# Proceedings of the 8th International Workshop on the Implementation of Logics

Geoff Sutcliffe Eugenia Ternovska Stephan Schulz

# Table of Contents

Three Years of Experience with Sledgehammer, a Practical Link between Automatic and Interactive Theorem Provers *(invited talk)* . . . . . 1
*Lawrence C. Paulson, Jasmin Christian Blanchette*

Exploring Steinitz-Rademacher polyhedra: a challenge for automated reasoning tools . . . . . 2
*Jesse Alama*

Tableau Calculus for Dummett Logic Based on Present and Next State of Knowledge . . . . . 3
*Guido Fiorino*

A Prolog-based Proof Tool for Type Theory $TA_\lambda$ and Implicational Intuitionistic-Logic . . . . . 4
*L. Yohanes Stefanus, Ario Santoso*

On Implementing Modular Complexity Analysis . . . . . 5
*Harald Zankl, Martin Korp*

Implementing an Efficient SAT Solver for a Probabilistic Description Logic 6
*Pavel Klinov, Bijan Parsia*

Optimizing the AES S-Box using SAT . . . . . 7
*Carsten Fuhs, Peter Schneider-Kamp*

# Three Years of Experience with Sledgehammer, a Practical Link between Automatic and Interactive Theorem Provers

Lawrence C. Paulson
Computer Laboratory
University of Cambridge, U.K.
lp15@cam.ac.uk

Jasmin Christian Blanchette
Institut für Informatik
Technische Universität München, Germany
blanchette@in.tum.de

**Abstract**

Sledgehammer is a highly successful subsystem of Isabelle/HOL that calls automatic theorem provers to assist with interactive proof construction. It requires no user configuration: it can be invoked with a single mouse gesture at any point in a proof. It automatically finds relevant lemmas from all those currently available. An unusual aspect of its architecture is its use of unsound translations, coupled with its delivery of results as Isabelle/HOL proof scripts: its output cannot be trusted, but it does not need to be trusted. Sledgehammer works well with Isar structured proofs and allows beginners to prove challenging theorems.

## 1 Introduction

Interactive theorem provers are widely used by researchers for modelling complex algorithms or systems. They typically support rich logical formalisms that include recursive functions and types. Some even make it possible to reason about a partial recursive function's domain of definition or to define a relation coinductively. The main weakness of these tools is the actual proving, which is extremely laborious.

For nearly 20 years, researchers have sought to make interactive theorem provers better at proving theorems. Much of this effort has been devoted to implementing decision procedures. Certainly decision procedures are essential, especially for arithmetic: without them, obvious identities can take hours to prove; with them, complicated facts can be proved instantly. But most problems lie beyond the scope of standard decision procedures. Automatic tools are needed that can work on any type of problem.

Automatic theorem provers (ATPs) are capable of creating long, incomprehensible chains of deduction. Many researchers have attempted to use them to support interactive theorem proving; particularly pertinent are Ahrendt et al. [1], Bezem et al. [7], Hurd [12], and Siekmann et al. [33]. But the only tool to pass the test of time is Sledgehammer [19, 28], which links Isabelle/HOL to the automatic provers E [32], SPASS [37], and Vampire [29]. Isabelle users invoke Sledgehammer routinely when undertaking difficult proofs. On a representative corpus of older proof scripts, we find that Sledgehammer can prove 38% of the goals that are not provable by the standard automatic tactics.

Sledgehammer was first released in February 2007 to users daring enough to download an Isabelle nightly build. It was announced in November 2007 as a component of Isabelle2007. This paper outlines the design goals that made Sledgehammer successful (§2) and presents an updated summary of Böhme and Nipkow's [8] empirical study (§3). It describes some of the lessons learnt in the past three years and its effect on the way Isabelle/HOL is taught (§4). Avenues for research are also discussed (§5). An earlier version of this paper was presented at the PAAR-2010 workshop [27].

## 2 Design Principles

The single most important design goal was one-click invocation. Some earlier systems required the user to gather up all the facts that could be relevant to the problem, and furthermore to reduce it to first-order form. Problem preparation could easily take hours, with no guarantee that the call to a first order theorem prover would succeed. Such a tool would be of little value to users.

The two aspects of problem preparation (translation into first-order logic and identification of relevant facts) each required a substantial research effort. The numerous choices outlined below were made on the basis of innumerable experiments that consumed many thousands of hours of processor time.

## 2.1 Translation into First-Order Logic

Most interactive theorem provers support a language much richer than that of first-order logic. Isabelle/HOL [21] supports polymorphic higher-order logic [2, 9, 10], augmented with axiomatic type classes [39].[1] Many user problems contain no higher-order features and might be imagined to lie within first-order logic; however, even these problems are full of typing information. Type information can take quadratic space [17] because every term must be annotated with its type, recursively, right down to the variables. Hurd [13] observed that omitting type information greatly improved the success rate of his theorem prover, Metis. This is hardly surprising, since the type information virtually buries the terms themselves. Hurd was able to omit type information because his proofs are reconstructed within HOL4 [10], which rejected any proofs that did not correspond to well-typed higher-order logic deductions.

Sledgehammer was always intended to rely on an analogous process of sound proof reconstruction, and from the outset it was clear that including complete type information would be unworkable. Completely omitting type information, although successful for HOL4, would not have worked for Isabelle because of its heavy use of type classes. We chose to include enough type information to enforce correct type class reasoning (the type class hierarchy is easily expressed using Horn clauses) but not to specify the type of every term [19, §4]. Some colleagues have expressed horror at the very idea of using unsound translations; the first author has written a lengthy exploration of the salient issues [17, §2.8].

Higher-order problems posed special difficulties. We never expected first-order theorem provers to perform deep higher-order reasoning, but merely hoped to automate proofs where the higher-order steps were trivial. We examined several methods of translating higher-order problems into first-order logic, allowing truth values to be the values of terms and curried functions to take varying numbers of arguments [17]. We eventually adopted a translation based on the one that we used for first-order logic, modified to introduce higher-order mechanisms (such as an "apply operator" for function values) only when absolutely necessary. We thereby eliminated our original distinction between first-order and higher-order problems. A higher-order feature within a problem affects the translation locally, yielding a smooth transition from purely first-order to heavily higher-order problems.

We also experimented with two methods of eliminating $\lambda$-abstractions in terms: by translating them into combinator form or by declaring equivalent functions. We ultimately opted for a naive translation scheme based on the combinators S, K, I, B, and C: more sophisticated schemes delivered no additional benefits. Unfortunately, our experience suggests that Sledgehammer is seldom successful on problems containing higher-order elements. Integration with a genuine higher-order automatic theorem prover, such as LEO-II [5] and Satallax [3], seems necessary. This would pose interesting problems for proof reconstruction: LEO-II's approach is to reduce higher-order problems to first-order ones by repeatedly applying specialised inference rules and then calling first-order ATPs. A LEO-II proof will therefore consist of a string of higher-order steps followed by a first-order proof. The latter part we know how to do; the crucial challenge is to devise a reliable way of emulating the higher-order steps within Isabelle.

Arithmetic remains an issue. A purely arithmetic problem can be solved using decision procedures, but what about problems that combine arithmetic with a significant amount of logic? In principle, Sledgehammer could solve such problems with the help of an ATP that combined arithmetic and logical reasoning, analogous to LEO-II's approach to higher-order logic. Current SMT solvers are probably of little

[1] Isabelle [26] is a generic theorem prover, based on a logical framework [23]. Isabelle/HOL is the specialisation of Isabelle for higher-order logic.

value, because they do not handle quantified formulas well. But progress in that field is extremely rapid, and soon this option could become attractive.

## 2.2 Relevance Filtering

Our initial goals for Sledgehammer were modest: to improve upon Isabelle's built-in automatic tools, using only the lemma libraries used by those tools. There were two libraries, one consisting of facts useful for forward and backward chaining, the other consisting of rewriting rules for simplification. Each library contained hundreds of lemmas. We discovered that automatic theorem provers could solve only trivial problems in the presence so many extraneous facts; by developing a lightweight, symbol-based relevance filter, we greatly improved the success rate [18]. Users would still have to identify relevant facts that did not belong to these lemma libraries.

Tobias Nipkow made the crucial suggestion to dispense with the lemma libraries, substituting the full collection of Isabelle theorems (around 10,000). This idea offered the enticing prospect that any relevant existing theorem, however obscure, could be located. We thought this goal to be unrealistic; it seemed to have too much in common with McAllester's Ontic system [14]. Ontic was intended to be able to prove mathematical results using known results that it identified automatically, and it seems fair to say that this objective was too ambitious. But Sledgehammer would only be part of the system rather than all-encompassing, and it could take advantage of 20 years of increasing hardware performance.

We were able to scale up the relevance filter to cope with the 20-fold increase in the number of facts to process. However, it relies on ad hoc heuristics that sometimes deliver poor results. Briefly, it assigns a score to every available theorem based upon how many constants that theorem shares with the conjecture; this process iterates to include theorems relevant to those just accepted, but with a decay factor to ensure termination. The constants are weighted to give unusual ones greater significance. The relevance filter copes best when the conjecture contains some unusual constants; if all the constants are common, it is unable to discriminate among the hundreds of facts that are picked up. The relevance filter is also memoryless: it has no information about how many times a particular fact has been used in a proof, and it cannot learn.

It would obviously be preferable for the automatic theorem provers themselves to perform relevance filtering. Or we could use a sophisticated system based on machine learning, such as Josef Urban's MaLARea [36], where successful proofs provide information to guide other proofs. Unfortunately, any such approach will fail given Sledgehammer's use of unsound translations. In unpublished work by Urban, MaLARea easily proved the full Sledgehammer test suite by identifying an inconsistency in the translated lemma library; once MaLARea had found the inconsistency in one proof, it easily found it in all the others. Sledgehammer is successful only because its relevance filter generally selects too few lemmas to produce an inconsistent axiom set, even with the unsound translations.

To accomplish better relevance filtering, we must decide whether to adopt a general first-order approach or to build a sophisticated relevance filter directly into Isabelle. The former approach could take advantage of the efforts of the entire ATP community, but it would have to be good enough to cope with soundly translated (and presumably enormous) formulas. The latter approach would avoid translation issues, but it would impose the entire effort onto a few Isabelle developers.

## 2.3 Proof Reconstruction

Isabelle subscribes to the LCF philosophy: all proofs ultimately reduce to primitives executed by a logical kernel. Isabelle users would not trust a tool that uncritically accepted proofs from an external source. But Sledgehammer had an even stronger design objective: to deliver Isabelle proofs in source form. We envisaged that Sledgehammer runs would demand substantial computational resources; if somebody

used Sledgehammer many times while constructing a proof, would it be feasible to run that proof again, perhaps to modify it using a laptop while at a conference? To be useful, Sledgehammer would have to return a piece of proof script that could be executed cheaply.

### 2.3.1 Reconstruction of the Resolution Proof

The original plan was to emulate the inference rules of automatic theorem provers directly within Isabelle. We should have known better: Hurd [12] had noticed that the proofs delivered by Gandalf [35] were not detailed and explicit enough. We made the same discovery with SPASS and, despite considerable efforts, were only able to reconstruct a handful of proofs [19].

We came up with a new plan: to use a general theorem prover, Metis, to reconstruct each proof step. Metis was designed to be interfaced with LCF-style interactive theorem provers, specifically HOL4. Integrating it with Isabelle's proof kernel required significant effort [28]. Metis then became available to Isabelle users, and it turned out to be capable of reconstructing proof steps easily. The output of Sledgehammer was now a list of calls to Metis, each of which proved a clause. While the output is primarily designed for replaying proofs, it also has a pedagogical value: unlike Isabelle's automatic tactics, which are black boxes, the proofs delivered by Sledgehammer can be inspected and understood.

Consider the theorem "$length\ (tl\ xs) \leq length\ xs$", which states that the tail of a list (the list from which we remove its first element, or the empty list if the list is empty) is shorter than or of equal length as the original list. The proof produced by Vampire, expressed in Isabelle's structured Isar format, looks as follows:

**proof** *neg_clausify*
  **assume** "$\neg\ length\ (tl\ xs) \leq length\ xs$"
  **hence** "$drop\ (length\ xs)\ (tl\ xs) \neq []$" **by** (*metis drop_eq_Nil*)
  **hence** "$tl\ (drop\ (length\ xs)\ xs) \neq []$" **by** (*metis drop_tl*)
  **hence** "$\forall u.\ xs\ @\ u \neq xs \vee tl\ u \neq []$" **by** (*metis append_eq_conv_conj*)
  **hence** "$tl\ [] \neq []$" **by** (*metis append_Nil2*)
  **thus** "*False*" **by** (*metis tl.simps*(1))
**qed**

The *neg_clausify* method transforms the Isabelle conjecture into negated clause form, ensuring that it has the same shape as the corresponding ATP conjecture. The negation of the clause is introduced by the **assume** keyword, and a series of intermediate facts introduced by **hence** lead to a contradiction.

