, constant	no	C/Prolog

3 Evaluating First-Order Modal Theorem Provers

In this section the ATP systems presented in Section 2 are evaluated on the QMLTP library for first-order modal logics. Some details about the QMLTP library are given first before comprehensive performance results and comparisons are presented.

3.1 Problem Set: The QMLTP Library

Testing ATP systems using standardized problem sets is a well-established method for measuring their performance. For example, the TPTP library [41] and the ILTP library [30] were developed for classical and intuitionistic logic, respectively. These problem libraries have fostered the development of more efficient systems for these logics. The basic idea of using problem libraries is to run the ATP systems on the problems included in the library, determine the number of problems solved within a given time limit, and to collect further data, e.g., the average run time used to solve the problems. In order to have these performance results reflect the capabilities of the systems accurately, the problem library shall be large enough, span a variety of difficulty and subject matters, and have a standardized syntax [41].

Until recently there existed only very small collections of formulas that could be used for testing and evaluating ATP systems for *first-order* modal logics. A small set of first-order formulas was used for testing GQML-Prover [44]. For *propositional* modal logics there exist some scalable problem classes [3] and approaches that generate formulas randomly in a normal form [26]. But generating formulas randomly is not an appropriate approach for the first-order case.

The Quantified Modal Logic Theorem Proving (QMLTP) library [29] provides a comprehensive set of standardized problems in first-order modal logics and, thus, constitutes a convenient basis for testing and evaluating the performance of ATP systems for first-order modal logics. The main purpose of the QMLTP library is to stimulate the development of new ATP systems and calculi for first-order modal logics. The current release v1.0 of the QMLTP library contains 500 problems represented in an extended TPTP syntax. The problem set is available for download at `http://www.iltp.de/qmltp`.

Release v1.0 of the QMLTP library includes 245 problems that are generated by using Gödel's embedding of intuitionistic logic into the modal logic S4 [19]. The original problems were taken from the TPTP library [41]. 10 problems were taken from applications, e.g., planning, querying databases, natural language processing and communication, and software verification [10, 11, 13, 32, 38, 39]. 175 problems come from various textbooks [14, 15, 16, 18, 28, 35, 44] and 70 problems from the TANCS-2000 system competition for modal ATP systems [21].

There are only few problems from real applications in the current release of the QMLTP library. Future versions will include more problems from applications mentioned in Section 1 once they are submitted to the QMLTP library. The aim of the QMLTP library is to start a cycle in which developers are inspired to improve their ATP systems and users are encouraged to apply these systems and to contribute their problems to the library stimulating the development of more efficient systems.

Each problem in the QMLTP library is assigned a modal status and a modal rating. The *status* is either `Theorem`, `Non-Theorem` or `Unsolved`. Problems with `Unsolved` status have not been solved by any ATP system.[5] The *rating* determines the difficulty of a problem with respect to current state-of-the-art ATP systems. It is the fraction of state-of-the-art ATP systems that are *not* able to solve a problem within a given time limit. For example a rating of 0.3 indicates that 30% of the state-of-the-art systems do *not* solve the problem; a problem with rating of 1.0 cannot be solved by any state-of-the-art system. A *state-of-the-art* system is an ATP system whose set of solved problems is not subsumed by that of any another ATP system. In the current release of the QMLTP library status and rating information is given with respect to the constant and cumulative domains of the modal logics D and S4.

In order to represent modal problems in a standardized *syntax*, the Prolog syntax of the TPTP library [41] is extended by the modal operators $\Box$ and $\Diamond$. The two Prolog atoms "`#box`" and "`#dia`" are used for representing $\Box$ and $\Diamond$, respectively. The formulas $\Box F$ and $\Diamond F$ are then represented by "`#box:F`" and "`#dia:F`", respectively

[5]No theoretical investigations regarding the status of formulas have been done.

```
%------------------------------------------------------------------------
% File     : SYM001+1 : QMLTP v1.0
% Domain   : Syntactic (modal)
% Problem  : Barcan scheme instance. (Ted Sider's qml wwf 1)
% Version  : Especial.
% English  : if for all x necessarily f(x), then it is necessary that for
%            all x f(x)
% Refs     : [Sid09] T. Sider. Logic for Philosophy. Oxford, 2009.
%          : [Brc46] [1] R. C. Barcan. A functional calculus of first
%            order based on strict implication. Journal of Symbolic Logic
%            11:1-16, 1946.
% Source   : [Sid09]
% Names    : instance of the Barcan formula
%
% Status   :      cumulative  constant
%            D    Unsolved    Theorem       v1.0
%            S4   Unsolved    Theorem       v1.0
%
% Rating   :      cumulative  constant
%            D    1.00         0.00        v1.0
%            S4   1.00         0.00        v1.0
%
% term conditions for all terms:  designation: rigid,  extension: local
%
% Comments :
%------------------------------------------------------------------------
qmf(con,conjecture,
(( ! [X] : (#box : ( f(X) ) ) ) => (#box : ( ! [X] : ( f(X) ) )))).
%------------------------------------------------------------------------
```

Figure 1: Example of problem file `SYM001+1` of the QMLTP library.

(see also Figure 1). For future extensions to multi-modal logic these atoms can be extended to, e.g., Prolog terms of the form "`#box(i)`" or "`#dia(i)`" in which the index "`i`" is an arbitrary Prolog atom. As there exists no ATP system for first-order multi-modal logic, the current release of the QMLTP library is restricted to uni-modal problems only.

A header with useful information is added to the *presentation* of each problem. It is adapted from the TPTP library and includes information about the file name, the problem description, the modal status and the modal difficulty rating. An example file of a first-order modal problem is given in Figure 1.

Note that the problem files of the QMLTP library are primary intended to present the *syntax* of modal formulas. The options of the intended *semantics*, e.g., the interpretation of the modal operators in the different modal logics is left to the (user of the) particular ATP system.[6] This increases the flexibility of the library as it can be used for all (uni-modal) logics that share the standard modal syntax.

3.2 Performance Evaluation

Several criteria are used to evaluate and compare the performance of ATP systems. The main measure is the number of problems that an ATP system solves within a given time limit. For the evaluation the output of a proof or a counter model is not required, i.e., an assurance of the existence of a proof or counter model is sufficient. The time limit is in terms of CPU time since no extensive memory usage was observed and, thus, wall clock time is not significantly higher than CPU time. The average CPU runtime of an ATP system was determined only for problems solved by *all* systems in order to make the comparison fair. Otherwise, a system that spends more time solving a difficult problem that is not solved by other systems would be disadvantaged.

[6]Even though some problems, e.g., problems stemming from Gödel's embedding were originally developed with a specific modal logic in mind.

Another interesting property when evaluating ATP systems is their time complexity behavior, i.e., the increase of number of solved problems when increasing the time limit. When an ATP system solves most problems of the ones it can solve at all within a second, its time complexity behavior is worse than that of a systems that still solves a significant number of problems after 10 or 100 seconds. A very steep time complexity behavior indicates that the ATP system's underlying proof calculus needs to be improved. What kind of problems are solved is interesting as well. Problems with equality are of interest, because some calculi and techniques might be more appropriate to deal with equality than others.

There are three further criteria to rate the performance of ATP systems: the state-of-the-art (SOTA) system rating, the state-of-the-art contribution (SOTAC) and the efficiency measure [41, 42]. The *SOTA system rating* states how many difficult problems an ATP system can solve. It is the fraction of problems that can be solved by the regarded system but not by all systems. The value is 1.0 if the system can solve all difficult problems, and is 0.0 if it can *only* solve problems that are solved by all (state-of-the-art) systems as well. The *SOTAC* of an ATP system measures its unique problems solving capability. The SOTAC of a specific *problem* is the inverse of the number of state-of-the-art systems that solve the problem. For example, the maximal value 1.0 indicates that only one system solves the problem, 0.5 means that two state-of-the-art system solve the problem. The SOTAC of an ATP *system* is the average SOTAC over all problems solved by the system. The less "unique" an ATP system the smaller is its SOTAC. Finally, the *efficiency measure* takes the number of solved problems and time taken to solve them into account. It is the fraction of solved problems divided by the average time needed to solve these problems. The more problems are solved and the faster they are solved by an ATP system the higher is its efficiency measure.

3.3 The Test Environment

The ATP systems described in Section 2 are evaluated on the modal logics D and S4 with constant and cumulative domains. These are the modal logics supported by the majority of available ATP systems. The standard semantics for the modal logics D and S4, rigid term designation and local terms are used [15].

Table 2: Test Environment

hardware	3.4 GHz Xeon, 4 GB RAM	
operating system	Linux 2.6.24-24.x86_64	
time limit	600 sec.	
modal logic	D	S4
accessibility relation	serial	reflexive, transitive
axioms	$\Box A \rightarrow \Diamond A$	$\Box A \rightarrow A$, $\Box A \rightarrow \Box\Box A$
domains	cumulative, constant	
terms	designation: rigid, extension: local	

A test environment was developed for automatically conducting all performance tests and for collecting and evaluating the results of all test runs. These test runs were conducted on an eight-processor cluster system in order to simultaneously test several ATP systems at a time. All ATP systems and components written in Prolog use

ECLiPSe Prolog 5.10. For M-Satallax 1.4 and M-Leo-II 1.2 the binaries of the CASC-J5 [43] were used. For MSPASS the sources of SPASS 3.0 were compiled using the GNU gcc compiler version 4.2.4.

The CPU time limit for all proof attempts was set to 600 seconds. For handling equality the equality axioms were added in a preprocessing step for the ATP systems MleanSeP, MleanTAP, GQML-Prover and f2p-MSPASS. The time required for adding the equality axioms is less than a second and not included in the overall timings. Table 2 summarizes the test conditions.

3.4 Performance Statistics

Table 3 and Table 4 give an overview of the test results for all ATP systems described in Section 2. M-Satallax and M-Leo-II can be applied to the constant domains only.

Table 3: Number of proved problems of the QMLTP library v1.0

	D		S4	
	cumulative	constant	cumulative	constant
MleanSeP 1.1	117	120	203	201
MleanTAP 1.1	84	120	189	205
M-Satallax 1.4	-	107	-	188
M-Leo-II 1.2	-	104	-	172
GQML-Prover 1.2	88	95	137	133
f2p-MSPASS 3.0	47	47	88	88

Table 4: Number of found counter models of the QMLTP library v1.0

	D		S4	
	cumulative	constant	cumulative	constant
MleanSeP 1.1	1	1	1	1
MleanTAP 1.1	1	1	1	1
M-Satallax 1.4	-	7	-	71
M-Leo-II 1.2	-	0	-	0
GQML-Prover 1.2	0	0	0	0
f2p-MSPASS 3.0	108	107	42	36

Table 5, 7, 9, and 11 present the performance results for the modal logics D and S4 with cumulative and constant domains, respectively. They contain the number of solved problems, the number of proved problems, and the number of counter models (disproved) found within the time limit, the fraction of solved problems, the number of solved problems within a specific time interval, the number of time outs, the number of solved problems containing equality, the number of problems solved by only one ATP system, the SOTA system rating, the SOTAC, and the efficiency measure as described in section 3.2. The average run time was determined only for problems solved by *all* ATP systems. These are 9% and 16% of all problems for the modal logics D and S4, respectively.