This approach was inspired by the Otterfier proof transformation service [40]. Resolution proofs should ideally be translated to natural, intuitive Isabelle proofs. The best-known prior work on translating resolution proofs is TRAMP [16]; its applicability to Sledgehammer is unexplored.

Preliminary work has commenced at Munich to see to what extent resolution proofs can be transformed into intelligible proofs. The first step is to transform the proof into a direct proof by applying contraposition repeatedly and introducing case splits where appropriate. For example, the proof above is transformed into

**proof** –
  **have** "$tl\ [] = []$" **by** (*metis tl.simps*(1))
  **hence** "$\exists u.\ xs\ @\ u = xs \wedge tl\ u = []$" **by** (*metis append_Nil2*)
  **hence** "$tl\ (drop\ (length\ xs)\ xs) = []$" **by** (*metis append_eq_conv_conj*)
  **hence** "$drop\ (length\ xs)\ (tl\ xs) = []$" **by** (*metis drop_tl*)
  **thus** "$length\ (tl\ xs) \leq length\ xs$" **by** (*metis drop_eq_Nil*)
**qed**

For most Isabelle users, the direct proof is much easier to understand and maintain.

### 2.3.2 One-Line Reconstruction, or ATPs as Relevance Filters

Having Metis available made possible an entirely different approach to proof reconstruction: to throw the proof away and allow Metis to find its own proof, using the lemmas that took part in the original resolution proof [28]. For example, the Isar proofs from §2.3.1 reduce to simply

**by** (*metis append_Nil2 append_eq_conv_conj drop_eq_Nil drop_tl tl.simps*(1))

With this approach, Sledgehammer became merely a lemma finder, one that used automatic theorem provers merely as relevance filters. But this approach was generally effective and had the great advantage that each Sledgehammer call now delivered a one-line result, rather than a lengthy and often incomprehensible proof script in which all formulas were in clause form. At least one ATP implementer expressed disbelief that his system could be used merely as a relevance filter, but this approach allows any ATP to be used provided it returns the list of axioms used in its proof.

Metis sometimes fails to reconstruct the result of a Sledgehammer call within a reasonable time. Reconstruction necessarily fails if the resolution proof does not correspond to a well-typed Isabelle proof (recall that, normally, Sledgehammer omits most type information when translating Isabelle formulas into first-order logic). This type of failure could be eliminated by using sound translations, but the overall success rate would actually decrease considerably. The seasoned user eventually learns to recognise unsound proofs—certain lemmas always seem to be mentioned. The number of such proofs is reduced by removing certain lemmas from the scope of Sledgehammer, both manually and automatically. Machine learning could perhaps be used here to determine which lemmas are harmful.

### 2.3.3 Proof Minimisation

Sometimes Metis is simply not powerful enough to prove a theorem that has already been proved by a more powerful system, despite being given a small list of axioms. ATPs frequently use many more axioms than are necessary. Sledgehammer's minimisation tool takes a set of axioms returned by a given ATP and repeatedly calls the same ATP with subsets of those axioms in order to find a minimal set. Reducing the number of axioms improves Metis's speed and success rate, while also removing superfluous clutter from the proof scripts.

ATPs themselves could return proofs using a minimum of axioms or, alternatively, proofs of a minimum length. Vampire's well-known limited resource strategy [30], although designed to cope with limited processor time, could probably be modified to minimise proofs efficiently.

## 2.4 Parallelism

Parallelism was another design objective, both to exploit the abundance of cheap processing power and so that users would not have to wait. Sledgehammer was intended to run in the background; Isabelle would continue to respond to commands, and users could keep working. This idea has turned out to be somewhat misconceived: thinking is difficult, and users typically wait for Sledgehammer to return, hoping to get a proof for free. We hope that eventually Sledgehammer will be configured to run spontaneously, without even the need for a mouse click. Then users will simply work and occasionally be delighted to have solutions displayed for them. Such a configuration would require a machine with enough processing power to support several ATP executions without becoming sluggish. An agent-based implementation of similar ideas, using a blackboard architecture, has for some time been part of the ΩMEGA system [6, 33].

The parallel invocation of different theorem provers is invaluable. Böhme and Nipkow [8] have demonstrated that running three different theorem provers (E, SPASS, and Vampire) for five seconds solves as many problems as running the best theorem prover (Vampire) for a full two minutes. It would

be better to employ even more theorem provers. We have undertaken informal, unpublished experiments involving many other systems.

- Gandalf [35] shows great potential, but unfortunately it does not output useful proofs; one cannot easily identify which axioms have taken part in the proof. A simple source code modification to improve the legibility of proofs would allow Gandalf to make useful contributions. Unfortunately, we were unable to identify the necessary changes. Gandalf has been found to be unsound,[2] but a small percentage of incorrect (and hence unreconstructable) proofs would be tolerable.
- SInE, the Sumo Inference Engine [11], is a wrapper around E that is designed to cope with large axiom bases. We pass it more facts than can be handled by the other ATPs, and it sometimes surprises us with original proofs. In the current experimental setup, it is invoked remotely via SystemOnTPTP [34] in parallel with Vampire.
- People sometimes suggest that we include Prover9 [15]. In our experiments, Prover9 performed poorly on the large problems generated by Sledgehammer. It could be effective in conjunction with an advanced and selective relevance filter.
- We could also run multiple instances of a theorem prover with different heuristics. This is not necessary with Vampire, which attempts a variety of heuristics in separate time slices. It could be particularly effective with E, but designing suitable heuristics requires highly specialised skills.

## 3 Evaluation

In their "Judgement Day" study, Böhme and Nipkow [8] evaluated Sledgehammer with E, SPASS, and Vampire on 1240 provable proof goals arising in seven representative Isabelle theories:

| | |
|---|---|
| *Arrow* | Arrow's impossibility theorem |
| *NS* | Needham–Schroeder shared-key protocol |
| *Hoare* | Completeness of Hoare logic with procedures |
| *Jinja* | Type soundness of a subset of Java |
| *SN* | Strong normalisation of the typed $\lambda$-calculus with de Bruijn indices |
| *FTA* | Fundamental theorem of algebra |
| *FFT* | Fast Fourier transform |

Sledgehammer has been developed further since they ran their experiments. In particular, it now communicates with ATPs using full first-order logic instead of clause form, adds SInE to the collection of ATPs, and employs the latest versions of SPASS, Vampire, and Metis. To account for these changes, we ran the Judgement Day benchmark suite on the same hardware as Böhme and Nipkow but with the latest version of Sledgehammer and of the Isabelle theories.

When running the four ATPs in parallel for 120 seconds, followed by Metis with a 30-second time limit, Sledgehammer now solves 52% of the goals (compared with 48% in Böhme and Nipkow). The table below gives the success rates for each ATP and theory.

| | *Arrow* | *NS* | *Hoare* | *Jinja* | *SN* | *FTA* | *FFT* | Avg. |
|---|---|---|---|---|---|---|---|---|
| SInE 0.4 | 18% | 22% | 43% | 31% | 61% | 53% | 17% | 40% |
| E 1.0 | 19% | 39% | 45% | 33% | 66% | 57% | 17% | 44% |
| SPASS 3.7 | 30% | 35% | 43% | 32% | 59% | 58% | 17% | 44% |
| Vampire 1.0 | 36% | 40% | 50% | 35% | 63% | 60% | 17% | 47% |
| Together | 43% | 45% | 54% | 41% | 68% | 65% | 26% | 52% |

[2] See `http://www.cs.miami.edu/~tptp/TPTP/BustedAsUnsound.html`.

About one third of the goals are "trivial" in the sense that they can be solved by standard Isabelle tactics invoked with no arguments. For these, the overall success rate is 84%. Regrettably, for the nontrivial goals, which users are especially keen on seeing solved by Sledgehammer, the overall success rate is much lower at 38%.[3]

Proof reconstruction using Metis loses about 10% of ATP proofs, partly because some of the ATP proofs are unsound in a typed setting, but mostly because Metis times out. Proof minimisation reduces the required number of facts by about one third, thereby helping 20% of the failed Metis proofs to succeed. It would be interesting to measure the effect of reconstructing Isar proof texts when the one-line Metis calls time out; regrettably, we currently lack the infrastructure to do so.

# 4 Sledgehammer and Teaching

Sledgehammer was not designed specifically as an aid to novices. Experienced users have come to rely on it. But Sledgehammer seems to offer the greatest benefits to the least experienced users. It has certainly transformed the way Isabelle is taught. There are two reasons for this:

- Because it identifies relevant facts, users no longer need to memorise lemma libraries.
- Because it works in harmony with Isar structured proofs, users no longer need to learn many low-level tactics.

Demonstrations of interactive theorem provers necessarily involve deception. The implementers naturally want to show off their system in the best possible light, so they present examples that look more difficult than they really are. Typically they define some recursive functions and prove properties using obvious inductions followed by some sort of automatic tactic. The audience will be duly impressed, and some among them will decide to adopt that tool as the basis for their Ph.D. research. Too late, they encounter the crucial issue:

*What do I do when the automatic tactics fail?*

Typically, the answer is that one must write incomprehensible scripts that invoke a plethora of obscure commands. These generally include tactics to manipulate the set of assumptions in natural deduction or a sequent calculus. There may be tactics to transform an assumption by applying a rewrite rule or theorem, to create a case split from an assumption, or to substitute user-supplied terms into theorems.

In Isabelle, the simple combination of structured proofs and Sledgehammer takes the user surprisingly far. This is not the place to give a detailed tutorial on Isar structured proofs [20, 38]. In brief, they support natural deduction through local scopes that can introduce assumptions (using the keyword **assume**) as well as local variables and definitions. Moreover, while traditional tactic scripts contain only commands, a structured proof explicitly states the assumptions and goals. When proving a proposition, users can state intermediate properties that they believe to be helpful. If they understand the problem well enough to propose some intermediate properties, then all they have to do is state a progression of properties in small enough steps for Sledgehammer to be able to prove each one.

In the example below, taken a measure theory development, two intermediate facts (introduced by **hence**) assist in the proof of the conclusion (introduced by **thus**):

**proof** –
  **assume** *x*: "*x* ∈ *lambda_system M f* "

[3] Böhme and Nipkow reported a success rate of 34% for nontrivial goals. Their result cannot be directly compared with ours because of methodological differences: to establish triviality, they relied on a syntactic criterion and ignored two of the theories, as opposed to actually running the Isabelle tactics on all proof goals.

**hence** "$x \subseteq$ *space M*"
**by** (*metis sets_into_space lambda_system_sets*)
**hence** "*space M* $-$ (*space M* $-$ *x*) = *x*"
**by** (*metis double_diff equalityE*)
**thus** "*space M* $-$ *x* $\in$ *lambda_system M f*" **using** *x*
**by** (*force simp add*: *lambda_system_def*)
**qed**

Each of the intermediate facts is proved by a call to Metis that was generated using Sledgehammer. While the example features a linear progression of facts, Isar proofs can also be nested to any depth.

Isar also supports calculational reasoning [4]. A chain of reasoning steps, connected by familiar relations such as $=$, $\leq$, and $<$, can be written with separate proofs for each step of the calculation. Once again, if the user can see the intermediate stages of the transformation, then the proof of each step can easily be found. The example below illustrates this type of reasoning:

**proof** –
⋮
**have** "$f(u \cap (x \cap y)) + f(u - x \cap y) = (f(u \cap (x \cap y)) + f(u \cap y - x)) + f(u - y)$"
**by** (*metis class_semiring.add_a ey*)
**also have** "$\ldots = (f((u \cap y) \cap x) + f(u \cap y - x)) + f(u - y)$"
**by** (*metis Int_commute Int_left_commute*)
**also have** "$\ldots = f(u \cap y) + f(u - y)$" **using** *fx Int y u*
**by** *auto*
**also have** "$\ldots = f\,u$"
**by** (*metis fy u*)
**finally show** "$f(u \cap (x \cap y)) + f(u - x \cap y) = f\ u$" **.**
**qed**

Top down proof development is greatly assisted by a trivial Isar feature: the ability to omit proofs. Where a proof is required, the user may simply insert the word **sorry**. Isabelle then regards the theorem as proved.[4] The user can then check that the newly introduced proposition indeed suffices to prove the next proposition in the development. A difficult proof can develop as a series of propositions, each initially "proved" using **sorry** but eventually using either Sledgehammer, an automatic tactic, or a nested proof development of the same form. Progress in such a proof can be measured in terms of the difficulty of the propositions that lack real proofs. Although we can never be certain that a proof development can be completed until the very end, the ability to write **sorry** in place of a proof reduces the risk of discovering that a lemma is useless only after spending weeks proving it.

In January 2010, as part of its new M.Phil. programme, the University of Cambridge offered a lecture course on Isabelle [22]. The course materials included almost no information about the low-level tactics that had been the mainstay of Isabelle proofs for nearly 20 years. Only two of the 12 lectures were devoted to Isar structured proofs, and they took a novel approach: rather than proceeding methodically through the Isar fundamentals, the lectures presented the outer skeleton of a proof, with crucial sections replaced by **sorry**. They described the idea of trying to eliminate each **sorry** using either Sledgehammer or some automatic tactic. Practical work submitted by the students later demonstrated that several of them had learnt how to write complex, well-structured proofs. We were happy to reassure them that submitting work generated largely by Sledgehammer was by no means cheating!

[4] The existence of **sorry** does not compromise Isabelle's soundness, because it is only permitted during interactive sessions. A theory file containing an occurrence of **sorry** may not be imported by another theory.

The first author has taught Isabelle on a number of occasions, starting in the mid-1990s. Sledgehammer is obviously not the only thing to have changed in that period. The introduction of Isar, continual improvements to the automated reasoners and counterexample generators, and 15 years of Moore's Law have transformed the user experience. Interactive theorem proving has never been practical because it required far too much effort, even from highly specialised experts. For the first time, we can envisage the day when interactive theorem proving becomes straightforward enough to be adopted on a large scale.

## 5 Conclusions

Sledgehammer has been available for over three years, and in that time it has become an essential part of the Isabelle user's workflow. It is the only interface between an interactive theorem prover and automatic ones to achieve such popularity with users. It has transformed the way beginners perceive Isabelle.

Sledgehammer has a number of limitations which we hope to address. The relevance filter is primitive, but an improved one will have to be part of Sledgehammer itself as long as unsound translations are used. Unsound translations can be used safely because Sledgehammer does not trust the proofs that it receives from ATPs but merely uses them as hints to generate Isabelle proof scripts; proofs that violate Isabelle's typing rules are eliminated at this stage.

Sledgehammer's performance on higher-order problems is unimpressive, and given the inherent difficulty of performing higher-order reasoning using first-order theorem provers, the way forward is to integrate Sledgehammer with higher-order automatic theorem provers, such as LEO-II [5] and Satallax [3]. Proof reconstruction would benefit from new ideas, especially so that it can deliver natural, intuitive proofs. Finally, we hope to generalise Sledgehammer so that it can also cope with Isabelle/ZF, the Zermelo–Fraenkel set-theory instantiation of Isabelle [24, 25].

## Acknowledgements

Christoph Benzmüller commented on the version of this paper presented at PAAR-2010. Mark Summerfield suggested several textual improvements to a later draft.

The Cambridge team that developed Sledgehammer included Jia Meng, Claire Quigley, and Kong Woei Susanto. Sledgehammer has since been refined, improved, and tested by the Isabelle team in Munich—notably Sascha Böhme, Fabian Immler, Philipp Meyer, Tobias Nipkow, and Markus Wenzel. Throughout the development, we benefited from Joe Hurd's expert help with Metis.

The research was supported by the Engineering and Physical Sciences Research Council (*Automation for Interactive Proof*, grant number GR/S57198/01) and also partly by the Deutsche Forschungsgemeinschaft (*Quis Custodiet*, grant number Ni 491/11-1).

## References

[1] Wolfgang Ahrendt, Bernhard Beckert, Reiner Hähnle, Wolfram Menzel, Wolfgang Reif, Gerhard Schellhorn, and Peter H. Schmitt. Integrating automated and interactive theorem proving. In Wolfgang Bibel and Peter H. Schmitt, editors, *Automated Deduction—A Basis for Applications*, volume II. Systems and Implementation Techniques, pages 97–116. Kluwer Academic Publishers, 1998.

[2] Peter B. Andrews. *An Introduction to Mathematical Logic and Type Theory: To Truth Through Proof (2nd Ed.)*. Springer, 2002. Applied Logic 27.

[3] Julian Backes and Chad E. Brown. Analytic tableaux for higher-order logic with choice. In Jürgen Giesl and Reiner Hähnle, editors, *Automated Reasoning—5th International Joint Conference, IJCAR 2010*, LNAI 6173, pages 76–90. Springer, 2010.

[4] Gertrud Bauer and Markus Wenzel. Calculational reasoning revisited (an Isabelle/Isar experience). In Richard J. Boulton and Paul B. Jackson, editors, *Theorem Proving in Higher Order Logics: TPHOLs 2001*, LNCS 2152, pages 75–90. Springer, 2001. Online at `http://link.springer.de/link/service/series/0558/tocs/t2152.htm`.

[5] Christoph Benzmüller, Lawrence C. Paulson, Frank Theiss, and Arnaud Fietzke. LEO-II—A cooperative automatic theorem prover for higher-order logic. In Alessandro Armando, Peter Baumgartner, and Gilles Dowek, editors, *Automated Reasoning—4th International Joint Conference, IJCAR 2008*, LNAI 5195, pages 162–170. Springer, 2008.

[6] Christoph Benzmüller and Volker Sorge. OANTS—An open approach at combining interactive and automated theorem proving. In Manfred Kerber and Michael Kohlhase, editors, *Symbolic Computation and Automated Reasoning*, pages 81–97. A. K. Peters, 2000.

[7] Marc Bezem, Dimitri Hendriks, and Hans de Nivelle. Automatic proof construction in type theory using resolution. *Journal of Automated Reasoning*, 29(3–4):253–275, 2002.

[8] Sascha Böhme and Tobias Nipkow. Sledgehammer: Judgement day. In Jürgen Giesl and Reiner Hähnle, editors, *Automated Reasoning (IJCAR 2010)*, LNCS 6173, pages 107–121. Springer, 2010.

[9] Alonzo Church. A formulation of the simple theory of types. *Journal of Symbolic Logic*, 5:56–58, 1940.

[10] M. J. C. Gordon and T. F. Melham. *Introduction to HOL: A Theorem Proving Environment for Higher Order Logic*. Cambridge University Press, 1993.

[11] Kryštof Hoder. SInE (Sumo Inference Engine). `http://www.cs.man.ac.uk/~hoderk/sine/`.

[12] Joe Hurd. Integrating Gandalf and HOL. In Yves Bertot, Gilles Dowek, André Hirschowitz, Christine Paulin, and Laurent Théry, editors, *Theorem Proving in Higher Order Logics: TPHOLs '99*, LNCS 1690, pages 311–321. Springer, 1999.

[13] Joe Hurd. First-order proof tactics in higher-order logic theorem provers. In Myla Archer, Ben Di Vito, and César Muñoz, editors, *Design and Application of Strategies/Tactics in Higher Order Logics*, number NASA/CP-2003-212448 in NASA Technical Reports, pages 56–68, September 2003.

[14] David McAllester. Ontic: A knowledge representation system for mathematics. In Ewing Lusk and Ross Overbeek, editors, *9th International Conference on Automated Deduction*, LNCS 310, pages 742–743. Springer, 1988.

[15] William McCune. Prover9 and Mace4. `http://www.cs.unm.edu/~mccune/prover9/`.

[16] Andreas Meier. TRAMP: Transformation of machine-found proofs into natural deduction proofs at the assertion level (system description). In David McAllester, editor, *Automated Deduction—CADE-17 International Conference*, LNAI 1831, pages 460–464. Springer, 2000.

[17] Jia Meng and Lawrence C. Paulson. Translating higher-order clauses to first-order clauses. *Journal of Automated Reasoning*, 40(1):35–60, 2008.

[18] Jia Meng and Lawrence C. Paulson. Lightweight relevance filtering for machine-generated resolution problems. *Journal of Applied Logic*, 7(1):41–57, 2009.

[19] Jia Meng, Claire Quigley, and Lawrence C. Paulson. Automation for interactive proof: First prototype. *Information and Computation*, 204(10):1575–1596, 2006.

[20] Tobias Nipkow. A tutorial introduction to structured Isar proofs. `http://isabelle.in.tum.de/dist/Isabelle/doc/isar-overview.pdf`.

[21] Tobias Nipkow, Lawrence C. Paulson, and Markus Wenzel. *Isabelle/HOL: A Proof Assistant for Higher-Order Logic*. Springer, 2002. LNCS 2283.

[22] Lawrence C. Paulson. Interactive formal verification. `http://www.cl.cam.ac.uk/teaching/0910/L21/`. Lecture course materials.

[23] Lawrence C. Paulson. The foundation of a generic theorem prover. *Journal of Automated Reasoning*, 5(3):363–397, 1989.

[24] Lawrence C. Paulson. Set theory for verification: I. From foundations to functions. *Journal of Automated Reasoning*, 11(3):353–389, 1993.

[25] Lawrence C. Paulson. Set theory for verification: II. Induction and recursion. *Journal of Automated Reasoning*, 15(2):167–215, 1995.

[26] Lawrence C. Paulson. Tool support for logics of programs. In Manfred Broy, editor, *Mathematical Methods in Program Development: Summer School Marktoberdorf 1996*, NATO ASI Series F, pages 461–498. Springer, 1997.

[27] Lawrence C. Paulson. Three years of experience with Sledgehammer, a practical link between automatic and interactive theorem provers. In Boris Konev, Renate Schmidt, and Stephan Schulz, editors, *PAAR-2010: Workshop on Practical Aspects of Automated Reasoning*, 2010.

[28] Lawrence C. Paulson and Kong Woei Susanto. Source-level proof reconstruction for interactive theorem proving. In Klaus Schneider and Jens Brandt, editors, *Theorem Proving in Higher Order Logics: TPHOLs 2007*, LNCS 4732, pages 232–245. Springer, 2007.

[29] Alexander Riazanov and Andrei Voronkov. Vampire 1.1 (system description). In Rajeev Goré, Alexander Leitsch and Tobias Nipkow, editors, *Automated Reasoning—First International Joint Conference, IJCAR 2001*, LNAI 2083, pages 376–380. Springer, 2001.

[30] Alexandre Riazanov and Andrei Voronkov. Limited resource strategy in resolution theorem proving. *Journal of Symbolic Computation*, 36(1–2):101–115, 2003.

[31] Alan Robinson and Andrei Voronkov, editors. *Handbook of Automated Reasoning*. Elsevier Science, 2001.

[32] Stephan Schulz. System description: E 0.81. In David Basin and Michaël Rusinowitch, editors, *Automated Reasoning—Second International Joint Conference, IJCAR 2004*, LNAI 3097, pages 223–228. Springer, 2004.

[33] Jörg Siekmann, Christoph Benzmüller, Armin Fiedler, Andreas Meier, Immanuel Normann and Martin Pollet. Proof development with ΩMEGA: The irrationality of $\sqrt{2}$. In Fairouz Kamareddine, editor, *Thirty Five Years of Automating Mathematics*, pages 271–314. Kluwer Academic Publishers, 2003.

[34] Geoff Sutcliffe. System description: SystemOnTPTP. In David McAllester, editor, *Automated Deduction—CADE-17 International Conference*, LNAI 1831, pages 406–410. Springer, 2000.

[35] Tanel Tammet. Gandalf. *Journal of Automated Reasoning*, 18(2):199–204, 1997.

[36] Josef Urban. MaLARea: A metasystem for automated reasoning in large theories. In Geoff Sutcliffe, Josef Urban, and Stephan Schulz, editors, *ESARLT 2007: Empirically Successful Automated Reasoning in Large Theories*, volume 257 of *CEUR Workshop Proceedings*, 2007.

[37] Christoph Weidenbach. Combining superposition, sorts and splitting. In Robinson and Voronkov [31], chapter 27, pages 1965–2013.

[38] Makarius Wenzel. Isabelle/Isar—A generic framework for human-readable proof documents. In Roman Matuszewski and Anna Zalewska, editors, *From Insight to Proof—Festschrift in Honour of Andrzej Trybulec*. University of Białystok, 2007. Studies in Logic, Grammar, and Rhetoric 10(23).

[39] Markus Wenzel. Type classes and overloading in higher-order logic. In Elsa L. Gunter and Amy Felty, editors, *Theorem Proving in Higher Order Logics: TPHOLs '97*, LNCS 1275, pages 307–322. Springer, 1997.

[40] Jürgen Zimmer, Andreas Meier, Geoff Sutcliffe, and Yuan Zhan. Integrated proof transformation services. In Christoph Benzmüller and Wolfgang Windsteiger, editors, *Workshop on Computer-Supported Mathematical Theory Development, IJCAR 2004*, ENTCS, 2004.

# Exploring Steinitz-Rademacher polyhedra: a challenge for automated reasoning tools

Jesse Alama*
Center for Artificial Intelligence
Department of Computer Science, Faculty of Science and Technology
New University of Lisbon, Portugal
j.alama@fct.unl.pt

**Abstract**

This note reports on some experiments, using a handful of standard automated reasoning tools, for exploring Steinitz-Rademacher polyhedra, which are models of a certain first-order theory of incidence structures. This theory and its models, even simple ones, presents significant, geometrically fascinating challenges for automated reasoning tools are.