For some problems MleanSeP and MleanTAP produce a stack overflow (stack). f2p-MSPASS cannot be applied to 299 problems (gave up) as these problems contain both existential and universal quantifiers (see remarks in Section 2.4). For some problems GQML-Prover returns wrong results (inconsistent).[7]

Table 6, 8, 10, and 12 show the number of problems solved by one ATP system but not by another system. For example, for the modal logic D with cumulative domains MleanSeP solves 34 problems that are not solved by MleanTAP (see Table 6). The performance graph of all considered ATP systems for the modal logics D and S4 with cumulative and constant domains are depicted in Figure 2, 3, 4, and 5, respectively.

To compare the performance of the modal ATP systems with an ATP system for classical logic, the classical prover leanTAP [5] was run on all modal problems, in which the modal operators have been removed. Out of the 500 problems leanTAP 2.3 proves 282 problems and finds a counter model for one problem. It solves all except one problem within one second.

Table 5: Benchmark results for modal logic D with cumulative domains

	MleanSeP 1.1	MleanTAP 1.1	GQML-Prover 1.2	f2p-MSPASS 3.0
solved	118	85	88	155
[%]	24%	17%	18%	31%
proved	117	84	88	47
disproved	1	1	0	108
0s to 1s	117	85	71	155
1s to 10s	0	0	1	0
10s to 100s	0	0	16	0
100s to 600s	1	0	0	0
time out	349	411	143	46
stack / gave up	33	4	264	0
not applicable	0	0	0	299
inconsistent	0	0	5	0
solved with equality	37	18	16	0
only by this system	27	0	22	102
average time [s]	<0.01	<0.01	0.01	0.01
SOTA system rating	0.26	0.16	0.17	0.38
SOTAC	0.49	0.32	0.51	0.76
efficiency measure	0.05	289.00	0.02	102.23

Table 6: Number of problems solved by A but *not* by B for D with cumulative domains

system A	system B			
	MleanSeP 1.1	MleanTAP 1.1	GQML-Prover 1.2	f2p-MSPASS 3.0
MleanSeP 1.1	0	34	53	71
MleanTAP 1.1	1	0	26	37
GQML-Prover 1.2	28	34	0	55
f2p-MSPASS 3.0	108	107	117	0

[7] An inconsistency occurs, e.g., for problem SYM176+1 for all considered modal logics.

Table 7: Benchmark results for modal logic D with constant domains

	MleanSeP 1.1	MleanTAP 1.1	M-Satallax 1.4	M-Leo-II 1.2	GQML-Prover 1.2	f2p-MSPASS 3.0
solved	121	121	114	104	95	154
[%]	24%	24%	23%	21%	19%	31%
proved	120	120	107	104	95	47
disproved	1	1	7	0	0	107
0s to 1s	93	118	97	98	79	154
1s to 10s	27	3	10	0	1	0
10s to 100s	0	0	2	1	15	0
100s to 600s	1	0	5	5	0	0
time out	346	377	386	396	142	47
stack / gave up	33	2	0	0	257	0
not applicable	0	0	0	0	0	299
inconsistent	0	0	0	0	6	0
solved with equality	36	28	12	10	16	0
only by this ATP	6	4	3	1	15	99
average time [s]	<0.01	<0.01	0.34	0.05	<0.01	0.01
SOTA system rating	0.15	0.15	0.14	0.13	0.11	0.21
SOTAC	0.29	0.28	0.26	0.23	0.38	0.72
efficiency measure	0.04	2.08	0.04	0.03	0.02	38.25

Table 8: Number of problems solved by A but *not* by B for D with constant domains

system A	system B					
	MleanSeP 1.1	MleanTAP 1.1	M-Satallax 1.4	M-Leo-II 1.2	GQML-Prover 1.2	f2p-MSPASS 3.0
MleanSeP 1.1	0	14	31	35	53	74
MleanTAP 1.1	14	0	29	31	48	73
M-Satallax 1.4	24	22	0	12	45	64
M-Leo-II 1.2	18	14	2	0	37	58
GQML-Prover 1.2	33	28	32	34	0	63
f2p-MSPASS 3.0	107	106	104	108	116	0

3.5 Comparison of Performance Results

In general, ATP systems prove more problems of the QMLTP library with respect to the modal logic S4 than with respect to the modal logic D. Furthermore, more problems are proved for the constant domains condition than for the cumulative domains condition.

These results are in line with the fact that every formula that is valid in the modal logic D is also valid in S4, and that every formula that is valid for the cumulative domains condition is also valid for the constant domains condition as shown in the following figure:

Table 9: Benchmark results for modal logic S4 with cumulative domains

	MleanSeP 1.1	MleanTAP 1.1	GQML-Prover 1.2	f2p-MSPASS 3.0
solved	204	190	137	130
[%]	41%	38%	27%	26%
proved	203	189	137	88
disproved	1	1	0	42
0s to 1s	184	186	126	129
1s to 10s	8	2	0	0
10s to 100s	8	1	10	1
100s to 600s	4	1	1	0
time out	262	306	270	71
stack / gave up	34	4	91	0
not applicable	0	0	0	299
inconsistent	0	0	2	0
solved with equality	51	29	6	1
only by this system	28	16	28	46
average time [s]	0.72	<0.01	0.18	0.01
SOTA system rating	0.31	0.28	0.16	0.15
SOTAC	0.44	0.41	0.44	0.53
efficiency measure	0.06	0.23	0.06	1.28

Table 10: Number of problems solved by A but *not* by B for S4 with cumulative dom.

system A	system B			
	MleanSeP 1.1	MleanTAP 1.1	GQML-Prover 1.2	f2p-MSPASS 3.0
MleanSeP 1.1	0	32	95	123
MleanTAP 1.1	18	0	85	108
GQML-Prover 1.2	30	34	0	74
f2p-MSPASS 3.0	49	48	65	0

$$\begin{array}{ccc} \{F \mid F \text{ is valid in D cumulative domains}\} & \subset & \{F \mid F \text{ is valid in D constant domains}\} \\ \subset & & \subset \\ \{F \mid F \text{ is valid in S4 cumulative domains}\} & \subset & \{F \mid F \text{ is valid in S4 constant domains}\}. \end{array}$$

However, for the modal logic S4 MleanSeP proves more problem for the cumulative domains than for the constant domains. The reason for this behavior is that the inclusion of the Barcan formula increases the search space for formulas that are valid under both domain conditions.

MleanSeP proves the highest number of problems, except for S4 constant domain where MleanTAP proves more problems than any other prover. MleanSeP also proves the highest number of problems containing equality for all considered modal logics.

In general MleanTAP proves only slightly fewer problems than MleanSeP. It has the best performance for S4 with constant domains and proves many problems with equality as well. The time complexity behavior of MleanTAP is worse than that of MleanSeP. Both MleanSeP and MleanTAP each found only one counter model.

Table 11: Benchmark results for modal logic S4 with constant domains

	MleanSeP 1.1	MleanTAP 1.1	M-Satallax 1.4	M-Leo-II 1.2	GQML-Prover 1.2	f2p-MSPASS 3.0
solved	202	206	259	172	133	124
[%]	40%	41%	52%	34%	27%	25%
proved	201	205	188	172	133	88
disproved	1	1	71	0	0	36
0s to 1s	170	203	166	155	123	123
1s to 10s	20	2	51	7	0	0
10s to 100s	8	1	28	4	9	1
100s to 600s	4	0	14	6	1	0
time out	265	292	241	328	271	77
stack / gave up	33	2	0	0	83	0
not applicable	0	0	0	0	0	299
inconsistent	0	0	0	0	13	0
solved with equality	45	29	14	9	4	1
only by this system	14	10	24	1	8	0
average time [s]	0.05	<0.01	0.41	0.08	0.18	0.01
SOTA system rating	0.16	0.16	0.22	0.12	0.08	0.07
SOTAC	0.30	0.29	0.35	0.22	0.29	0.29
efficiency measure	0.05	2.61	0.04	0.05	0.06	1.08

M-Satallax and M-Leo-II prove only slightly fewer problems than MleanSeP and MleanTAP for D and S4 with constant domains. For the modal logic S4 M-Satallax is very strong in finding counter models and solves a large number of problems that were taken from the TANCS-2000. M-Leo-II does not find counter models and its time complexity is slightly worse than that of M-Satallax.

f2p-MSPASS finds a high number of counter models for all considered modal logics. Like M-Satallax it solves a high number of TANCS-2000 problems. However, its time complexity behavior is very steep. All except one problem are solved within one second. f2p-MSPASS cannot be applied to 299 problems (see remarks in Section 2.4). There are only 31 non-propositional modal problems that are suitable for the instance-

Table 12: Number of problems solved by A but *not* by B for S4 with constant domains

system A	system B					
	MleanSeP 1.1	MleanTAP 1.1	M-Satallax 1.4	M-Leo-II 1.2	GQML-Prover 1.2	f2p-MSPASS 3.0
MleanSeP 1.1	0	23	43	54	90	122
MleanTAP 1.1	27	0	41	48	88	124
M-Satallax 1.4	100	94	0	91	122	136
M-Leo-II 1.2	24	14	4	0	50	93
GQML-Prover 1.2	34	28	9	24	0	81
f2p-MSPASS 3.0	44	42	1	45	59	0

based approach of f2p-MSPASS.[8] This problem set is too small to provide meaningful comparisons on the performance of f2p-MSPASS on first-order modal problems.

The classical leanTAP prover solves 283 problems (ignoring the modal operators) indicating that many problems are hard even for classical tableau-based ATP systems.

4 Conclusion

Despite the fact that modal logics are considered as one of the most important non-classical logics, the availability of *implementations* of automated theorem provers for *first-order* modal logics is very limited so far. In this paper several new ATP systems for various first-order modal logics based on different proof calculi and methods were introduced.

The performance of all new and existing ATP systems for first-order modal logic was evaluated on all 500 problems included in the first release v1.0 of the QMLTP library. Comprehensive statistics including different performance measures as well as illustrative performance graphs of the time complexity behavior were given for each considered ATP system.

Even though most of the 500 problems included in the first release of the QMLTP library have a rather syntactic nature, the QMLTP library serves as a useful basis in order to obtain a first impression of the performance of ATP systems for first-order modal logics.[9] Future releases of the QMLTP library will provide more problems from actual applications, thus, putting the testing and evaluation of ATP systems for modal logic on a more stable basis. All interested users are invited to submit new first-order problems to the QMLTP library.

An analysis of the performance results shows that the ATP system based on standard modal sequent calculi (performing an analytic tableau-like search) proves the highest number of problems. It slightly outperforms the system based on prefixed tableau calculi.[10] As for classical logic the ATP system based on an instance-based method finds by far the highest number of counter models. The systems using the embedding of modal logic into type theory show a solid performance as well. All considered ATP system solve most problems of the ones they solve at all within one second. This behavior is similar to that of classical ATP systems using standard tableau calculi. It indicates that the underlying proof calculi need to be improved.

The implementation of ATP systems for first-order modal logic is still in its infancy. Future work includes the implementation of, e.g., connection-based calculi for first-order modal logic and the extension of existing ATP systems to some first-order *multi*-modal logics.

Acknowledgements.

The authors would like to thank Christoph Benzmüller for providing the LEO-II system and the embedding into simple type theory, Ulrich Hustadt and Renate Schmidt for providing the MSPASS system, and Geoff Sutcliffe for help with the installation of some ATP systems and many useful suggestions.

[8]Of these problems ft2-MSPASS solves 16 problems (5 proved/11 counter models) for cumulative D, 15 problems (5/10) for constant D, 15 problems (9/6) for cumulative S4, and 9 problems (8/1) for constant S4.