## 1 Introduction

This note reports on some experiments, using a handful of standard automated reasoning tools, for exploring a certain first-order theory of three-dimensional polyhedra. Polyhedra are understood here as combinatorial objects, rather than as, say, certain kinds of structures in $\mathbf{R}^3$.

Specifically, the polyhedra considered here are Steinitz-Rademacher polyhedra (they will be defined in Section 2). As first-order structures, these polyhedra are directed graphs with three sorts—vertices, edges, and faces—satisfying some intuitive geometric principles shared by "everyday" three-dimensional polyhedra.

Restricting ourselves to first-order logic makes it possible to take advantage of automated reasoning tools that work well for FOL, but our restriction comes with a price: many interesting features of graphs, such as connectivity or the property of satisfying Euler's formula, for example, cannot be expressed in FOL.[1] Nonetheless, it is not clear that FOL's lack of expressive power precludes the possibility of learning something about polyhedra. We believe that the preliminary results discussed here do have interesting mathematical content.

An investigation of polyhedra with automated reasoners is valuable for two domains. First, the investigation is valuable for mathematics, because automated reasoners offer the possibility, in certain contexts, of a more objective investigation than one carried out by entirely by humans, who are prone to make subtle flaws when reasoning about space. Second, the investigation is valuable for automated reasoning, because working with polyhedra—even small ones—naturally leads to challenging problems, as we shall see.

There do not appear to be many explorations of polyhedra using automated reasoning tools. Much has been done on enumerating polyhedra (or related combinatorial structures, such as planar graphs) using mathematical rather than logical techniques; one such system is the highly efficient plantri [1]). Within the realm of automated reasoning, L. Schewe has used SAT solvers to investigate realizability of abstract simplicial complexes [6].

*Partially supported by the ESF research project *Dialogical Foundations of Semantics* within the ESF Eurocores program *LogICCC* (funded by the Portuguese Science Foundation, FCT LogICCC/0001/2007).

[1] FOL's failure to express these and other properties of graphs holds even when one restricts attention to finite structures; such results can be shown using Ehrenfeucht-Fraïssé games [4].

## 2 Steinitz-Rademacher polyhedra

E. Steinitz asked [7]: when can a combinatorially given polyhedron—a collection of abstract vertices, edges, and faces, with an incidence relation among these polytopes—be realized as a three-dimensional convex polyhedron in $\mathbf{R}^3$?[2] As part of his initial investigation Steinitz formulated a basic theory of combinatorial polyhedra that forms the basis for our investigation as well.

First, we specify an unsorted first-order signature:

**Definition 1.** *Let $\pi$ be the first-order signature (with equality) containing three unary predicates $V$ (for "vertex"), $E$ (for "edge"), and $F$ (for "face"), and one binary relation $I$ (for incidence).*

The signature $\pi$ provides a rudimentary language for talking about three-dimensional polyhedra. Alternatively, one can view $\pi$-structures simply as directed graphs whose nodes can be painted with one of three "colors" $V$, $E$, and $F$.

**Definition 2.** *The theory* SR *of Steinitz-Rademacher polyhedra consists of the statements:*

- *there are vertices, edges, and faces;*
- *every element is a vertex, edge, or a face;*
- *$I$ is symmetric;*
- *no two vertices are incident, and the same goes for edges and faces;*
- *if $V(v)$, $E(e)$, $F(f)$, $I(v,e)$ and $I(e,f)$, then $I(v,f)$;*
- *every edge is incident with exactly two vertices;*
- *every edge is incident with exactly two faces;*
- *$V(v)$, $F(f)$ and $I(v,f)$ imply that there are exactly two edges incident with both $v$ and $f$; and*
- *every vertex and every face is incident with at least one other element.*

These conditions are expressible as first-order $\pi$-sentences.

**Definition 3.** *A **Steinitz-Rademacher polyhedron** (or **SR-polyhedron**) is a model of the theory* SR.

Some questions that we would like to address about SR-polyhedra, in this note, are:

1. SR is consistent (just think of, say, a tetrahedron). What is the smallest model?[3]
2. For which natural numbers $k$ is SR $k$-categorical?[4]
3. How hard is it to "recover" (that is, produce as models of SR) well-known polyhedra (e.g., the platonic solids) as models of SR?
4. Can we discover unusual or unexpected SR-polyhedra?

The rest of the paper takes up these questions.

Question 1 is also one of the first questions raised by Steinitz and Rademacher [7].

[2] The answer, known as Steinitz's theorem [3], says that a directed graph $g$ is isomorphic to the 1-skeleton of a real convex three-dimensional polyhedron iff $g$ is planar and three-connected.

[3] There is no largest finite SR-polyhedron. Consider, for example, the sequence $\langle P_n \mid n \geq 3 \rangle$ of pyramids, each $P_n$ characterized by an $n$-gon $B_n$ for its base and a single point "above" the base incident with each of the vertices of $B_n$. Each $P_n$ is evidently an SR-polyhedron has cardinality $(n+1)+2n+(n+1) = 4n+2$.

[4] SR is not $\lambda$-categorical for any infinite cardinal $\lambda$. Consider, for example, the "tessellation" $M^3_\lambda$ having $\lambda$ vertices, $\lambda$ edges, and $\lambda$ faces, each of which is a triangle that meets three other triangles along its three edges, *ad $\lambda$-infinitum*; and $M^4_\lambda$, which is a "tessellation" like $M^3_\lambda$ except that each face of $M^4_\lambda$ has four edges rather than three. In $M^3_\lambda$ the $\pi$-sentence "every face is a triangle" holds, but by construction it fails in $M^4_\lambda$. Or, continuing the discussion of the previous footnote, consider a tetrahedron and a cube, each of whose edges and faces contains $\lambda$-many vertices (but each having their usual finite number of edges and faces).

**Theorem 4.** *There is a Steinitz-Rademacher polyhedron of cardinality 6, but none of smaller cardinality.*

*Proof.* This result is readily confirmed with the help of a first-order model generation tool (e.g., mace4 [5] or paradox/equinox [2]). A refutational theorem prover can then show that SR, extended with an axiom saying that there are at most five elements, is inconsistent. □

Figure 1 illustrates this smallest SR-polyhedron, $M_6$, with two vertices, two edges, and two faces; the two edges are the upper and lower semicircular arcs, and the two faces are the inside and outside of the circle. $M_6$ has the curious feature that every vertex is incident with every edge and with every face.

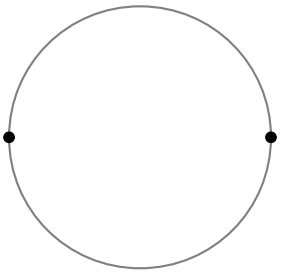


Figure 1: $M_6$: the smallest SR-polyhedron, with six elements: two vertices, two edges, two faces

Depending on one's view about polyhedra, $M_6$ might be a positive solution to Question 4; see Section 3.

**Theorem 5.** *Up to isomorphism, $M_6$ is the only SR-polyhedron of cardinality six.*

*Proof.* Each of the 28 triples of natural numbers $(N_0, N_1, N_2)$ that sum to 6 gives rise to an extension $\mathsf{SR}_{N_0,N_1,N_2}$ of SR obtained by adding axioms saying that there are exactly $N_0$ vertices, $N_1$ edges, and $N_2$ faces. All but one of these 28 theories—namely, $\mathsf{SR}_{2,2,2}$—are inconsistent; this can be shown by applying a standard refutational theorem prover to the 27 theories different from $\mathsf{SR}_{2,2,2}$.

To show that $M_6$ is, up to isomorphism, the only SR-polyhedron with exactly 2 vertices, 2 edges, and 2 faces, consider the extension of SR by the formula

$$\varphi := \exists x_0, \ldots, x_5 \left[ \begin{array}{c} V(x_0) \wedge V(x_1) \wedge x_0 \neq x_1 \wedge \forall x (V(x) \rightarrow (x = x_0 \vee x = x_1)) \\ \wedge \\ E(x_2) \wedge E(x_3) \wedge x_2 \neq x_3 \wedge \forall x (E(x) \rightarrow (x = x_2 \vee x = x_3)) \\ \wedge \\ F(x_4) \wedge F(x_5) \wedge x_4 \neq x_5 \wedge \forall x (F(x) \rightarrow (x = x_4 \vee x = x_5)) \\ \wedge \\ \neg (I(x_0, x_2) \wedge I(x_1, x_2) \wedge I(x_0, x_3) \wedge I(x_1, x_3) \wedge I(x_2, x_4) \wedge I(x_3, x_4) \wedge I(x_2, x_5) \wedge I(x_3, x_5)) \end{array} \right]$$

The first three bundles of conjunctions in the matrix of $\varphi$ express the cardinality of the sets of vertices, edges, and faces. The final conjunction expresses the essential incidence relations holding among the elements of $M_6$, when one labels the vertices $x_0$ and $x_1$, the edges $x_2$ and $x_3$, and the faces $x_4$ and $x_5$; it is negated because we are trying to find a model that is *unlike* $M_6$. One then shows, with, e.g., mace4, that $\mathsf{SR} \cup \{\varphi\}$ is unsatisfiable. □

**Theorem 6.** *There are at least two SR-polyhedron of cardinality 8.*

One such cardinality 8 SR-polyhedron is depicted in Figure 2. The two vertices are clear; the three edges are the upper and lower semicircles, plus the straight line segment joining the two vertices. The three faces are: the exterior of the circle, incident with the upper and lower semicircular arcs; and the two semicircles, each incident with one of the semicircular arcs and both incident with the straight line

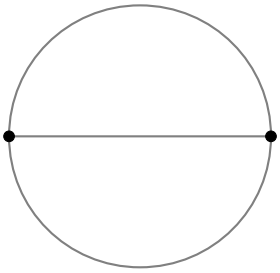


Figure 2: An SR-polyhedron with eight elements: two vertices, three edges, three faces.

segment. Another is its dual, $M_8^d$, which is obtained by exchanging the vertices and faces of $M_8$ (notice that SR is invariant under this exchange).

Returning to Question 4, as with, $M_6$, the SR-polyhedron $M_8$ and its dual $M_8^d$ could be regarded as an unusual or unexpected model of SR. Although SR does admit curious "polyhedra", it should be clear that one can take nearly any familiar polyhedron, such as the platonic solids, as SR-polyhedra.

| **Solid** | **Num. Vertices** | **Num. Edges** | **Num. Faces** | **Cardinality** |
|---|---|---|---|---|
| Tetrahedron | 4 | 6 | 4 | 14 |
| Cube | 8 | 12 | 6 | 26 |
| Octahedron | 6 | 12 | 8 | 26 |
| Dodecahedron | 20 | 30 | 12 | 62 |
| Icosahedron | 12 | 30 | 20 | 62 |

Table 1: Cardinal numbers for the platonic solids

Notice that, since they are duals, the cube and the octahedron, as well as the dodecahedron and the icosahedron, show that SR is neither 26- nor 62-categorical. We now have the modest beginnings of an answer to Question 2: for $k = 6$ we have that SR is $k$-categorical, and for $k = 8$, 26, and 62 we know that SR is not $k$-categorical, by duality.

Although the tetrahedron can be recovered as an SR-polyhedron (by, e.g., paradox), the remaining platonic solids seem to lie tantalizingly beyond the scope of current automated reasoning tools: a very large amount of time indeed seems to be required to automatically generate these solids. It would be interesting to determine whether the cube and its dual the icosahedron are the only SR-polyhedra of cardinality 26.

For lack of space the investigation has to be cut short here.

# 3 Future work

The range of Steinitz-Rademacher polyhedra is arguably too large: if the tetrahedron is the "simplest" three-dimensional polyhedron, then the six-element SR-polyhedron $M_6$ in Figure 2 and other SR-polyhedra of cardinality less than that of the tetrahedron, such as $M_8$ in Figure 2, show that SR alone lacks sufficient geometric content and needs to be extended by principles that rule out such models. Extensionality is a natural candidate. $M_6$, for example, has two distinct vertices that share the same edges and faces, two distinct edges that share the same two vertices and faces, and two distinct faces that share the same vertices and edges. The eight-element model $M_8$ is similar. Intuitively, the incidence relation between polytopes is extensional: if two polytopes $p$ and $q$ are incident with the same set of polytopes, then $p = q$.

Adding extensionality to SR yields a new theory $\mathsf{SR}_{\text{ext}}$ of extensional SR-polyhedra. paradox can recover the tetrahedron as the smallest model of $\mathsf{SR}_{\text{ext}}$, thereby answering Question 1 and part of Question 3 for the new theory. However, extensionality raises a high computational hurdle. Concerning Questions 2 and 3, the search for models of $\mathsf{SR}_{\text{ext}}$ has so far not been unsuccessful beyond the aforementioned recovery of the tetrahedron. Question 4, about $k$-categoricity, is an even greater challenge for $\mathsf{SR}_{\text{ext}}$ than it was for SR and is more intriguing because its models are more geometrically intuitive.

The signature $\pi$ of SR-polyhedra is unsorted: $V$, $E$, $F$, and $I$ are relations that could hold for arbitrary elements in a $\pi$-structure. It is likely that tools for sorted FOL, such as SPASS [8], would be more effective than the unsorted tools used so far. Constraint techniques for model generation, such as those behind sem [9], ought also to be evaluated.

# 4 Conclusion

Combinatorial polyhedra abstract away from positions in space and regard polyhedra as incidence structures. Steinitz-Rademacher polyhedra are one such kind of polyhedra axiomatized by an intuitive first-order theory. We have proposed a handful of basic questions about these polyhedra and shown that automated reasoners can tackle some of them with verve, though others remain only partially answered. Even small Steinitz-Rademacher polyhedra present significant challenges for automated reasoners; dealing with larger ones will likely require new techniques. We thus urge combinatorial polyhedra as a tantalizingly fertile source of challenging automated reasoning problems.

# References

[1] G. Brinkmann and B. D. McKay. Fast generation of planar graphs. *MATCH Comm. in Comp. Chem.*, 58:323–357, 2003.

[2] K. Claessen and N. Sorensson. New Techniques that Improve MACE-style Finite Model Finding. In P. Baumgartner and C. Fermueller, editors, *Proc. of the CADE-19 Workshop: Model Computation: Principles, Algorithms, Applications*, 2003.

[3] B. Grünbaum. Graphs of polyhedra; polyhedra as graphs. *Discrete Mathematics*, 307:445–463, 2007.

[4] L. Libkin. *Elements of Finite Model Theory*. Texts in Theoretical Computer Science. Springer, 2004.

[5] W. McCune. Prover9 and Mace4. `http://www.cs.unm.edu/~mccune/prover9/`.

[6] L. Schewe. Generation of oriented matroids using satisfiability solvers. In A. Iglesias and N. Takayama, editors, *Proc. of the 2nd Int. Conf. on Math. Soft.*, volume 4151 of *Lecture Notes in Computer Science*. Springer, 2006.

[7] E. Steinitz and H. Rademacher. *Vorlesungen über die Theorie der Polyeder unter Einschluss der Elemente der Topologie*. Springer, 1976. Reprint of the original 1934 edition.

[8] C. Weidenbach, D. Dimova, A. Fietzke, R. Kumar, M. Suda, and P. Wischnewski. SPASS version 3.5. In R. Schmidt, editor, *Automated Deduction: CADE 22*, volume 5663 of *Lecture Notes in Computer Science*, pages 140–145. Springer, 2009.

[9] J. Zhang and H. Zhang. SEM: A system for enumerating models. In C. P. Mellish, editor, *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, pages 298–303, 1995.

# Tableau Calculus for Dummett Logic Based on Present and Next State of Knowledge

Guido Fiorino
Dipartimento di Metodi Quantitativi per le Scienze Economiche ed Aziendali,
Università di Milano-Bicocca, Piazza dell'Ateneo Nuovo, 1, 20126 Milano, Italy.
guido.fiorino@unimib.it

**Abstract**

In this paper we use the Kripke semantics characterization of Dummett logic to introduce a new way of handling non-forced formulas in tableau proof systems. We pursue the aim of reducing the search space by strictly increasing the number of forced propositional variables after the application of non-invertible rules. The focus of the paper is on a new tableau system for Dummett logic, for which we have an implementation. The ideas presented can be extended to intuitionistic logic as well.

## 1 Introduction

In this paper we present a tableau calculus and theorem prover for propositional Dummett logic. By exploiting the linearly ordered Kripke semantics of Dummett logic, we devise a tableau calculus working on two semantical levels: the present and the next state of knowledge. In this way we can guarantee that as the construction of the tableau proceeds, moving from a state of knowledge to another the known information strictly increases. Moreover, the calculus can be equipped with specialized rules to reduce the branching. As a result, the decision procedure is speeded up. In Section 6 we discuss how to apply our ideas to a tableau calculus exploiting information about the next state of knowledge in the case of Intuitionistic deduction.

Dummett logic has been extensively investigated both by people working in computer science and in logic. The history of this logic starts with Gödel, who studied the family of logics semantically characterizable by a sequence of $n$-valued ($n > 2$) matrices ([12]). In paper [7] Dummett studied the logic semantically characterized by an infinite valued matrix which is included in the family of logics studied by Gödel and proved that such a logic is axiomatizable by adding to any Hilbert system for propositional intuitionistic logic the axiom scheme $(p \rightarrow q) \vee (q \rightarrow p)$. Moreover, it is well-known that such a logic is semantically characterizable by linearly ordered Kripke models. Dummett logic has been extensively studied also in recent years for its relationships with computer science ([4]) and fuzzy logics ([14]). For a survey in proof theory in Gödel-Dummett logics we quote [6].

To perform automated deduction both tableau and sequent calculi have been proposed. Paper [1] provides tableau calculi whose distinguishing feature is a multiple premise rule for implicative formulas that are not-forced. The syntactical way to express this semantical meaning is by the sign **F**. We recall that the sign **F** comes from Smullyan ([18, 11]) and labels those formulas that in a sequent calculus occur in the right-hand side of $\Rightarrow$. A tableau calculus derived from those of [1] is provided in paper [9]. Its main feature is that the depth of every deduction is linearly bounded in the length of the formula to be proved.

The approach of [1], based on characterizing Dummett logic by means of the multiple premise rule, has been criticized because, from the worst case analysis perspective, there are simple examples of sets of formulas giving rise to a factorial number of branches in the number of formulas in the set. Paper [5] shows how to get rid of the multiple premise rule. New rules are provided whose correctness is strictly related to the semantics of Dummett logic. These ideas have been further developed and in paper [16]

a graph-theoretic decision procedure is described and implemented. The approach introduced in [5] has also disadvantages with respect to the multiple premise rule proposed in [1] and these disadvantages have been considered in [10], where also a new version of the multiple premise rule is proposed. This version, from a practical point of view, can reduce the branching when compared with the original one. Paper [10] also provides an implementation that outperforms the one of [16], thus proving that the approach based on the multiple premise rule of [1] deserves attention also from the practical point of view. As a matter of fact, on the one hand the rules of [5] give rise to two branches at most, on the other hand there are cases of formulas that multiple premise calculi decide with a number of steps lower than the calculi based on [5].

In this paper we continue our investigation around multiple premise calculi for Dummett logic. The tool we use to prove our results is the characterization of Dummett logic via linearly ordered Kripke models, whose elements are considered *worlds* or *states of knowledge* ordered w.r.t. $\leq$ and $\alpha \leq \beta$ means that, roughly speaking, $\beta$ is in a subsequent point of the time w.r.t. $\alpha$.

With the aim to reduce the size of the proofs, we present a calculus whose main feature is the way the non-forced formulas are handled. In tableau calculi proofs start by supposing that in the present state of knowledge $A$ is not forced. A proof of our calculus starts by adding a further semantical constraint, namely, by supposing that the present state of knowledge is the last where $A$ is not forced. This further constraint implies that in the present we have information about a different semantical status of $A$ in the future: if we conclude that there exists the next state of knowledge, then we deduce that in such a future state of knowledge $A$ is forced (note that this implies that starting from such a future state of knowledge $A$ is equivalent to $\top$). Such information can be handled to draw deductions about the semantical status of the formulas both in the present and in the future state of knowledge. As a consequence, our calculus handles two kinds of signed formulas: the first kind describes the semantical status of a formula at the present state of knowledge, the second kind describes the semantical status of a formula to the next state of knowledge. The introduction of the signs related to the next state of knowledge allows to introduce specialized rules that exploit the knowledge about the future to draw deductions about the present. In particular, we refer to the introduction of specialized and single conclusion rules to handle the formulas of the kind $\mathbf{F}(A \rightarrow B)$ (to be read "at the present state of knowledge, the fact $A \rightarrow B$ is not known"), thus reducing the branching generated by the multiple premise rule.

The semantical constraint used in calculus has also the advantage to change the effect of the application of the non-invertible rules. Roughly speaking, non-invertible rules draw conclusions about future states of knowledge having as premise facts about the present. The application of these rules has an effect in the proof-search, usually requiring the introduction of backtracking to guarantee the completeness of the proof-search procedure. The semantical effect is that in the construction of the counter model a new state of knowledge is added, corresponding to the facts derived by the non-invertible rule. In the known sequent/tableau calculi for intermediate logics there is no guarantee that in the conclusion of the non-invertible rules at least one fact that in the premise was explicitly unknown becomes explicitly known in the conclusion. In the semantical construction this means that there is no guarantee that between two subsequent worlds, knowledge increases. The two semantical levels used in our calculus allow us to provide non-invertible rules such that there is at least one fact that in the premise is explicitly signed as not known and in the conclusion is explicitly signed as known. This implies that moving from the present to the next state of knowledge there is at least one explicitly unknown fact of the present that becomes explicitly known in the future. This has also a correspondence in the construction of the Kripke counter model, where there are no two elements exactly forcing the same propositional variables. Such an increment of known information, that is formulas equivalent to $\top$, is exploited by the *replacement rule*, which replaces all the occurrences of a formula proved to be equivalent to the $\top$ with $\top$ . Such a replacement reduces the size of the set to be decided. Further reductions are possible by applying replacements based on the truth table of the connectives. Rules based on the replacement have been proved

effective to dramatically reduce the search space both in classical and intuitionistic logic ([17, 2]), thus we have developed a calculus whose rules are designed to conclude as much as possible about formulas equivalent to $\top$. We remark that by using the appropriate data structures to represent formulas and sets of formulas, the application of the replacement rule and the reduction rules based on the truth table of the connectives can be performed in constant time. The results of our prolog prototype show that the calculus we provide is suitable for an implementation outperforming those cited above.

## 2 Basic definitions, the calculus and general considerations

We consider the propositional language based on a denumerable set of propositional variables $\mathscr{PV}$, the boolean constants $\top$ and $\bot$ and the logical connectives $\neg, \wedge, \vee, \rightarrow$. We call *atoms* the elements of $\mathscr{PV} \cup \{\top, \bot\}$. In the following, formulas (respectively set of formulas and propositional variables) are denoted by letters $A$, $B$, $C\ldots$ (respectively $S$, $T$, $U$,... and $p$, $q$, $r$,...) possibly with subscripts or superscripts.

From the introduction we recall that *Dummett Logic* (**Dum**) can be axiomatized by adding to any axiom system for propositional intuitionistic logic the axiom scheme $(p \rightarrow q) \vee (q \rightarrow p)$ and a well-known semantical characterization of **Dum** is by *linearly ordered Kripke models*. In the paper *model* means a linearly ordered Kripke model, namely a structure $\underline{K} = \langle P, \leq, \rho, \Vdash \rangle$, where $\langle P, \leq, \rho \rangle$ is a linearly ordered set with $\rho$ minimum with respect to $\leq$ and $\Vdash$ is the *forcing relation*, a binary relation on $P \times (\mathscr{PV} \cup \{\top, \bot\})$ such that: (i) if $\alpha \Vdash p$ and $\alpha \leq \beta$, then $\beta \Vdash p$; (ii) for every $\alpha \in P$, $\alpha \Vdash \top$ holds and $\alpha \Vdash \bot$ does not hold. Hereafter we denote the members of $P$ by means of lowercase letters of the Greek alphabet.

The forcing relation is extended in a standard way to arbitrary formulas of our language as follows:

1. $\alpha \Vdash A \wedge B$ iff $\alpha \Vdash A$ and $\alpha \Vdash B$;
2. $\alpha \Vdash A \vee B$ iff $\alpha \Vdash A$ or $\alpha \Vdash B$;
3. $\alpha \Vdash A \rightarrow B$ iff, for every $\beta \in P$ such that $\alpha \leq \beta$, $\beta \Vdash A$ implies $\beta \Vdash B$;
4. $\alpha \Vdash \neg A$ iff for every $\beta \in P$ such that $\alpha \leq \beta$, $\beta \Vdash A$ does not hold.

We write $\alpha \nVdash A$ when $\alpha \Vdash A$ does not hold. It is easy to prove that for every formula $A$ the *persistence* property holds: If $\alpha \Vdash A$ and $\alpha \leq \beta$, then $\beta \Vdash A$. We say that $\beta$ is *immediate successor* of $\alpha$ iff $\alpha < \beta$ and there is no $\gamma \in P$ such that $\alpha < \gamma < \beta$. A formula $A$ *is valid in a model* $\underline{K} = \langle P, \leq, \rho, \Vdash \rangle$ if and only if $\rho \Vdash A$. It is well-known that **Dum** coincides with the set of formulas valid in all models.