[9]Observe the fact that problem libraries for *classical* first-order logic started with 75 (printed) problems in 1986. In 1997 the TPTP library v2.0.0 included the first 217 (non-clausal) *first-order* problems [41]. Its number has risen to over 5000 problems in the most recent version of the TPTP library.

[10]A similar behavior has already been shown by ATP systems for intuitionistic first-order logic [25].

References

[1] S. Autexier et al. VSE: formal methods meet industrial needs. *STTT*, 3(1):66–77, 2000.

[2] J. Backes, C. E. Brown. Analytic Tableaux for Higher-Order Logic with Choice. *IJCAR 2010*, LNCS 6173, pp. 76–90. Springer, 2010.

[3] P. Balsiger, A. Heuerding, S. Schwendimann. A Benchmark Method for the Propositional Modal Logics K, KT, S4. *Journal of Automated Reasoning*, 24:297–317, 2000.

[4] G. Beckert, R. Goré. Free Variable Tableaux for Propositional Modal Logics. In D. Galmiche, Ed., *TABLEAUX-1997*, LNAI 1227, pp. 91–106, Springer, 1997.

[5] B. Beckert, J. Posegga. leanTAP: Lean Tableau-based Deduction. *Journal of Automatic Reasoning*, 15(3): 339–358, 1995.

[6] C. Benzmüller, L. Paulson. Quantified Multimodal Logics in Simple Type Theory. Seki Report SR-2009-02 (ISSN 1437-4447), Saarland University, 2009.

[7] C. Benzmüller, L. Paulson, F. Theiss, A. Fietzke. LEO-II – A Cooperative Automatic Theorem Prover for Higher-Order Logic. *IJCAR 2008*, LNAI 5195, pp. 162–170. Springer, 2008.

[8] B. Beckert, M. Giese, R. Hähnle, V. Klebanov, P. Rümmer, S. Schlager, P. H. Schmitt. The KeY System (Deduction Component). In F. Pfennig, Ed., *CADE-21*, LNCS 4603, pp. 379–384. Springer, 2007.

[9] P. Blackburn, J. van Bentham, F. Wolter. *Handbook of Modal Logic*. Elsevier, 2006.

[10] V. Boeva, L. Ekenberg. A Transition Logic for Schemata Conflicts. *Data & Knowledge Engineering*, 51(3):277–294, 2004.

[11] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, R. Rosati. EQL-Lite: Effective First-Order Query Processing in Description Logics. In M. M. Veloso, Ed., *IJCAI-2007*, pp. 274–279, 2007.

[12] P. R. Cohen, H. Levesque. Intention is choice with commitment. *Artificial Intelligence*, 42(3):213–261, 1990.

[13] L. Fariñas del Cerro, A. Herzig, D. Longin, O. Rifi. Belief Reconstruction in Cooperative Dialogues. *AIMSA 1998*, LNCS 1480, pp. 254–266. Springer, 1998.

[14] M. Fitting. *Types, Tableaus, and Goedel's God*. Kluwer, 2002.

[15] M. Fitting, R. L. Mendelsohn. *First-Order Modal Logic*. Kluwer, 1998.

[16] G. Forbes. *Modern Logic. A Text in Elementary Symbolic Logic*. Oxford University Press, 1994.

[17] J. Garson. Quantification in Modal Logic. *Handbook of Philosophical Logic*, volume II, pp. 249–307. D. Reidel Publ. Co, 1984.

[18] R. Girle. *Modal Logics and Philosophy*. Acumen Publ., 2000.

[19] K. Gödel. An Interpretation of the Intuitionistic Sentential Logic. In J. Hintikka, Ed., *The Philosophy of Mathematics*, pp. 128–129. Oxford University Press, 1969.

[20] U. Hustadt, R. A. Schmidt. MSPASS: Modal Reasoning by Translation and First-Order Resolution. R. Dyckhoff., Ed., *TABLEAUX-2000*, LNAI 1847, pp. 67–81. Springer, 2000.

[21] F. Massacci, F. M. Donini: Design and Results of TANCS-2000 Non-classical (Modal) Systems Comparison. R. Dyckhoff., Ed., *TABLEAUX-2000*, LNAI 1847, pp. 50–56. Springer, 2000.

[22] C. Kreitz, J. Otten. Connection-based Theorem Proving in Classical and Non-classical Logics. *Journal of Universal Computer Science* 5, 88–112 (1999).

[23] R. C. Moore. A formal theory of knowledge and action. In J.R. Hobbs, R.C. Moore, Eds., *Formal Theories of the Commonsense World*, pp. 319–358. Ablex Pubs., 1985.

[24] J. Otten. ileanTAP: An Intuitionistic Theorem Prover. In D. Galmiche, Ed., *TABLEAUX-97*, LNAI 1227, pp. 307–312. Springer, 1997.

[25] J. Otten. leanCoP 2.0 and ileanCoP 1.2: High Performance Lean Theorem Proving in Classical and Intuitionistic Logic. In A. Armando, P. Baumgartner, G. Dowek, Eds., *IJCAR 2008*, LNCS 5195, S. 283–291. Springer, 2008.

[26] P. F. Patel-Schneider, R. Sebastiani. A New General Method to Generate Random Modal Formulae for Testing Decision Procedures. *Journal of Articial Intelligence Research*, 18:351–389, 2003.

[27] R. Petrick, F. Bacchus. A knowledge-based approach to planning with incomplete information and sensing. In *AIPS-2002*, pp. 212–221.AAAI Press, 2002.

[28] S. Popcorn. *First Steps in Modal Logic*. Cambridge University Press, 1994.

[29] T. Raths, J. Otten. The QMLTP Library: Benchmarking Theorem Provers for Modal Logics. Technical Report, University of Potsdam, 2011.

[30] T. Raths, J. Otten, C. Kreitz. The ILTP Problem Library for Intuitionistic Logic. *Journal of Automated Reasoning*, 38(1–3): 261–271, 2007.

[31] W. Reif, G. Schellhorn, K. Stenzel. Proving System Correctness with KIV 3.0. In W. McCune, Ed., *CADE-14*, LNAI 1249, pp. 69–72. Springer, 1997.

[32] R. Reiter. What Should a Database Know? *Journal of Logic Programming* 14(1–2):127–153, 1992.

[33] D. Sadek, P. Bretier, F. Panaget. ARTIMIS: Natural dialogue meets rational agency. In *IJCAI-1997*, pp. 1030–1035, 1997.

[34] S. Schulz. E - a brainiac theorem prover. *AI Communications*, 15(2):111–126, 2002.

[35] T. Sider. *Logic for Philosophy*. Oxford University Press, 2009.

[36] P.A. Spruit, R.J. Wieringa, J. Meyer. Regular database update logics. *TCS*, 254(1-2):591–661, 2002.

[37] M. Steedman, R. Petrick. Planning Dialog Actions. In S. Keizer, H. Bunt, T. Baek, Eds., *SIGdial 2007*, pp. 265–272, 2007.

[38] M. Stone. Abductive Planning With Sensing. In *AAAI-98*, pp. 631–636. Menlo Park CA., 1998.

[39] M. Stone. Towards a Computational Account of Knowledge, Action and Inference in Instructions. *Journal of Language and Computation*, 1:231–246, 2000.

[40] M. Stone, C. Doran, B. Webber, T. Bleam, M. Palmer. Microplanning with Communicative Intentions: The SPUD System. *Computational Intelligence*, 19(4):311–381, 2003.

[41] G. Sutcliffe. The TPTP Problem Library and Associated Infrastructure: The FOF and CNF Parts, v3.5.0. *Journal of Automated Reasoning*, 43(4):337–362, 2009.

[42] G. Sutcliffe. The CADE-22 automated theorem proving system competition – CASC-22.. *AI Communications*, 23(1):47–59, 2010.

[43] G. Sutcliffe. The 5th IJCAR automated theorem proving system competition – CASC-J5.. *AI Communications*, 24(1):75–89, 2011.

[44] V. Thion, S. Cerrito, M. Cialdea Mayer. A General Theorem Prover for Quantified Modal Logics. In U. Egly, C. G. Fermüller, Eds., *TABLEAUX-2002*, LNCS 2381, pp. 266–280. Springer, 2002.

[45] L. Wallen. *Automated deduction in nonclassical logic*. MIT Press, Cambridge, 1990.