The rules of our calculus $\mathbb{D}$ are in Figures 1-4, where rules in Figures 1 and 2 form the logical apparatus necessary to decide **Dum**, whereas rules in Figures 3 and 4 are optimization rules, introduced to speed-up the deduction. The calculus $\mathbb{D}$ works on signed formulas, that is well-formed formulas prefixed with one of the *signs* **T** (with **T**$A$ to be read "the fact $A$ is known at the present state of knowledge"), **F** (with **F**$A$ to be read "the fact $A$ is not known at the present state of knowledge"), $\mathbf{F_l}$ (with $\mathbf{F_l}A$ to be read "this is the last state of knowledge where $A$ is not known"), $\mathbf{F_n}$ (with $\mathbf{F_n}A$ to be read "$A$ is not known in the next state of knowledge") and $\mathbf{T_n}$ (with $\mathbf{T_n}A$ to be read "$A$ will be known in the next state of knowledge"), and on sets of signed formulas (hereafter we omit the word "signed" in front of "formula" in all the contexts where no confusion arises). Formally, the meaning of the signs is provided by the relation *realizability* ($\rhd$) defined as follows: Let $\underline{K} = \langle P, \leq, \rho, \Vdash \rangle$ be a model, let $\alpha \in P$, let $H$ be a signed formula and let $S$ be a set of signed formulas. We say that $\alpha$ *realizes* $H$ (respectively $\alpha$ *realizes* $S$ and $\underline{K}$ *realizes* $S$), and we write $\alpha \rhd H$ (respectively $\alpha \rhd S$ and $\underline{K} \rhd S$), if the following conditions hold:

1. $\alpha \rhd \mathbf{T}A$ iff $\alpha \Vdash A$;

$$\frac{S,\mathbf{T}(A\wedge B)}{S,\mathbf{T}A,\mathbf{T}B}\mathbf{T}_\wedge \qquad \frac{S,\mathbf{F}(A\wedge B)}{S,\mathbf{F}A|S,\mathbf{F}B}\mathbf{F}_\wedge \qquad \frac{S,\mathbf{T}\neg(A\wedge B)}{S,\mathbf{T}\neg A|S,\mathbf{T}\neg B}\mathbf{T}_{\neg\wedge} \qquad \frac{S,\mathbf{T}\neg\neg(A\wedge B)}{S,\mathbf{T}\neg\neg A,\mathbf{T}\neg\neg B}\mathbf{T}_{\neg\neg\wedge}$$

$$\frac{S,\mathbf{T}(A\vee B)}{S,\mathbf{T}A|S,\mathbf{T}B}\mathbf{T}_\vee \qquad \frac{S,\mathbf{F}(A\vee B)}{S,\mathbf{F}A,\mathbf{F}B}\mathbf{F}_\vee \qquad \frac{S,\mathbf{T}\neg(A\vee B)}{S,\mathbf{T}\neg A,\mathbf{T}\neg B}\mathbf{T}_{\neg\vee} \qquad \frac{S,\mathbf{T}\neg\neg(A\vee B)}{S,\mathbf{T}\neg\neg A|S,\mathbf{T}\neg\neg B}\mathbf{T}_{\neg\neg\vee}$$

$$\frac{S,\mathbf{F}\neg A}{S,\mathbf{T}\neg\neg A}\mathbf{F}_\neg \qquad \frac{S,\mathbf{T}\neg\neg\neg A}{S,\mathbf{T}\neg A}\mathbf{T}_{\neg\neg\neg} \qquad \frac{S,\mathbf{T}\neg(A\to B)}{S,\mathbf{T}\neg\neg A,\mathbf{T}\neg B}\mathbf{T}_{\neg\to} \qquad \frac{S,\mathbf{T}\neg\neg(A\to B)}{S,\mathbf{T}\neg A|S,\mathbf{T}\neg\neg B}\mathbf{T}_{\neg\neg\to}$$

$$\frac{S,\mathbf{T}((A\wedge B)\to C)}{S,\mathbf{T}(A\to(B\to C))}\mathbf{T}_{\to\wedge} \qquad \frac{S,\mathbf{T}(\neg A\to B)}{S,\mathbf{T}\neg\neg A|S,\mathbf{T}B}\mathbf{T}_{\to\neg} \qquad \frac{S,\mathbf{T}((A\vee B)\to C)}{S,\mathbf{T}(A\to C),\mathbf{T}(B\to C)}\mathbf{T}_{\to\vee}$$

$$\frac{S,\mathbf{T}((A\to B)\to C)}{S,\mathbf{F}(A\to p),\mathbf{T}(p\to C),\mathbf{T}(B\to p)|S,\mathbf{T}C}\mathbf{T}_{\to\to}\text{ with } p \text{ a new atom}$$

$$\frac{S,\mathbf{F_l}(A\vee B)}{S,\mathbf{F}A,\mathbf{F_l}B,\mathbf{T_n}B|S,\mathbf{F}B,\mathbf{F_l}A,\mathbf{T_n}A}\mathbf{F_l}_\vee \qquad \frac{S,\mathbf{F}(A\to B)}{S,\mathbf{T}A,\mathbf{F_l}B,\mathbf{T_n}B|S,\mathbf{F_n}(A\to B)}\mathbf{F}_\to$$

$$\frac{S,\mathbf{T_n}(A\wedge B)}{S,\mathbf{T_n}A,\mathbf{T_n}B}\mathbf{T_n}_\wedge \qquad \frac{S,\mathbf{T_n}(\neg A)}{S,\mathbf{T}\neg A}\mathbf{T_n}_\neg$$

$$\frac{S,\mathbf{F}(A\to B)}{S,\mathbf{T}A,\mathbf{F_l}B}\mathbf{F}_{\to_1}\text{, provided } \mathbf{T_n}B\in S$$

$$\frac{S,\mathbf{F_l}(A\to B)}{S,\mathbf{T}A,\mathbf{F_l}B,\mathbf{T_n}B}\mathbf{F_l}_\to \qquad \frac{S,\mathbf{F_l}(A\wedge B)}{S,\mathbf{F_l}A,\mathbf{T_n}A,\mathbf{T_n}B|S,\mathbf{T}A,\mathbf{T_n}B,\mathbf{F_l}B}\mathbf{F_l}_\wedge$$

$$\frac{S,\mathbf{T}(A\to B)}{S_{cl},\mathbf{T}\neg A|S_{cl},\mathbf{T}B}\mathbf{T}_\to\text{-cl} \qquad \frac{S,\mathbf{T}\neg\neg A}{S_{cl},\mathbf{T}A}\mathbf{T}_{\neg\neg}\text{-cl}$$
provided $S$ only contains $\mathbf{T}$ and $\mathbf{T_n}$-formulas

$$\frac{S,\mathbf{F_l}(\neg A)}{S_{cl},\mathbf{T}A}\mathbf{F_l}_\neg \qquad \frac{S,\mathbf{F_l}\bot}{S_{cl}}\mathbf{F_l}\bot, \qquad \frac{S,\mathbf{T_n}\bot}{S_{cl}}\mathbf{T_n}\bot,$$

where

$$S_{cl} \quad = \quad \{\mathbf{T}A|\mathbf{T}A\in S\}\cup\{\mathbf{T}\neg A|\mathbf{F}A\in S\}\cup\{\mathbf{T}\neg A|\mathbf{F_l}A\in S\}$$

Figure 1: The invertible rules of $\mathbb{D}$.

2. $\alpha\rhd\mathbf{F}A$ iff $\alpha\nVdash A$;
3. $\alpha\rhd\mathbf{F_n}A$ iff there exists $\beta>\alpha$, $\beta\rhd\mathbf{F}A$;
4. $\alpha\rhd\mathbf{T_n}A$ iff for every $\beta>\alpha$, $\beta\rhd\mathbf{T}A$;
5. $\alpha\rhd\mathbf{F_l}A$ iff $\alpha\rhd\mathbf{F}A$ and $\alpha\rhd\mathbf{T_n}A$;
6. $\alpha\rhd S$ iff $\alpha$ realizes every formula in $S$;

Before to enter into technical details, let us justify the introduction of the signs $\mathbf{F_l}$, $\mathbf{F_n}$ and $\mathbf{T_n}$ in the object language by means of a motivating case. Let us suppose that a world $\alpha$ of a model $\underline{K}$ realizes $S_{\mathbf{F}_\to}=\{\mathbf{F}(A_1\to B_1),\ldots,\mathbf{F}(A_n\to B_n)\}$. Then there exists the last element $\beta\geq\alpha$ such that $\beta\rhd\mathbf{T}A_j,\mathbf{F}B_j,S_{\mathbf{F}_\to}\setminus\{\mathbf{F}(A_j\to B_j)\}$. Without loss of generality we let $j=1$. Analogously, there exists an element $\gamma\geq\beta$ such that $\gamma\rhd\mathbf{T}A_i,\mathbf{F}B_i,\mathbf{T}A_1,S_{\mathbf{F}_\to}\setminus\{\mathbf{F}(A_1\to B_1),\mathbf{F}(A_i\to B_i)\}$. Without loss of generality we let $i=2$. Now, $\beta<\gamma$ or $\beta=\gamma$. If $\beta<\gamma$, then, since $\beta$ is last element where $\mathbf{F}B_1$ holds, it follows that

$$\frac{S, \mathbf{T}\neg\neg A}{S_{cl}, \mathbf{T}A | S_{\phi}, \mathbf{T}A}\mathbf{T}_{\neg\neg}\text{-Atom},$$ provided $S$ does not contain $\mathbf{F_n}$-formulas.

where

$$\begin{array}{rcl} S_{\phi} & = & \{\mathbf{T}A | \mathbf{T}A \in S\} \cup \{\mathbf{T}A | \mathbf{T_n}A \in S\} \cup \{\mathbf{T}A | \mathbf{F_l}A \in S\} \text{ and} \\ S_{cl} & = & \{\mathbf{T}A | \mathbf{T}A \in S\} \cup \{\mathbf{T}\neg A | \mathbf{F}A \in S\} \cup \{\mathbf{T}\neg A | \mathbf{F_l}A \in S\}; \end{array}$$

$$\frac{S, \mathbf{F_n}A_1, \ldots, \mathbf{F_n}A_u}{V_1 | \ldots | V_j | \ldots | V_u}\mathbf{F_n}$$

where:

for $j = 1, \ldots, u$, $V_j = S_c \cup \{\mathbf{F_n}A_1, \ldots, \mathbf{F_n}A_{j-1}, \mathbf{F_l}A_j, \mathbf{F}A_{j+1}, \ldots, \mathbf{F}A_u\}$;

$S_c = \{\mathbf{T}A | \mathbf{T}A \in S\} \cup \{\mathbf{T}A | \mathbf{F_l}A \in S\} \cup \{\mathbf{T}A | \mathbf{T_n}A \in S\}$;

Figure 2: The non-invertible rules of $\mathbb{D}$.

$$\frac{S, \mathbf{T}A}{S[A/\top], \mathbf{T}A}\text{Replace}\,\mathbf{T} \qquad \frac{S, \mathbf{T}\neg A}{S[A/\bot], \mathbf{T}\neg A}\text{Replace}\,\mathbf{T}_{\neg}$$

$$\frac{S, \mathbf{T}\neg\neg A}{S[\neg A/\bot], \mathbf{T}\neg\neg A}\text{Replace}\,\mathbf{T}_{\neg\neg} \qquad \frac{S, \mathbf{F_n}A, \mathbf{F_l}B}{S, \mathbf{F_n}A[B/\top], \mathbf{F_l}B}\text{Replace}\,\mathbf{F_l}$$

$$\frac{S, \mathscr{S}A, \mathbf{T_n}B}{S, \mathscr{S}A[B/\top], \mathbf{T_n}B}\text{Replace}\,\mathbf{T_n}, \text{ with } \mathscr{S} \in \{\mathbf{T_n}, \mathbf{F_n}\}$$

Figure 3: The Replacement rules

$$\frac{S}{S[A \wedge \bot/\bot]}\text{Simp}\wedge\bot \quad \frac{S}{S[\bot \wedge A/\bot]}\text{Simp}\bot\wedge \quad \frac{S}{S[A \wedge \top/A]}\text{Simp}\wedge\top \quad \frac{S}{S[\top \wedge A/A]}\text{Simp}\top\wedge$$

$$\frac{S}{S[A \vee \bot/A]}\text{Simp}\vee\bot \quad \frac{S}{S[\bot \vee A/A]}\text{Simp}\bot\vee \quad \frac{S}{S[A \vee \top/\top]}\text{Simp}\vee\top \quad \frac{S}{S[\top \vee A/\top]}\text{Simp}\top\vee$$

$$\frac{S}{S[\bot \to A/\top]}\text{Simp}\bot\to \quad \frac{S}{S[A \to \bot/\neg A]}\text{Simp}\to\bot \quad \frac{S}{S[\top \to A/A]}\text{Simp}\top\to \quad \frac{S}{S[A \to \top/\top]}\text{Simp}\to\top$$

$$\frac{S}{S[\neg\top/\bot]}\text{Simp}\neg\top \quad \frac{S}{S[\neg\bot/\top]}\text{Simp}\neg\bot$$

Figure 4: The Simplification rules

$\gamma \rhd \mathbf{T}B_1$ holds. If $\beta = \gamma$, then $\beta$ is the last element where both $B_1$ and $B_2$ are not forced. The example shows that we can give a rule ($\mathbf{F}\to$) taking into account that if $\alpha \rhd \mathbf{F}(A_i \to B_i)$, for $i = 1, \ldots, n$, then the set $\{\mathbf{T}A_i, \mathbf{F}B_i\}$ is realized in $\theta \in \{\alpha, \beta\}$ and for every world $\gamma > \theta$, $\gamma \rhd \mathbf{T}B_i$ holds. The sign $\mathbf{F_l}$ aims to codify such a semantical property of $B_i$.