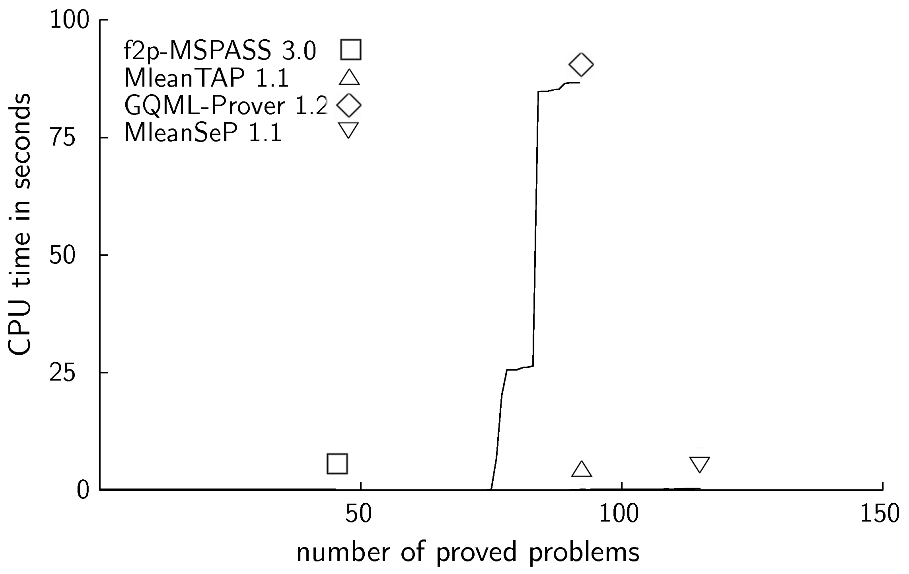


Figure 2: Performance graph for modal logic D with cumulative domains

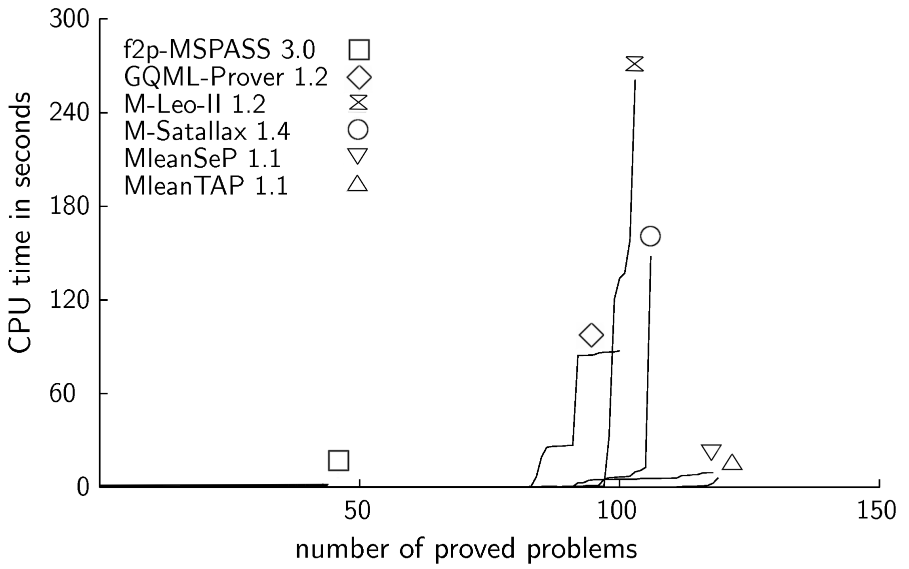


Figure 3: Performance graph for modal logic D with constant domains

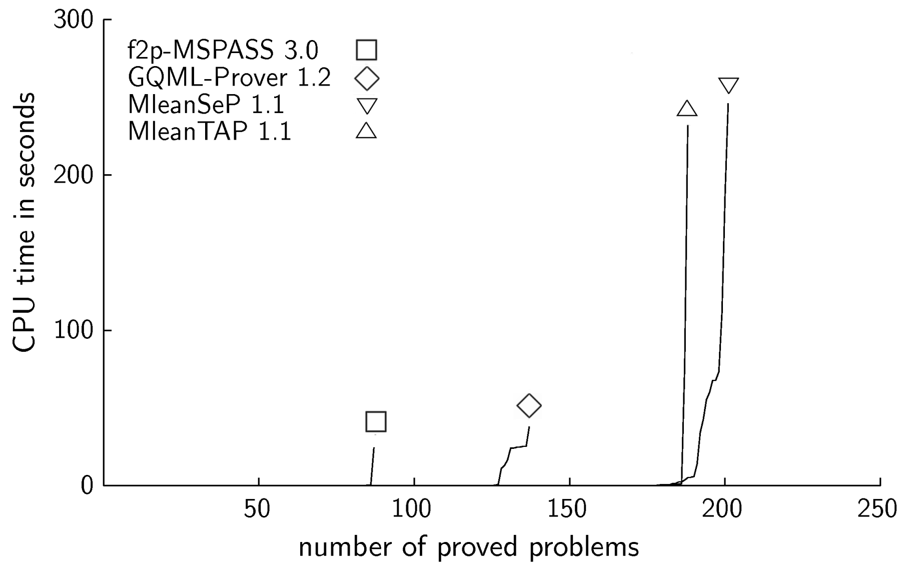


Figure 4: Performance graph for modal logic S4 with cumulative domains

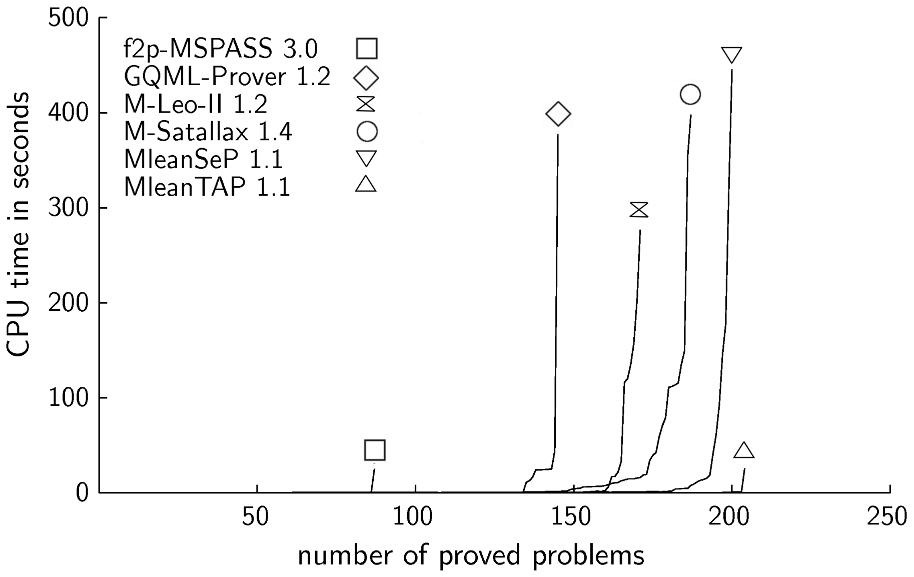


Figure 5: Performance graph for modal logic S4 with constant domains

Modular Termination and Combinability for Superposition Modulo Counter Arithmetic[*]

Christophe Ringeissen, Valerio Senni[†]
LORIA-INRIA Nancy Grand Est — E-mail: FirstName.LastName@loria.fr

Software verification tasks require the availability of solvers that are able to discharge proof obligations involving data-structures together with arithmetic constraints and other mathematical abstractions, such as size abstractions. Besides, the use of Satisfiability Modulo Theories (SMT) solvers allows us to focus on the development of satisfiability procedures for such mixed theories. In this setting, the problem of designing the satisfiability procedures is often addressed with success by using approaches based on combination.

Problems arise when we consider combinations involving theories whose signatures are non-disjoint. This is especially the case when we consider theories sharing some algebraic constraints. In order to combine satisfiability procedures for the single theories to handle constraints in their non-disjoint union one needs to rely on powerful methods such as the combination framework of [3]. These methods are based on semantic properties of the considered theories, such as compatibility and computability of bases of the shared entailed equalities, which often require complex proofs.

A further issue concerns the development of correct and efficient satisfiability procedures for the single theories, possibly using a systematic approach. In this regard, the use of superposition calculus has proved to be effective to deal with classical data structures (e.g., lists and records)and integer offsets [1, 2].

In this paper we address both aspects by: (1) considering a superposition calculus with a built-in theory of counter arithmetic [4] and (2) providing modularity results for termination and combinability, based on conditions on the saturations of the component theories that can be checked automatically.

Our contributions are twofold. First, we prove a modular termination result for extending the applicability of the superposition calculus to theories that share a theory of counter arithmetic. This generalizes, to the non-disjoint case, the results in [1], where the authors consider the standard superposition calculus and signature-disjoint theories. This result allows us to drop some of the complex conditions required by the combination framework when we deal with theories that can be treated uniformly through superposition.

Second, we prove a general compatibility result that allows us to use our superposition-based satisfiability procedures into the combination framework

[*]An extended version of this abstract can be found in [5].

[†]The author acknowledges support from ERCIM during his stay at LORIA-INRIA.

of [3]. We prove that any satisfiability procedure obtained by using our modular termination result is able to compute a finite basis of the shared entailed equalities. In addition, we provide a sufficient condition on the form of the saturations that allows us to conclude compatibility of the theories and, thus, completeness of their combination.

As an outcome, we have less and simpler restrictions on combinability and we are able to obtain satisfiability procedures both by a uniform approach for theories treated through superposition (e.g., data structures) and by combination for theories which are not 'superposition-friendly' (e.g., theories of arithmetic).

Application

To show the application of our results in practice, we introduce a class of new theories modeling data structures and equipped with a counting operator that allows us to keep track of the number of the modifications (writes, constructors, etc.) performed on a data structure. In these theories we are able to distinguish between versions of the same data structure obtained by some update. In particular, we consider a theory of lists T_{LV} (including the classic car and cons operators) and a theory of records T_{RV} (including the the classic $\mathsf{rselect_i}$ and $\mathsf{rstore_i}$ operators), both equipped with a counting operator: $\mathsf{count}_R(r)$, which denotes the number of updates performed on the record r, and $\mathsf{count}_L(l)$, which denotes the number of elements inserted into the list l and coincides with the size of the list. Our modular termination result applies to $T_{LV} \cup T_{RV}$, which means that our superposition calculus leads a $T_{LV} \cup T_{RV}$-satisfiability procedure. Then, our general compatibility result applies to $T_{LV} \cup T_{RV}$. Consequently, the superposition-based $T_{LV} \cup T_{RV}$-satisfiability procedure can be combined with a theory of linear arithmetic without loss of completeness when considering a shared theory of counter arithmetic. This leads to a combined decision procedure of practical interest for verifying programs with lists and records.

References

[1] Alessandro Armando, Maria Paola Bonacina, Silvio Ranise, and Stephan Schulz. New results on rewrite-based satisfiability procedures. *ACM Trans. Comput. Log.*, 10(1), 2009.

[2] Alessandro Armando, Silvio Ranise, and Michaël Rusinowitch. A rewriting approach to satisfiability procedures. *Inf. Comput.*, 183(2):140–164, 2003.

[3] Silvio Ghilardi. Model-theoretic methods in combined constraint satisfiability. *J. Autom. Reasoning*, 33(3-4):221–249, 2004.

[4] Enrica Nicolini, Christophe Ringeissen, and Michaël Rusinowitch. Combining satisfiability procedures for unions of theories with a shared counting operator. *Fundam. Inform.*, 105(1-2):163–187, 2010.

[5] Christophe Ringeissen and Valerio Senni. Modular Termination and Combinability for Superposition Modulo Counter Arithmetic. To appear in the Proceedings of the 8th International Symposium on Frontiers of Combining Systems (FroCoS 2011). Also in Research report, INRIA, 2011.

Author Index

An, Xuanye	7
Antonsen, Roger	45
Aravantinos, Vincent	16
Erbatur, Serdar	31
Fujita, Hiroshi	7
Giese, Martin	45
Hansen, Christian Mahesh	45
Hasegawa, Ryuzo	7
Klaedtke, Felix	6
Koshimura, Miyuki	7
Lynch, Christopher	31
Narendran, Paliath	31
Otten, Jens	47
Peltier, Nicolas	16
Raths, Thomas	47
Ringeissen, Christophe	64
Senni, Valerio	64
Waaler, Arild	45

Part II
Gentzen Systems and Beyond 2011 International Workshop

Preface

This booklet contains the abstracts of presentations from the Second International Workshop Gentzen Systems and Beyond, held on July 4, 2011 in Bern, Switzerland. The aim of the workshop was to explore and compare the motivations for and relative merits of the various formalisms that have emerged as a result of attempts to construct proof systems for logics that do not seem to admit a standard Gentzen-style system, including but not limited to hypersequents, display calculi, labeled deductive systems, tableaux, deep inference systems, and proof nets. The workshop was co-located with the 20th International Conference on Automated Reasoning with Analytic Tableaux and Related Methods (Tableaux 2011). The first edition of the workshop was co-located with Tableaux 2009, held in Oslo, Norway.

The program of the workshop included six contributed talks, as well as invited talks “Nested Sequents and Prefixed Tableaus” by Melvin Fitting and “A Symmetric Natural Deduction” by Michel Parigot.

We would like to thank all the contributors for their efforts.

July 2011 *Roman Kuznets*
Bern *Richard McKinley*

Nested Sequents and Prefixed Tableaus

Melvin Fitting

Abstract: It is common knowledge that tableaus and sequent calculi are dual proof mechanisms. A proof using one methodology can be seen as a proof in the other, "upside-down." In the 1970's I introduced so-called *prefixed tableaus*, which supply proof procedures for several modal logics that don't have tableau systems in a more conventional sense. These were then given very nice modular style rules by Massacci. Prefixed tableaus involve extra machinery, and it has been a nagging question what a sequent counterpart might be. Very recently Kai Brünnler has been investigating something called *nested sequent calculi*. In these, reasoning is not just at the top level, but can occur deep inside a formula. It turns out these are what was wanted all along. The fit between prefixed tableaus and nested sequents is simple, and similar to that between ordinary tableaus and sequent calculi. Since prefixed tableaus have been around for some time, we can use the connection to create a number of interesting nested sequent proof systems. I will present the background, as well as recent work on modal systems. In addition, I will also discuss a simple and interesting nested sequent system for constant domain intuitionistic propositional logic.

Constructive Realization in Justification Logics via Nested Sequents

Remo Goetschi

Justification logics are refinements of modal logics where modalities are replaced by justification terms. Given a modal formula such as $\Box A$, which can be read as *A is provable* or as *A is known*, a justification counterpart of this formula of the form $t : A$ can be read as *t is a proof of A* or as *A is known for reason t*. Many modal logics have one or several *corresponding* justification logics. They are connected via so-called realization theorems. The first such theorem was established by Artemov for the modal logic $\mathsf{S4}$ and the justification logic LP, called the *Logic of Proofs*. It has two directions. First, each provable formula of $\mathsf{S4}$ can be turned into a provable formula of LP by realizing instances of modalities with justification terms. Second and vice versa, if all terms in a provable formula of LP are replaced with modalities, then the resulting modal formula is provable in $\mathsf{S4}$. Similar correspondences have been established for several other modal logics besides $\mathsf{S4}$, either via a constructive method based on cut-free Gentzen systems or via a non-constructive method based on Kripke-style semantics for justification logics. However, until recently there was no uniform method of proving the realization theorem for all the 15 normal modal logics formed from the axioms $\mathsf{d}, \mathsf{t}, \mathsf{b}, \mathsf{4}$, and $\mathsf{5}$. In particular, there was no realization theorem for modal logics that lack cut-free sequent systems, such as $\mathsf{K5}$, $\mathsf{D5}$ and KB.

We present a general realization method that works for a wide class of cut-free *nested sequent calculi* à la Kai Brünnler. Nested sequents are a generalization of (one-sided) Gentzen sequents in the sense that they have an additional structural connective for $\Box$ which can be arbitrarily nested. A crucial feature of these proof systems is *deep inference*, which is the ability to apply inference rules to formulas arbitrarily deep inside a nested sequent. In particular, our method applies to a system by Brünnler that captures all the 15 normal modal logics from the "modal cube", which allows us to prove a uniform and constructive realization theorem for these logics.

Based on our realization method, we discuss the question of modularity of realizations: each modal axiom has a natural justification counterpart. However, one modal logic may have several axiomatizations, accordingly it is natural to suggest there to be several justification counterparts, essentially one for each axiomatization. We classify these various realizations by introducing a natural equivalence relation on them that extends that of Fitting. The naturality of the equivalence relation here means that justification logics are equivalent iff they realize the same modal logics.

This is a joint work with Kai Brünnler and Roman Kuznets (both from University of Bern).

Definite and Indefinite Descriptions

Norbert Gratzl

In this talk I'd like to present the theories of definite descriptions as formulated by Russell in the *Principia Mathematica* as well as his – not so well known – account of indefinite (or ambiguous) descriptions that were briefly explained in his *Introduction to Mathematical Philosophy*.

Russell's theory of definite descriptions is well explored. In this talk I aim for a concise presentation of the theory of descriptions as it has been developed by Russell in the Principia Mathematica *14. Although, the approach to definite descriptions is (in)famous – or even a paradigm of philosophical analysis – we shall put forward a sample of arguments that purport the view that Russell's theory might be in need of several modifications. I will point out that some logical problems might occur if the contextual definitions for introducing definite descriptions are viewed as axioms (or rather axiom schemes). The proposed solution consists of several formal systems, growing in logical strength, that are formulated in suitable Gentzen-style sequent calculi. The cut-elimination theorem holds for each of the systems and its proof will be briefly outlined. Furthermore, the claim that definite descriptions *are incomplete symbols* is interpreted formally by stating an elimination theorem that shows in effect that each theory (developed here) is a conservative extension of (non-classical) predicate logic with identity.

Russell in his *Introduction to Mathematical Philosophy* discusses in the section "Descriptions" not solely definite descriptions but commences with indefinite descriptions. At heart of his analysis the statements containing indefinite descriptions are analysed away: "an object having property A has property B" means: "some A′s are B′s" — the expression "an object having property A" is supposed to be an indefinite description. A formulation of this claim as an axiom (scheme) gives rise to an inconsistency. However, by amending the underlying logic it can be proved that the obtained systems are consistent. The consistency result follows from the cut-elimination theorem for some Gentzen-style sequent calculi. Finally, I would like to indicate some relations to Hilbert's epsilon calculus.

In the last part of the talk I would like to present a formal system (in a Russellian spirit) that contains both, definite and indefinite descriptions.

A Tentative Atomic Calculus for Natural Deduction

Tom Gundersen and Michel Parigot

June 16, 2011

Abstract

We present a term calculus with explicit contraction and weakening, which is typed by deep-inference derivations. Using a version of the *medial* rule of deep-inference we give a set of reduction rules that are atomic, in the sense that they never copy or erase unbounded subterms. Finally, we give an interpretation of our calculus in terms of the lambda calculus and show how our reduction can simulate beta reduction.

A symmetric natural deduction

Michel Parigot

June 20, 2011

Having a Natural Deduction system where the rules for disjunction are simply the dual of the rules for conjunction, is an old dream of structural proof theorists. We show that the Open Deduction formalism for Deep Inference provides a way of constructing such a system and discuss some of its properties.

Sequent Calculus for Justifications

Yury Savateev

The Logic of Proofs was introduced by Artemov in order to give a provability semantics to the modal logic S4. It is based on a notion of proof polynomial, which allows to talk about proofs and propositions in the same language. In a sense it is a refinement of modal logic — a formula $\Box A$, which is typically interpreted as *A is provable* or *A is known* is replaced by an expression of the form $t : A$, that can be read as *t is a formal proof of A* or *A is known because of the evidence provided by t.* Several variants of models and realization technics were later developed for different versions of this logic.

Since in these logics proof polynomial typically denote proofs in a Hilbert-style system, the sequent calculi developed for them do not internalize their own proofs and it is hard to describe and explore the relation between proof polynomials and cut elimination and other important properties of proofs in Gentzen-style systems. Therefore there is a need for a sequent calculus that can talk about its own proofs. Some attempts in that direction were made by Brezhnev, but system presented there did not have the subformula property and did not include the function symbol for cut, and thus only could formalize cut-free proofs.

The system presented in this paper can formalize all its own proofs. The main difference from the previous approaches is that here the construction $t : A$ is not used. Instead, proof polynomial are considered to be formulas themselves. When we write a proof polynomial t in this system we intend it to be read as *t is a valid proof in our system.* Proof polynomials here are designed to contain all available information about the proof. In particular, it is always possible to determine which sequent is proved by a given proof polynomial, thus there is no need to specify it. This idea allows us to build a cut-free system that satisfy a version of the subformula property.

The calculus presented in this paper can realize the modal logic K45. As a part of realization procedure we translate our formulas into the nested sequent calculus — a deep inference system for modal logics introduced by Brünnler.

Some Remarks on Nested Sequent Systems for Modal Logics

Lutz Straßburger

INRIA Saclay – Île-de-France

École Polytechnique — LIX — Rue de Saclay — 91128 Palaiseau Cedex — France
`http://www.lix.polytechnique.fr/~lutz`

Abstract. Standard sequent calculus has difficulties with many modal logics. Recently, many authors have indepently proposed the notion of nested sequents to present cut-free deductive systems for various modal logics. Some progress has been made towards a modularity in the presentation: A given Hilbert-axiom is translated into a set of inference rules, where a difference can be made between so-called diamond-rules and so-called structural rules. In this talk, I will present the state of the art, exhibit some unexpected difficulties, and show some work in progress in reaching full modularity.

On the correspondence between display postulates and deep inference in nested sequent calculi for tense logics

Alwen Tiu
The Australian National University
(Joint work with Rajeev Goré and Linda Postniece)

Abstract

We consider two styles of proof calculi for a family of tense logics, presented in a formalism based on nested sequents. A nested sequent can be seen as a tree of traditional single-sided sequents. Our first style of calculi is what we call *shallow calculi*, where inference rules are only applied at the root node in a nested sequent. Our shallow calculi are extensions of Kashima's calculus for tense logic and share an essential characteristic with display calculi, namely, the presence of structural rules called *display postulates*. Shallow calculi enjoy a simple cut elimination procedure, but are unsuitable for proof search due to the presence of display postulates and other structural rules. The second style of calculi uses deep-inference, whereby inference rules can be applied at any node in a nested sequent. We show that, for a range of extensions of tense logic, the two styles of calculi are equivalent, and there is a natural proof theoretic correspondence between display postulates and deep inference. The deep inference calculi enjoy the subformula property and have no display postulates or other structural rules, making them a better framework for proof search.

Part III
Tutorials

Tableaux(-like) Methods for the Satisfiability Problems of Temporal Logics

Martin Lange

January 12, 2011

1 Content

Temporal logics are modal logics over infinite flows of time. They form important tools for the specification of program behaviour. Their satisfiability problems therefore form the algorithmic essence of consistency checks for logical specifications of correct program behaviour.

Temporal operators usually have elegant characterisations in terms of fixpoint solutions to certain recursive equations. This often makes the evaluation of temporal formulas in Kripke structures relatively simple using fixpoint iteration techniques. It also introduces very particular difficulties for deciding satisfiability: one has to ensure that iterations corresponding to least fixpoint constructs are well-founded.

In this tutorial we will review basic temporal logics and tableau-based methods for their satisfiability problems. We will illustrate the problems arising with a mixture of least and greatest fixpoint constructs in such tableaux and discuss known solutions as well as related and open questions in this area.

2 Level

The tutorial will be held at an introductory level. No particular knowledge about temporal logics will be required. However, participant will be expected to have some knowledge and logic in general. Some knowledge on automata theory will be helpful but is not essential.

3 Relevance

The tutorial introduces a major application area for tableaux as a tool for logical decision procedures.

4 Interest

The tutorial will be of interest to young researchers attending TABLEAUX'11. It covers topics that usually do not occur in courses at (under)graduate level. The open problems and related areas to be dealt with in the tutorial may give enthusiatic researchers ideas for future work.

Tutorial: Introduction to Proof Nets

Lutz Straßburger

INRIA Saclay – Île-de-France — Équipe-projet Parsifal

École Polytechnique — LIX — Rue de Saclay — 91128 Palaiseau Cedex — France
`lutz at lix dot polytechnique dot fr`

This tutorial is intended to be a basic introduction to proof nets. The term *proof net* has been coined by Girard [Gir87] for his *bureaucracy-free* presentation of proofs in linear logic. He used the term *bureaucracy* for the phenomenon of *trivial rule permutations* in the sequent calculus that do not change the essence of a proof. From the beginning, proof nets served two purposes: first, provide a concise presentation of proofs, similar to λ-terms for intuitionistic logic, and second, to simplify the cut elimination proof and to study the normalization of proofs.

In the first part of the tutorial I will introduce proof nets for various fragments of linear logic and discuss various *correctness criteria* [DR89,HvG03].

In the second part of the tutorial I will present more recent developments in the area of classical logic. There are, on one side, proof nets that follow the tradition of the linear logic proof nets [Rob03], and on the other side, there are proof net variants [LS05] that follow the tradition of Andrews' *matings* [And76], Bibel's *matrix proofs* [Bib81], and Buss' *logical flow graphs* [Bus91]. The normalization behaviour of these objects is very different from the usual one in the sequent calculus.

A particularly interesting variant are *atomic flows* [GG08,GGS10] because they are subject to transformations that can be lifted to formal derivations in the deep inference deductive system SKS [BT01], and these transformations can be composed such that we obtain novel a cut-elimination procedure for SKS.