Generalizing the case given above, let $S_{\mathbf{F}\to} = \{\mathbf{F}(A_1 \to B_1), \ldots, \mathbf{F}(A_n \to B_n)\}$ and let $S$ be a set of formulas. Let us suppose that $\alpha \rhd S, S_{\mathbf{F}\to}$. Then, for $i = 1, \ldots, n$, let us consider the element $\beta_i$ such that $\beta_i \rhd \mathbf{T}A_i, \mathbf{F}B_i$ and for every $\gamma > \beta_i$, $\gamma \rhd \mathbf{T}A_i, \mathbf{T}B_i$. Thus $\beta_i$ is the maximum element such that $\beta_i \rhd \mathbf{T}A_i, \mathbf{F}B_i$. Since $\alpha \rhd \mathbf{F}(A_i \to B_i)$, such an element $\beta_i$ exists. Let $\beta_j = \min\{\beta_1, \ldots, \beta_n\}$, with $j \in \{1, \ldots, n\}$. There are two cases: (i) $\beta_j = \alpha$, then we conclude that $\beta$ realizes the set $S, \mathbf{T}A_j, \mathbf{F_l}B_j, S_{\mathbf{F}\to} \setminus \{\mathbf{F}(A_j \to B_j)\}$; (ii) $\beta_j > \alpha$, then $\beta_j$ realizes the set $S_c, \mathbf{T}A_j, \mathbf{F_l}B_j, S_{\mathbf{F}\to} \setminus \{\mathbf{F}(A_j \to B_j)\}$, where, by the meaning of the signs,

$S_c$ consists of: (i) the formulas of $S$ signed with $\mathbf{T}$; (ii) the formulas $\mathbf{T}A$ such that $\mathbf{F_l}A \in S$.

With the aim of reducing the size of the proofs, we add further considerations justifying the introduction of the sign $\mathbf{F_n}$. Let us suppose that beside $\alpha \rhd S, S_{\mathbf{F}_\rightarrow}$ we also know that for every model $\beta_1, \ldots, \beta_{j-1} > \beta_j$ holds. This rules out that $\beta_j \rhd \mathbf{T}A_i, \mathbf{F_l}B_i$, for $i = 1, \ldots, j-1$, and implies that there exists an element $\gamma$ such that $\beta_j < \gamma$ and $\gamma \rhd \mathbf{F}(A_i \rightarrow B_i)$. The sign $\mathbf{F_n}$ aims to codify that the formulas $\mathbf{F}(A_i \rightarrow B_i)$ have the following semantical property: there exists an element $\gamma$ such that $\beta_j < \gamma$ and $\gamma \rhd \mathbf{F}(A_i \rightarrow B_i)$, thus the element $\beta_j$ is not the maximum element realizing $\mathbf{F}(A_i \rightarrow B_i)$. By the meaning of $\mathbf{F_n}$ it follows that $\beta_j \rhd S_c, \mathbf{F_n}(A_1 \rightarrow B_1), \ldots, \mathbf{F_n}(A_{j-1} \rightarrow B_{j-1}), \mathbf{T}A_j, \mathbf{F}B_j, S_{\mathbf{F}_\rightarrow} \setminus \{\mathbf{F}(A_1 \rightarrow B_1), \ldots, \mathbf{F}(A_{j-1} \rightarrow B_{j-1})\}$, where by the meaning of the signs, $S_c$ consists of: (i) the formulas of $S$ signed with $\mathbf{T}$; (ii) the formulas $\mathbf{T}A$ such that $\mathbf{F_l}A \in S$. These are the intuitions justifying the rules $\mathbf{F_n}$ and $\mathbf{F} \rightarrow$. In particular the aim of $\mathbf{F} \rightarrow$ is to distinguish if the last element realizing $\mathbf{F}(A \rightarrow B)$ is the same element $\alpha$ realizing the premise or an element $\beta$ such that $\alpha < \beta$. The rule $\mathbf{F_n}$ handles the formulas of the kind $\mathbf{F_n}A$. For every formula there exists a world $\beta$ such that $\alpha < \beta$ and $\beta$ is the last element such that $\beta \not\Vdash A$. The minimum of such $\beta$ does not force any formula in evidence in the premise.

The intuition behind the rule $\mathbf{F_n}$ can be explained as follows. Let us suppose that $\alpha \rhd S, \mathbf{F_n}A_1, \ldots, \mathbf{F_n}A_u$. Thus there exists $\beta_i$ such that $\alpha < \beta_i$ and $\beta_i \rhd \mathbf{F_l}A_i$, for $i = 1, \ldots, u$. We notice that $\beta_i$ realizes all the $\mathbf{T}$ formulas in $S$ and $\beta_i \rhd \mathbf{T}C$ if $\mathbf{F_l}C \in S$. Moreover, if $\beta_1 = \min\{\beta_1, \ldots, \beta_u\}$, then $\beta_1 \rhd \mathbf{F_l}A_1, \mathbf{F}A_2, \ldots, \mathbf{F}A_u$. If $\beta_2$ is the minimum and we know that there is no model realizing $\alpha \rhd S, \mathbf{F_n}A_1, \ldots, \mathbf{F_n}A_u$ having $\beta_1$ as the minimum, then, since $\beta_2 < \beta_1$ holds, we conclude that $\beta_2 \rhd \mathbf{F_n}A_1, \mathbf{F_l}A_2, \mathbf{F}A_3, \ldots, \mathbf{F}A_u$. Analogously, if $\beta_i$ is the minimum and we know that there is no model realizing $\alpha \rhd S, \mathbf{F_n}A_1, \ldots, \mathbf{F_n}A_u$ having $\beta_1$,... or $\beta_{i-1}$ as the minimum, then, since $\beta_i < \beta_1, \ldots, \beta_{i-1}$ holds, we conclude that $\beta_i \rhd \mathbf{F_n}A_1, \ldots, \mathbf{F_n}A_{i-1}, \mathbf{F_l}A_i, \mathbf{F}A_{i+1}, \ldots, \mathbf{F}A_u$.

The sign $\mathbf{T_n}$ is strictly related to $\mathbf{F_l}$ and conveys redundant information with respect to the completeness. For this reason we give only two rules to treat the formulas of this kind and the replacement rules. This presentation of the calculus takes into account the implementation of the prolog prototype. We have decided to embed in the logical part the information conveyed from the $\mathbf{F_l}$-formulas. The explicit introduction of $\mathbf{T_n}$ emphasizes how to handle the information of the $\mathbf{F_l}$-formulas and its computational cost. An alternative presentation could avoid to insert the sign $\mathbf{T_n}$. In this case the rule $\mathbf{F} \rightarrow_1$ becomes

$$\frac{S, \mathbf{F}(A \rightarrow B), \mathbf{F_l}B}{S, \mathbf{T}A, \mathbf{F_l}B}$$

Also the replacement rules for $\mathbf{T_n}$ can be restated as rules involving $\mathbf{F_l}$. The rules $\mathbf{T_n}\wedge$ and $\mathbf{T_n}\neg$ deserve some considerations. From the point of view of the presentation of the logical calculus they are not relevant and can be deleted from Figure 1. From the practical point of view, the application of these rules is useful because they break down the information deriving from $\mathbf{F_l}$ formulas and can be used, as an example, to turn a $\mathbf{F} \rightarrow$-formula into a $\mathbf{F_n} \rightarrow$-formula. Thus one wonders how to insert them in the decision procedure. Our implementation treats explicitly the information derived from $\mathbf{F_l}$-formulas by means of the sign $\mathbf{T_n}$ and of the logical rules. A different choice could be to use the $\mathbf{F_l}$-formulas and implicitly break down the semantical information of the $\mathbf{F_l}$-formulas to decide, as an example, if a given $\mathbf{F} \rightarrow$-formula can be turned into a $\mathbf{F_l} \rightarrow$-formula, or if $\mathbf{F_n}A$ is equivalent to (the inconsistent) formula $\mathbf{F_n}\top$. Our choice is to insert all the machinery in the logical apparatus of the calculus. Finally, the sign $\mathbf{T_n}$ permits to introduce many other special rules allowing a reduction of the branching. We postpone to Section 6 the discussion of these special rules.

From the meaning of the signs we get the conditions that make a set of formulas inconsistent. A set $S$ is *inconsistent* if one of the following conditions holds:

$$\begin{array}{llll} \text{-}\mathbf{T}\bot \in S; & \text{-}\mathbf{F}\top \in S; & \text{-}\mathbf{F_l}\top \in S; & \text{-}\mathbf{F_n}\top \in S; \\ \text{-}\{\mathbf{F_n}A, \mathbf{F_l}\neg B\} \subseteq S; & \text{-}\{\mathbf{F_n}A, \mathbf{T_n}\bot\} \subseteq S; & \text{-}\{\mathbf{F_n}A, \mathbf{F_l}\bot\} \subseteq S; & \end{array}$$

We emphasize that inconsistency conditions of the last line are related to the existence of a future state of knowledge $\alpha$ and in such a $\alpha$ the other formula of the pair is not realizable. It is easy to prove the following

**Proposition 1.** *If a set of formulas $S$ is inconsistent, then for every Kripke model $\underline{K} = \langle P,\leq,\rho,\Vdash\rangle$ and for every $\alpha \in P$, $\alpha \not\rhd S$.*

*Proof.* We only consider some significant the case $\{\mathbf{F_n}A, \mathbf{F_l}\neg B\} \subseteq S$. By absurd, let us suppose that $\alpha \rhd S$, then $\alpha \rhd \mathbf{F_l}\neg B$ and $\alpha \rhd \mathbf{F_n}A$. Since $\alpha \rhd \mathbf{F_n}A$, there exists $\beta \in P$ such that $\alpha < \beta$. By definition of $\mathbf{F_l}$, it follows that $\alpha \nVdash \neg B$ and for every $\beta \in P$, if $\alpha < \beta$, then $\beta \Vdash \neg B$. Note that such a $\beta$ exists. Since $\underline{K}$ is a linearly ordered Kripke model, by definition of negation it follows that $\alpha \Vdash \neg B$. Thus we have that $\alpha \nVdash \neg B$ and $\alpha \Vdash \neg B$, absurd. The other cases are easy to prove.

□

A proof table (or proof tree) for $S$ is a tree, rooted in $S$ and obtained by the subsequent instantiation of the rules of the calculus. The premise of the rules are instantiated in a duplication-free style: in the application of the rules we always consider that the formulas in evidence in the premise are not in $S$. We say that a rule $\mathscr{R}$ applies to a set $U$ when it is possible to instantiate the premise of $\mathscr{R}$ with the set $U$ and we say that a rule $\mathscr{R}$ applies to a formula $H \in U$ (respectively the set $\{H_1,\ldots,H_n\} \subseteq U$) to mean that it is possible to instantiate the premise of $\mathscr{R}$ taking $S$ as $U \setminus \{H\}$ (respectively $U \setminus \{H_1,\ldots,H_n\}$).

A *closed proof table* is a proof table whose leaves are all inconsistent sets. A closed proof table is a proof of the calculus and a formula $A$ is provable iff there exists a closed proof table for $\{\mathbf{F_l}A\}$. We refer to [13] for full details on tableaux systems.

The calculus contains two non-invertible rules. In Section 4 we prove that it is possible to devise a complete strategy that relies on respecting a particular sequence in the application of the rules: $\mathbf{T}\neg\neg$-Atom is applied if no other rule is applicable and $\mathbf{F_n}$ is applied if no other rule but $\mathbf{T}\neg\neg$-Atom is applicable.