Although for the "flow graph based" variants of proof nets we do not (yet) know polynomial correctness critiera, they can be used as *invariants for proofs* [Hug06] for investigating the question of the *identity of proofs*. This allows to compare proofs in different deductive systems, for example, the same proof can be given in a natural deduction system and in a tableau system.

The tutorial will be based on material from Chapters 2 and 3 of my ESSLLI'06 lecture notes [Str06], the work in [Str10], and the Chapter 2 of my habilitation thesis [Str11].

References

[And76] Peter B. Andrews. Refutations by matings. *IEEE Transactions on Computers*, C-25:801–807, 1976.

[Bib81] Wolfgang Bibel. On matrices with connections. *Journal of the ACM*, 28:633–645, 1981.

[BT01] Kai Brünnler and Alwen Fernanto Tiu. A local system for classical logic. In R. Nieuwenhuis and A. Voronkov, editors, *LPAR 2001*, volume 2250 of *LNAI*, pages 347–361. Springer, 2001.

[Bus91] Samuel R. Buss. The undecidability of k-provability. *Annals of Pure and Applied Logic*, 53:72–102, 1991.

[DR89] Vincent Danos and Laurent Regnier. The structure of multiplicatives. *Annals of Mathematical Logic*, 28:181–203, 1989.

[GG08] Alessio Guglielmi and Tom Gundersen. Normalisation control in deep inference via atomic flows. *Logical Methods in Computer Science*, 4(1:9):1–36, 2008.

[GGS10] Alessio Guglielmi, Tom Gundersen, and Lutz Straßburger. Breaking paths in atomic flows for classical logic. In *LICS 2010*, 2010.

[Gir87] Jean-Yves Girard. Linear logic. *Theoretical Computer Science*, 50:1–102, 1987.

[Hug06] Dominic J.D. Hughes. Proofs Without Syntax. *Annals of Mathematics*, 164(3):1065–1076, 2006.

[HvG03] Dominic Hughes and Rob van Glabbeek. Proof nets for unit-free multiplicative-additive linear logic. In *18th IEEE Symposium on Logic in Computer Science (LICS 2003)*, pages 1–10, 2003.

[LS05] François Lamarche and Lutz Straßburger. Naming proofs in classical propositional logic. In Paweł Urzyczyn, editor, *TLCA'05*, volume 3461 of *LNCS*, pages 246–261. Springer, 2005.

[Rob03] Edmund P. Robinson. Proof nets for classical logic. *Journal of Logic and Computation*, 13:777–797, 2003.

[Str06] Lutz Straßburger. Proof nets and the identity of proofs. Research Report 6013, INRIA, 10 2006. Lecture notes for ESSLLI'06.

[Str10] Lutz Straßburger. What is the problem with proof nets for classical logic? In Fernando Ferreira, Benedikt Löwe, Elvira Mayordomo, and Luís Mendes Gomes, editors, *Programs, Proofs, Processes, 6th Conference on Computability in Europe, CiE 2010. Proceedings*, volume 6158 of *Lecture Notes in Computer Science*, pages 406–416. Springer, 2010.

[Str11] Lutz Straßburger. *Towards a Theory of Proofs of Classical Logic.* Habiliatation à diriger des recherches, Université Paris VII, 2011.

Part IV
Short Papers

Dialogue Games for Classical Logic

Jesse Alama[1], Aleks Knoks[2], and Sara L. Uckelman[2]*

[1] Center for Artificial Intelligence
New University of Lisbon
`j.alama@fct.unl.pt`
[2] Institute for Logic, Language, and Computation
Universiteit van Amsterdam
`knoks@science.uva.nl;S.L.Uckelman@uva.nl`

Abstract. We define a class of dialogue games and prove that existence of winning strategies for the Proponent in this class of games corresponds to validity in classical propositional logic. Many authors have stated similar results without actually proving the correspondence. We modify the games used for intuitionistic logic given by Fermüller [3]. We employ standard dialogue games and a standard sequent calculus for classical logic. The result is a simple correspondence between dialogue games and classical logic.

1 Introduction

Dialogue games as a semantics for intuitionistic logic (IL) were developed by Lorenzen in the 1950s as an alternative to the operative approach to logic [8, 9]. A *dialogue game* is a finitary open two-person zero-sum game between the Proponent **P** and the Opponent **O**. Lorenzen's goal was to isolate a certain class of dialogue games such that **P** has a winning strategy for the dialogue game beginning with φ iff φ is a theorem of IL.

The first recognized successful proof for IL was given by Felscher [2]. Felscher's proof, though correct, is both complicated, with its introduction of the notion of protableaux, and difficult to understand. Fermüller [3] provides a variant of the E rules which allows him to give a simpler proof of the correspondence, without intermediate recourse to protableaux (the proof also appears in [4]). Our goal in this paper is to modify Fermüller's games and prove that these modified games characterize classical validity; we do this in §3.

2 Previous work

Numerous classes of dialogue games for classical logic have been defined in the literature, but with few exceptions, the correspondence between the class of games

* The first and third authors were funded by the FCT/NWO/DFG project "Dialogical Foundations of Semantics" (DiFoS) in the ESF EuroCoRes programme LogICCC (FCT LogICCC/0001/2007; LogICCC-FP004; DN 231-80-002; CN 2008/08314/GW). The authors would like to thank Morton Heine Sørensen for comments on an earlier version of this paper.

and classical validity is merely asserted, and not proved [1, §2], [5, pp. 352–53], [6, pp. 217–18], [7, p. 305], [9, p. 194], [10, §1.5], [11, §2], [12, passim], [14, p. 152]. The three exceptions are due to Fermüller and Sørensen & Urzyczyn. In [3], Fermüller defines classes of "parallel" dialogue games which capture various intermediate logics, including classical logic, characterized by various hypersequent calculi. In [13], Sørensen and Urzyczyn offer an elegant and compact correspondence between Felscher's dialogues for classical logic and a "dialogue-inspired" variant of the sequent calculus **LK** for classical logic that they call **LKD**. The calculus **LKD** has only three rules: a left-rule, a right-rule, and one structural rule, cut, which can be eliminated [13, Cor. 3.4]. The left and right rules have side conditions that are built directly from the dialogue rules for attacks and defenses of formulas; this justifies the presence of only two logical rules in the calculus, rather than the customary array of left and right rules for each of the connectives.

Our result in this paper improves on both Fermüller's and Sørensen and Urzyczyn's proofs. We show that, in the classical case, Fermüller's use of parallel dialogue games and hypersequents can be avoided, and that a correspondence with a standard sequent calculus, rather than Sørensen and Urzyczyn's "dialogue-inspired" version can be established.

3 Classical dialogical logic

Our language is a basic propositional language with a designated atom $\bot$ (*falsum*). We define $\neg\varphi$ as $\varphi \to \bot$. We will also make use of so-called *symbolic attacks*, $\wedge_L$, $\wedge_R$, and ?.

Dialogue games are specified by two types of rules, *particle* (local) rules and *structural* (global) rules. Particle rules (see Table 1) give the attack and defense conditions for each type of formula; note that we allow attacks on atoms, but these attacks cannot be defended. Structural rules define which sequences of dialogical moves will count as legal dialogues. A *dialogue* is a sequence of attacks and defenses that begins with a finite (possibly empty) multiset Π of formulas that are *initially granted* by **O** and a finite (nonempty) multiset Δ of formulas that are *initially disputed* by **O**. Formulas that have been initially granted by **O** can be attacked by **P** at any time, and formulas that are initially disputed by **O** can be asserted as a defense by **P** at any time. In the case where Δ is a singleton, we can understand the game as beginning with an assertion of Δ by **P**, with the first move then being an attack on Δ by **O**.

Definition 1 (CL **structural rules).**

Start *The first move of the dialogue is carried out by* **O** *and consists in an attack on (the unique) initially disputed formula* φ.

Alternation *Moves strictly alternate between players* **O** *and* **P**.

Atom *Atomic formulas, including* $\bot$, *may be stated by both players, but only* **O** *can attack them.*

E *Each move of* **O** *reacts directly to the immediately preceding move by* **P**.

Assertion	**Attack**	**Response**
p (atomic)	?	—
$\varphi \wedge \psi$	$\wedge_L$	φ
	$\wedge_R$	ψ
$\varphi \vee \psi$	?	φ or ψ
$\varphi \to \psi$	φ	ψ

Table 1. Particle rules for dialogue games

Definition 2 (Active formula). *The most recent formula which* **P** *has asserted that* **O** *must attack in the next round is the* active formula*, if it exists.*

Definition 3 (Winning conditions (for P)).

W^{CL} *The game ends with* **P** *winning if $\Pi \cap \Delta \neq \emptyset$.*
$W_{\bot}$ *The game ends when $\bot$ is granted.*

To establish the correspondence between classical validity and (existence of) winning strategies, we use a variant of the sequent calculus system **GKcp** [15] for classical propositional logic, which is itself a variant of the standard contraction- and weakening-friendly formulation of **LK** that copies the principal formula into the premise (or premises).

Definition 4 (The system GKcp′). *Derivable objects are sequents $\Pi \Rightarrow \Delta$, where Π and Δ are multisets of formulas. The sequent system* **GKcp′** *is specified by the axioms and rules in Figure 1, together with the usual weakening and contraction rules on both the left and the right, as well as cut.*

Axioms: $\varphi, \Pi \Rightarrow \Delta, \varphi$ and $\bot, \Pi \Rightarrow \Delta$

$$\vee\mathrm{L}\ \frac{A \vee B, A, \Pi \Rightarrow \Delta \qquad A \vee B, B, \Pi \Rightarrow \Delta}{A \vee B, \Pi \Rightarrow \Delta} \qquad \frac{\Pi \Rightarrow \Delta, A \vee B, A, B}{\Pi \Rightarrow \Delta, A \vee B}\ \vee\mathrm{R}$$

$$\wedge\mathrm{L}\ \frac{A \wedge B, A, B, \Pi \Rightarrow \Delta}{A \wedge B, \Pi \Rightarrow \Delta} \qquad \frac{\Pi \Rightarrow \Delta, A \wedge B, A \qquad \Pi \Rightarrow \Delta, A \wedge B, B}{\Pi \Rightarrow \Delta, A \wedge B}\ \wedge\mathrm{R}$$

$$\to\mathrm{L}\ \frac{A \to B, \Pi \Rightarrow \Delta, A \qquad A \to B, B, \Pi \Rightarrow \Delta}{A \to B, \Pi \Rightarrow \Delta} \qquad \frac{A, \Pi \Rightarrow \Delta, A \to B, B}{\Pi \Rightarrow \Delta, A \to B}\ \to\mathrm{R}$$

Fig. 1. Axioms and rules for **GKcp′**.

This system differs from ordinary **GKcp** in that we do not require the φ in the first axiom to be atomic.

Proposition 1. *$A, \Pi \Rightarrow A \to B, \Delta$ is provable in* **GKcp′** *iff $\Pi \Rightarrow A \to B, \Delta$ is provable in* **GKcp′**.

Proposition 2. *$A, \Pi \Rightarrow A \to B, B, \Delta$ is provable in* **GKcp′** *iff $A, \Pi \Rightarrow A \to B, \Delta$ is provable.*

Theorem 1. *Every winning strategy τ for $\Pi \vdash C, \Delta$ (i.e., for dialogue with initially disputed formula C where player O initially grants the formulas in Π and disputes the formulas in Δ) can be transformed into a* $\mathbf{GKcp'}$*-deduction of $\Pi \Rightarrow C, \Delta$.*

The proof of Thm. 1 is a straightforward adaptation of Fermüller's Thm. 1 [3]. Our proof that if φ is classically valid, then there exists a winning strategy for φ in our classical dialogue game is constructive: we will map strongly analytic $\mathbf{GKcp'}$-deductions into winning strategies.

Definition 5. *A* $\mathbf{GKcp'}$*-deduction is called* strongly analytic *if it contains no application of weakening, contraction, or cut.*

It is a well-known fact that:

Lemma 1. $\mathbf{GKcp'} \vdash \Pi \Rightarrow \Delta$ *iff there exists a strongly analytic deduction of $\Pi \Rightarrow \Delta$ in* $\mathbf{GKcp'}$ *[15].*

Lemma 2. *$\Pi \Rightarrow \emptyset$ is provable in* $\mathbf{GKcp'}$ *iff $\Pi \Rightarrow \bot$ is provable in* $\mathbf{GKcp'}$*.*

This brings us to the main result of this paper.

Theorem 2. *For every strongly analytic* $\mathbf{GKcp'}$*-deduction of $\Pi \Rightarrow \Delta$ and for every formula φ in Δ, there exists a winning strategy for the dialogue whose initial dialogue sequent is $\Pi \vdash \Delta$ and for which* **O***'s initial attack is against φ.*

Proof. In light of Lemma 2, we may assume that Δ is non-empty. The proof is by structural induction.

(1) The end-sequent $\Pi \Rightarrow \Delta$ of δ is an axiom because there exists a formula A such that $A \in \Pi$ and $A \in \Delta$. Regardless of which formula in Δ that **O** attacks initially, it is clear that after this initial move we reach a winning state for **P**.

(2) The end-sequent $\Pi \Rightarrow \Delta$ of δ is an axiom because $\bot \in \Pi$. The existence of a winning strategy here, regardless of the formula of Δ initially attacked by **O**, is as in the previous case.

(3) The final rule application of δ is $\rightarrow$ R. That is, δ ends as follows:

$$\rightarrow \mathrm{R}\,\frac{A, \Pi \Rightarrow A \rightarrow B, B, \Delta}{\Pi \Rightarrow A \rightarrow B, \Delta}$$

We build a winning strategy for $\Pi \Rightarrow A \rightarrow B, \Delta$ and for any initially disputed formula φ in $\{A \rightarrow B\} \cup \Delta$ as follows. The game begins with an attack by **O** on φ. The dialogue state is now $\Pi' \vdash A \rightarrow B, \Delta$, where Π' is Π in case φ is not an implication and $\Pi \cup \{C\}$ in case φ is an implication $C \rightarrow D$. **P** responds by asserting $A \rightarrow B$ (this is one of the initially disputed formulas). Since **P**'s move is a defense, by Rule E, in the next round **O** must attack this assertion by asserting the antecedent A. Let **P** defend against this attack by asserting B; we are now at an **O**-node and the dialogue state is $A, \Pi \vdash A \rightarrow B, \Delta'$. By the induction hypothesis, for this sequent we have a winning strategy τ. Simply glue τ to the end of the linear order of length 4 that we have defined so far. The result is a winning strategy, because we have accounted for all possible moves by **O**.

The remaining cases (for L$\rightarrow$, $\vee$ and $\wedge$) are analogous.

4 Conclusion

We have provided a class of dialogue games and proved that existence of winning strategies for the Proponent in these games corresponds to classical derivability in the sequent calculus system **GKcp′**. Our proof improves on two previous results by using standard dialogue games instead of parallel dialogue games, as well as a standard classical sequent calculus.

References

1. Clerbout, N., Gorisse, M.H., Rahman, S.: Context-sensitivity in Jain philosophy: Siddharṣigaṇi's *Commentary on the Handbook of Logic*. Journal of Philosophical Logic pp. 1–30 (2010)
2. Felscher, W.: Dialogues, strategies, and intuitionistic provability. Annals of Pure and Applied Logic 28, 217–254 (1985)
3. Fermüller, C.G.: Parallel dialogue games and hypersequents for intermediate logics. In: Mayer, M.C., Pirri, F. (eds.) TABLEAUX 2003. pp. 48–64 (2003)
4. Fermüller, C.G., Ciabattoni, A.: From intuitionistic logic to Gödel-Dummett logic via parallel dialogue games. In: Proceedings of the 33rd IEEE International Symposium on Multiple-valued Logic. pp. 188–195. IEEE Computer Society (2003)
5. Galmiche, D., Larchey-Wendling, D., Vidal-Rosset, J.: Some remarks on the relations between proofs and games. In: Bour, P., Rebuschi, M., Rollet, L. (eds.) Construction, pp. 339–357. College Publications (2010)
6. Kamlah, W., Lorenzen, P.: Logische Propädeutik: Vorschule des vernünftigen Redens. Bibliographisches Institut, Wissenschaftsverlag (1973)
7. Krabbe, E.C.W.: Formal systems of dialogue rules. Synthese 63, 295–328 (1985)
8. Lorenzen, P.: Einführung in die operative Logik und Mathematik. Springer (1955)
9. Lorenzen, P.: Logik und Agon. In: Atti del XII Congresso Internazionale di Filosofia. pp. 187–194. Sansoni (1960)
10. Rahman, S., Clerbout, N., Keiff, L.: On dialogues and natural deduction. In: Primiero, G. (ed.) Acts of Knowledge, History, Philosophy, Logic. College Publications (2009)
11. Rebuschi, M.: Implicit vs. explicit knowledge in dialogical logic. In: Majer, O., Pietarinen, A.V., Tulenheimo, T. (eds.) Games: Unifying Logic, Language, and Philosophy, pp. 229–246. Springer (2009)
12. Rückert, H.: Dialogues as a Dynamic Framework for Logic. Ph.D. thesis, Universiteit Leiden (2007)
13. Sørensen, M.H., Urzyczyn, P.: Sequent calculus, dialogues, and cut elimination. In: Barendsen, E., Geuvers, H., Capretta, V., Niqui, M. (eds.) Reflections on Type Theory, λ-Calculus, and the Mind, pp. 253–261. Universiteit Nijmegen (2007)
14. Stegmüller, W., Varga von Kibéd, M.: Strukturtypen der Logik. Springer (1984)
15. Troelstra, A.S., Schwichtenberg, H.: Basic Proof Theory. Cambridge University Press, 2nd edn. (2000)

A dynamic programming algorithm for prime implicates[⋆]

Andrew Matusiewicz

SUNY Albany CS Department

1 Introduction

Prime implicants were introduced by Quine [6] as a means of finding minimal representations of boolean functions, and since then the problem of finding prime implicants of a boolean function has been the subject of many algorithms and papers [7, 8, 1, 3]. The dual problem of finding prime implicates (minimal clauses implied by a formula) has use in contexts such as belief revision [10], abductive reasoning, and recently, polysynchronous systems via the work of Shukla et. al. [2].

Here we address the problem of generating all prime implicates of a CNF formula and present an algorithm parameterized in the channelwidth of the formula, a value corresponding to the treewidth of a graph associated with the formula. We believe this to be the first algorithm for this problem so parameterized.

The problems of finding prime implicates from CNF and finding prime implicants from DNF are equivalent via duality and thus this algorithm applies to the latter, more traditional problem as well. The algorithm is presented in its "implicate-CNF" form to support Sandeep Shukla's work on polysynchronous systems.

2 Preliminaries

For substitutions on boolean formula we use notation from [9], modified for CNF. For a set of variables V, $\Gamma(V)$ is the set of all complete assignments mapping V to $\{0,1\}$. $\Gamma(\varnothing)$ has exactly one member — the empty assignment. The restriction of the assignment γ to the set of variables V is expressed as $\gamma|_V$.

For a formula F, $F[c/v]$ is F under the substitution of constant $c \in \{0,1\}$ for variable v. Similarly, for $\gamma \in \Gamma(V)$, $F[\gamma]$ is F under the assignment γ.

A **clause** is a disjunction of propositional literals. A clause C **subsumes** a clause D iff the literals of C are a subset of those of D. Here subset is not strict, so any clause subsumes itself. All clauses are subsumed by the empty clause, denoted as $\Box$, which is equivalent to "false".

An **implicate** of a formula is a clause implied by the formula, and we call an implicate **prime** if it is subsumed by no other. Let $\mathcal{P}(F)$ be the set of prime implicates of the formula F.

⋆ This research was supported in part by the National Science Foundation under grants IIS-0712849 and IIS-0712752, and by the Air Force Research Laboratory, Award No. FA8750-10-1-0206, Subaward No. 450044-19191.

2.1 Structure Trees

Suppose we have a clausal formula $F = \{C_1, C_2, \ldots, C_m\}$ with variables $V = \{v_1, v_2, \ldots, v_n\}$. A **structure tree** for F is a triple $\langle T, A, B\rangle$ where

- T is a rooted tree with nodes I
- $A : I \rightarrow 2^V$ is such that $\{A(i)|i \in I\}$ partitions V
- $B : I \rightarrow 2^F$ is such that $\{B(i)|i \in I\}$ partitions F

In addition, for each clause $C \in B(i)$ if one of its variables v is in $A(j)$ then j must be an ancestor of i. We say the terms "descendant and "ancestor" include the node in question, so that any node is both a descendant and ancestor of itself. We say that A and B **locate** variables and clauses at each node, so $A(i)$ and $B(i)$ are the variables and clauses located at i.

A few associated definitions will be useful:

- The **channel variables** $CV(i)$ of a node i are those occurring in $A(j)$ for some ancestor j of i and also in some clause in $B(k)$ for a descendant of k of i.
- For node i, the **descendant formula** F_i is a formula consisting of all the clauses mapped to decendants of i.

The **channelwidth** of a structure tree S is $CW(S) = \max_{i \in I} |CV(i)|$, and the channelwidth of a formula F, $CW(F)$, is the minimum channelwidth over all structure trees of that formula. A structure tree whose channelwidth achieves this minimum is an **optimal** structure tree.

The following lemma, proven in [9], describes the separation properties induced on a CNF formula by a structure tree.

Lemma 1. *For an internal node i of a structure tree having distinct children j_1, j_2, their descendant formula F_{j_1} and F_{j_2} may share only the channel variables of i.*

Stearns and Hunt introduce structure trees in [9] and show that for a formula F, $CW(F) - 1$ is equal to the treewidth of the interaction graph of the variables — a graph of variables connected iff they appear together in some clause. Also, the width-preserving transformation from a tree decomposition of the interaction graph to a structure tree is simple and efficient, allowing existing tree decomposition algorithms to be applied to finding good structure trees.

3 Tables and their Scopes

In this section we describe a dynamic programming (DP) algorithm for prime implicates that employs a structure tree of the CNF formula in question. Like most such algorithms, this one is built around the construction of tables.

Let F be a formula and V a set of variables. Consider a function t from $\Gamma(V)$ to clause sets such that $\mathsf{t}(\gamma) = \mathcal{P}(F[\gamma])$. We will call t the **table for F with scope V**. It is obvious that such a function is well-defined and unique, so we use the definite article. Each individual mapping from an assignment to a clause set we refer to as an **entry** in the table.

Given a structure tree $S = \langle T, A, B\rangle$, the **table for node** i, denoted $\mathfrak{T}_i$, is simply the table for F_i with scope $CV(i)$. It is these tables that our dynamic programming algorithm will construct recursively.

3.1 The $\mathcal{S}$ Operator and Scope Adjustment

We have established in previous papers [4, 5] the existence of a $O(n^4)$ algorithm mapping $P_0 = \mathcal{P}(F[0/v])$ and $P_1 = \mathcal{P}(F[1/v])$ to $\mathcal{P}(F)$. We can express this algebraically as $\mathcal{S}(P_0, P_1, v) = \mathcal{P}(F)$ where $\mathcal{S}$ is the operation in question.

We may extend the usage of $\mathcal{S}$ to a table with scope V, yielding another table with a smaller scope. We call this extension *ContractOne* and define it as follows:

$$\mathit{ContractOne}(\mathfrak{t}, v) \ := \ \{\gamma \mapsto \mathcal{S}(\mathfrak{t}(\gamma_0),\ \mathfrak{t}(\gamma_1),\ v) \mid \gamma \in \Gamma(V \setminus \{v\})\}.$$

where γ_c is $\gamma \cup \{v \mapsto c\}$ for $c \in \{0, 1\}$. Note that $\mathfrak{t}(\gamma_c) = \mathcal{P}(F[\gamma][c/v])$ and thus, denoting $\mathcal{P}(F[\gamma][c/v])$ with $P_{\gamma c}$, $\mathit{ContractOne}(\mathfrak{t}, v)(\gamma) = \mathcal{S}(P_{\gamma 0},\ P_{\gamma 1},\ v) = \mathcal{P}(F[\gamma])$. This operation may be regarded as a single variable "contraction" of $\mathfrak{t}$ along v, as $\mathit{ContractOne}(\mathfrak{t}, v)$ is precisely the table for F with scope $V \setminus \{v\}$.

With this in mind we define two utility functions on tables: "*Expand*" and "*Contract*". *Expand* produces a new table which has a scope that is a superset of the given table and *Contract* produces a table with scope a subset of the original table.

Definition 1. *Where* $\mathfrak{t}$ *is the table for* F *with scope* V *and* $W \subseteq V$*, with* $V \setminus W = \{v_{i_1}, \ldots, v_{i_k}\}$*,*

$$\mathit{Contract}(\mathfrak{t}, W) := \mathit{ContractOne}(\ldots \mathit{ContractOne}(\mathfrak{t}, v_{i_1}) \ldots, v_{i_k}).$$

Definition 2. *Where* $\mathfrak{t}$ *is the table for* F *with scope* V *and* $W \supseteq V$*,*

$$\mathit{Expand}(\mathfrak{t}, W) := \{\gamma \mapsto \mathfrak{t}(\gamma|_V) \mid \gamma \in \Gamma(W)\}.$$

Contract always produces precisely the table for F with scope W — this follows directly from the correctness of *ContractOne*. *Expand* only preserves F when the new variables in W do not occur in F, and under this constraint its correctness is trivial.

Combining *Contract* and *Expand*, we may produce a simple and succinct function $\mathit{ReScope}(\mathfrak{t}, W) := \mathit{Expand}(\mathit{Contract}(\mathfrak{t}, V \cap W), W)$, which adjusts the scope of a table arbitrarily and preserves the underlying formula F under the condition that any variables added to the scope do not occur in F.

3.2 Unioning Tables

In general, the union of two sets of prime implicates is not a set of prime implicates. However, when formulas F_1 and F_2 are variable-disjoint, $\mathcal{P}(F_1) \cup \mathcal{P}(F_2) = \mathcal{P}(F_1 \wedge F_2)$[1], an easily proven fact that we make liberal use of in our algorithm.

[1] This is true in all instances except the following: $\mathcal{P}(F \wedge 0) \neq \mathcal{P}(F) \cup \mathcal{P}(0)$. This is simply because $\mathcal{P}(0) = \{\Box\}$, and the empty clause $\Box$ subsumes any clauses in $\mathcal{P}(F)$. For simplicity we will use "$\cup$" , but understand that if one of the arguments to $\cup$ is $\{\Box\}$, the result is always $\{\Box\}$.

In this vein, we extend the "$\cup$" operation to tables with the same scope in the following manner: if $\mathfrak{t}_1$ and $\mathfrak{t}_2$ are tables of F_1 and F_2 both having scope V, $\mathfrak{t} = \mathfrak{t}_1 \cup \mathfrak{t}_2$ means that $\mathfrak{t}(\gamma) = \mathfrak{t}_1(\gamma) \cup \mathfrak{t}_2(\gamma)$ for each γ in $\Gamma(V)$. As substitution of γ removes any variables in V, $\mathit{Var}(F_1) \cap \mathit{Var}(F_2) \subseteq V$ implies $\mathfrak{t}$ is a table for $F_1 \wedge F_2$ with scope V.

The condition that tables need identical scope to be combined using "$\cup$" explains the need for *Contract* and *Expand* — before combining tables, we must first alter them to the same scope.

3.3 DP algorithm for prime implicates

With the above supporting definitions, we may define our dynamic programming algorithm. To show the correctness of *DPPrimeImp*, we examine the correctness

Algorithm 1: *DPPrimeImp*(S, i)

```
input  : $S = \langle T, \alpha, \beta \rangle$ – a structure tree for $F$; $i$ – a node of $T$
output: $\mathfrak{T}_i$ – the table for node $i$
Initially, let $\mathfrak{T}_i$ be a table with scope $CV(i)$ and entries undefined;
Let $G$ be a formula with clauses $B(i)$;
for $\gamma \in \Gamma(CV(i))$ do
    if $G[\gamma] = 0$ then $\mathfrak{T}_i(\gamma) := \{\Box\}$ ;
    if $G[\gamma] = 1$ then $\mathfrak{T}_i(\gamma) := \{\}$ ;
Let $j_1, \ldots, j_n$ be the children of $i$ ;
for $j \in \{j_1, \ldots, j_n\}$ do
    Let $\mathfrak{T}_j = DPPrimeImp(S, j)$ ;                       // Recurse
    $\mathfrak{T}'_j := ReScope(\mathfrak{T}'_j,\ CV(i))$ ;         // Table for $F_j$, scope $CV(i)$
    $\mathfrak{T}_i := \mathfrak{T}_i \cup \mathfrak{T}'_j$ ;                  // Add $F_j$'s PIs to $\mathfrak{T}_i$
return $\mathfrak{T}_i$;
```

of the first and second loops separately.

In the first loop, with G defined as in the pseudocode, we construct the table for G with scope $CV(i)$. Observe that if i is a leaf node, then $F_i = G$ and $CV(i)$ is simply $\mathit{Var}(G)$. In this case constructing the table $\mathfrak{T}_i$ is a trivial matter of iterating over assignments in $\Gamma(CV(i))$ and adding the entry $\gamma \mapsto \{\}$ if $G[\gamma] = 1$ and $\gamma \mapsto \{\Box\}$ if $G[\gamma] = 0$ (recall $\mathcal{P}(1) = \{\}$ and $\mathcal{P}(0) = \{\Box\}$).

Therefore if i is a leaf, the first loop is sufficient to construct $\mathfrak{T}_i$. If i has children $j_1, \ldots, j_n$ where $n \neq 0$ then the second loop has items to iterate over and recurses on each child in turn. Obtaining each child table $\mathfrak{T}_j$ we adjust its scope to $CV(i)$ with *ReScope*. *ReScope* is valid here because any variables in $CV(i)$ not in $CV(j)$ are the result of some clause located at i, which is not contained in F_j, so any new variables introduced by *ReScope* are not in F_j.

Finally we union $\mathfrak{T}_i$ with $\mathfrak{T}'_j$, which corresponds to conjunction as for distinct children j_1, j_2, any variable shared by F_{j_1} and F_{j_2} is contained in $CV(i)$ by lemma 1. Conjoining the tables of the children to that of G reflects the fact that when i is an internal node $F_i = G \wedge F_{j_1} \wedge \ldots \wedge F_{j_n}$.

Thus at the end *DPPrimeImp*, we have that $\mathfrak{T}_i$ is indeed the table for F_i with scope $CV(i)$.

To obtain $\mathcal{P}(F)$, let r be the root node of S. We may obtain r's table, $\mathfrak{T}_r$, simply by invoking $\mathit{DPPrimeImp}(S, r)$. Since all nodes in T are descendants of r, we have $F_r = F$. Thus $\mathfrak{T}_r$ is a table for our original formula F with scope $CV(r)$.

Using $\mathit{ReScope}(\mathfrak{T}_r, \varnothing)$, we may narrow our scope to the empty set. Such a table has only one entry, that corresponding to the empty assignment $\gamma_\varnothing$. Obviously, $F_r[\gamma_\varnothing] = F_r = F$. This yields the following top-level function for the entire algorithm, and a lemma asserting its correctness.

Definition 3. *Where* $S = \langle T, \alpha, \beta \rangle$ *is a structure tree for* F*,* r *is the root of* S*, and* $\gamma_\varnothing$ *is the empty assignment,*

$$\mathit{Primes}(S) := \mathit{ReScope}(\mathit{DPPrimeImp}(S, r), \varnothing)(\gamma_\varnothing).$$

Lemma 2. *Where* S *is a structure tree for* F*,* $\mathit{Primes}(S) = \mathcal{P}(F)$*.*

References

1. Peter Jackson and J. Pais. Computing prime implicants. In *Proc. 10th International Conference on Automated Deductions, Kaiserslautern, Germany, July, 1990*, volume 449, pages 543–557, 1990. In Lecture Notes in Artificial Intelligence, Springer-Verlag, Vol. 449.
2. B.A. Jose and S.K. Shukla. An alternative polychronous model for embedded software synthesis:. Technical Report 2009-03, FERMAT Lab, Virginia Tech, (Available from `http://filebox.vt.edu/users/bijoyaj/emcodesyn.html`), 2009.
3. A. Kean and G. Tsiknis. An incremental method for generating prime implicants/implicates. *Journal of Symbolic Computation*, 9:185–206, 1990.
4. Andrew Matusiewicz, Neil V. Murray, and Erik Rosenthal. Prime implicate tries. In Martin Giese and Arild Waaler, editors, *TABLEAUX*, volume 5607 of *Lecture Notes in Computer Science*, pages 250–264. Springer, 2009.
5. Andrew Matusiewicz, Neil V. Murray, and Erik Rosenthal. Trie based subsumption and improving the pi-trie algorithm. In Renate Schmidt Boris Konev and Stephan Schulz, editors, *PAAR*, 2010.
6. W. V. Quine. The problem of simplifying truth functions. *The American Mathematical Monthly*, 59(8):521–531, 1952.
7. Anavai Ramesh, George Becker, and Neil V. Murray. Cnf and dnf considered harmful for computing prime implicants/implicates. *Journal of Automated Reasoning*, 18(3):337–356, 1997.
8. Richard L. Rudell. *Logic Synthesis for VLSI Design.* PhD thesis, EECS Department, University of California, Berkeley, 1989.
9. Richard Edwin Stearns and Harry B. Hunt III. An algebraic model for combinatorial problems. *SIAM J. Comput.*, 25(2):448–476, 1996.
10. Zhi Qiang Zhuang, Maurice Pagnucco, and Thomas Meyer. Implementing iterated belief change via prime implicates. In Mehmet A. Orgun and John Thornton, editors, *Australian Conference on Artificial Intelligence*, volume 4830 of *Lecture Notes in Computer Science*, pages 507–518. Springer, 2007.