## 3 Correctness

To prove the correctness of $\mathbb{D}$ with respect to Dummett logic we need to prove that, if there exists a closed proof table for $\{\mathbf{F_l}A\}$, then $A$ is a valid formula in Dummett logic. The main step is to prove that the rules of the calculus preserve realizability:

**Proposition 2.** *For every rule of $\mathbb{D}$, if a model realizes the premise, then there exists a model realizing at least one of the conclusions.*

*Proof.* Let $\alpha$ be an element of $\underline{K} = \langle P,\leq,\rho,\Vdash\rangle$. We analyze the correctness of some rules of $\mathbb{D}$.
Rule $\mathbf{F_l}\rightarrow$: if $\alpha \rhd S, \mathbf{F_l}(A \rightarrow B)$, then, by definition of $\mathbf{F_l}$, $\alpha \nVdash A \rightarrow B$ and for every $\beta \in P$, if $\alpha < \beta$, then $\beta \Vdash A \rightarrow B$. This implies, by definition of forcing of an implicative formula, that $\alpha \Vdash A$ and $\alpha \nVdash B$ hold. By the preservation of forcing we get that, for every $\beta \in P$, if $\alpha < \beta$, then $\beta \Vdash A$, thus $\beta \Vdash B$. Therefore $\alpha \rhd \mathbf{F_l}B$.

Rule $\mathbf{F_l}\wedge$: if $\alpha \rhd S, \mathbf{F_l}(A \wedge B)$ then, by definition of $\mathbf{F_l}$, $\alpha \nVdash A \wedge B$, thus $\alpha \nVdash A$ or $\alpha \nVdash B$. Moreover, for every $\beta \in P$, if $\alpha < \beta$, then $\beta \Vdash A \wedge B$, thus $\beta \Vdash A$ and $\beta \Vdash B$. We have two possible cases: (i) $\alpha \nVdash A$, thus $\alpha \rhd \mathbf{F_l}A$, $\alpha \rhd \mathbf{T_n}A$ and $\alpha \rhd \mathbf{T_n}B$; (ii) $\alpha \Vdash A$, thus $\alpha \rhd \mathbf{T}A$, $\alpha \rhd \mathbf{F_l}B$ and $\alpha \rhd \mathbf{T_n}B$.

Rule $\mathbf{F_l}\neg$: if $\alpha \rhd S, \mathbf{F_l}(\neg A)$, then we notice that no $\mathbf{F_n}$-formula is in $S$. By definition of $\mathbf{F_l}$, $\alpha \nVdash \neg A$, thus there exists $\gamma \in P$ such that $\alpha \leq \gamma$ and $\gamma \Vdash A$. Moreover, for every $\beta \in P$, if $\alpha < \beta$, then $\beta \Vdash \neg A$. By

definition of forcing, an element cannot force a formula and its negation, thus the only possibility is that $\alpha$ is the final element of $\underline{K}$, thus $\gamma = \alpha$. Since $\alpha$ is the final element of $\underline{K}$, for every formula $B$, $\alpha \nVdash B$ implies $\alpha \Vdash \neg B$ and $\alpha \Vdash \neg\neg B$ implies $\alpha \Vdash B$. Thus $\alpha \rhd S_{cl}$.

Rule $\mathbf{T}\neg\neg$-Atom: if $\alpha \rhd S, \mathbf{T}\neg\neg A$, then we have two cases: (i) $\alpha$ is not the final element $\phi$. By the semantical meaning of the signs, it follows that $\phi$ realizes all the $\mathbf{T}$-formulas in $S$ and if $\mathbf{F_l}B \in S$, then $\phi \rhd \mathbf{T}B$; (ii) if $\alpha$ is the final element, then it is immediate to check that $\alpha$ realizes the leftmost conclusion of $\mathbf{T}\neg\neg$-Atom.

Rule $\mathbf{T_n}\bot$: if $\alpha \rhd S, \mathbf{T_n}\bot$, then for every $\beta \in P$, if $\alpha < \beta$, then $\beta \Vdash \bot$. Since no world forces $\bot$ the only possibility is that $\alpha < \beta$ does not hold, thus $\alpha$ is the final element of $\underline{K}$. Since $\alpha$ behaves as a classical model with respect to the semantics of the connectives, we get that $\alpha \rhd S_{cl}$.

□

**Theorem 1.** *If there exists a proof table for A, then A is valid in* **Dum**.

*Proof.* By hypothesis there exists a proof table starting from $\{\mathbf{F_l}A\}$ whose leaves are all inconsistent sets. An inconsistent set is not realizable and the rules of $\mathbb{D}$ preserve the realizability, thus $\mathbf{F_l}A$ is not realizable. By absurd, let us suppose that there exist a model $\underline{K} = \langle P, \leq, \rho, \Vdash \rangle$ and a world $\alpha \in P$ such that $\alpha \nVdash A$. Then, there exists $\beta \in P$ s.t. $\alpha \leq \beta$, $\beta \nVdash A$ and for every $\gamma \in P$ s.t $\beta < \gamma$, $\gamma \Vdash A$ holds. Thus $\beta \rhd \mathbf{F_l}A$ and, since the rules of $\mathbb{D}$ preserve the realizability, the leaves of the proof table are realized, which is impossible. We conclude that that for every model $\underline{K} = \langle P, \leq, \rho, \Vdash \rangle$ and every $\alpha \in P$, $\alpha \Vdash A$, that is $A$ is valid in **Dum**. □

The aim of the sign $\mathbf{F_l}$ is to label those formulas that after an application of $\mathbf{F_n}$ become equivalent to $\top$. After an application of the rule $\mathbf{F_n}$, the formulas signed with $\mathbf{F_l}$ become signed with $\mathbf{T}$, thus they are equivalent to $\top$. Hence the rule Replace$\mathbf{T}$ of Fig. 3 are applicable. The occurrences of $\top$ are removed by using the simplification rules of Fig. 4. We refer to the whole machinery of Fig. 3 and 4 as *reduction rules*. It is an easy task to check that the rules are invertible. We emphasize that Replace$\mathbf{T}\neg$ and Replace$\mathbf{T}\neg\neg$ are a specialization of Replace$\mathbf{T}$. The rule Replace$\mathbf{F_l}$ exploits the meaning of $\mathbf{F_l}$ and $\mathbf{F_n}$ to perform a replacement: if $\alpha \rhd \mathbf{F_n}A, \mathbf{F_l}B$ holds, then we conclude $\beta \rhd \mathbf{F}A, \mathbf{T}B$, with $\beta$ immediate successor of $\alpha$. Thus $\beta \rhd \mathbf{F}A[B/\top]$ and $\alpha \rhd \mathbf{F}A[B/\top]$ hold.

It can be noticed that in the conclusion of the $\mathbf{F_l}$-rules there is a duplication of a subformula of the premise. This duplication, that occurs with sign $\mathbf{T_n}$, is correct but not necessary to the completeness and is related to the logical treatment of the information under sign $\mathbf{F_l}$. It has to be noticed that this duplication does not explode in an exponential number of formulas since the presence of the rule Replace$\mathbf{T_n}$ allows the $\mathbf{T_n}$-formulas generated by duplication from $\mathbf{F_l}$-formulas to be treated only once. As an example, consider $\mathbf{F_l} \rightarrow$ and let us suppose that $B$ occurs as subformula of $\mathbf{T_n}C$ in $S$. By applying Replace$\mathbf{T_n}$ the occurrence of $B$ in $C$ is replaced by $\top$, thus the rules of the calculus handle the connectives of $B$ twice: once when $\mathbf{F_l}B$ is handled and once when $\mathbf{T_n}B$ is handled. We also remark that there are only two rules for $\mathbf{T_n}$-formulas. This implies that the overhead necessary to manage by $\mathbf{T_n}$-formulas the information conveyed by $\mathbf{F_l}$-formulas does not implies the generation of branches and requires at most a number of applications of rules of $\mathbb{D}$ which is linear in the length of the $\mathbf{F_l}$-formulas.

Compared with the calculus in paper [10], introducing the sign $\mathbf{F_l}$ has a price: proof tables can be wider, because of the combined action of $\mathbf{F} \rightarrow$ and $\mathbf{F_n}$. To reduce this problem there is the rule $\mathbf{F} \rightarrow_1$, which is not necessary to the completeness and represents an example of exploitation of $\mathbf{T_n}$-formulas. Moreover, since the application of rules in Fig.2 turns the $\mathbf{F_l}$-formulas into $\mathbf{T}$-formulas, that is formulas equivalent to the logical value $\top$, we investigate if this feature can reduce the depth of the proofs. We

have experienced that the reduction rules reduce the size of the proofs, for this reason we are devising a calculus that introduces as much as possible formulas equivalent to the logical value $\top$.

# 4 A Strategy to Decide Dummett Logic and Its Completeness

In the following we sketch the recursive procedure DUM$(S)$. Given a set $S$ of formulas, DUM$(S)$ returns either a closed proof table for $S$ or NULL (if there exists a model realizing $S$). To describe DUM we use the following definitions and notations. We call $\alpha$-rules and $\beta$-rules the rules of Figure 1 with one conclusion and with two conclusions, respectively. The $\alpha$-formulas and $\beta$-formulas are the kind of the signed formulas in evidence in the premise of the $\alpha$-rules and $\beta$-rules, respectively (e.g. $\mathbf{T}(A \wedge B)$ is an $\alpha$-formula and $\mathbf{T}(A \vee B)$ is a $\beta$-formula). Let $S$ be a set of formulas, let $H \in S$ be an $\alpha$ or $\beta$-formula. With $Rule(H)$ we denote the rule corresponding to $H$ in Figure 1. Let $S_1$ or $S_1 \mid S_2$ be the nodes of the proof tree obtained by applying to $S$ the rule $Rule(H)$. If $Tab_1$ and $Tab_2$ are closed proof tables for $S_1$ and $S_2$ respectively, then $\frac{S}{Tab_1}Rule(H)$ or $\frac{S}{Tab_1 \mid Tab_2}Rule(H)$ denote the closed proof table for $S$ defined in the obvious way. Moreover, for $H$ different from $\mathbf{F_l}\bot$ and $\mathbf{T_n}\bot$, $\mathscr{R}_i(H)$ $(i = 1,2)$ denotes the set containing the formulas of $S_i$ which replaces $H$. For instance:
$\mathscr{R}_1(\mathbf{T}(A \wedge B)) = \{\mathbf{T}A, \mathbf{T}B\}$,
$\mathscr{R}_1(\mathbf{T}(A \vee B)) = \{\mathbf{T}A\}$, $\mathscr{R}_2(\mathbf{T}(A \vee B)) = \{\mathbf{T}B\}$,
In the case of $\mathbf{F_n}$ we generalize the above notation. Let $S_{\mathbf{F_n}}$ be the set of all the $\mathbf{F_n}$-formulas of $S$. Let $S_1|\ldots|S_n$ be the nodes of the proof tree obtained by applying to $S$ the rule $\mathbf{F_n}$. If $Tab_1 \ldots, Tab_n$ are closed proof tables for $S_1, \ldots, S_n$, respectively, then $\frac{S}{Tab_1 \mid \ldots \mid Tab_n}\mathbf{F_n}$ is the closed proof table for $S$. With $\mathscr{R}_i(S_{\mathbf{F_n}})$ we denote the set of formulas that replaces the set $S_{\mathbf{F_n}}$ in the $i$-th conclusion of $\mathbf{F_n}$. For example, given $S_{\mathbf{F_n}} = \{\mathbf{F_n}A_1, \mathbf{F_n}A_2, \mathbf{F_n}A_3\}$, $\mathscr{R}_2(S_{\mathbf{F_n}\rightarrow}) = \{\mathbf{F_n}A_1, \mathbf{F_l}A_2, \mathbf{F}A_3\}$.

We consider SIMPLIFICATION as function that applies to its parameters the reduction rules as long as possible.

FUNCTION DUM (S)

**1.** If $S$ is an inconsistent set, then DUM returns the proof $S$;

**2.** Let $S' =$SIMPLIFICATION$(S)$. If $S' \neq S$, then let $\pi =$ DUM$(S')$. If $\pi$ is a proof, then DUM returns $\frac{S}{\pi}$SIMP, otherwise DUM returns NULL;

**3.** If the rule $\mathbf{F_l}\bot$ or $\mathbf{T_n}\bot$ applies to $S$, then let $H \in S$ be the formula $\mathbf{T_n}\bot$ or $\mathbf{F_l}\bot$. If DUM$((S \setminus \{H\})_{cl})$ returns a proof $\pi$, then DUM returns the proof $\frac{S}{\pi}Rule(H)$, otherwise DUM returns NULL;

**4.** If the rule $\mathbf{T}\neg\neg$-cl or $\mathbf{F_l}\neg$ applies to $S$, then let $H \in S$ be a formula of the kind $\mathbf{T}\neg\neg A$ or $\mathbf{F_l}\neg$. If DUM$((S \setminus \{H\})_{cl} \cup \{\mathbf{T}A\})$ returns a proof $\pi$, then DUM returns the proof $\frac{S}{\pi}Rule(H)$, otherwise DUM returns NULL;

**5.** If an $\alpha$-rule applies to $S$, then let $H$ be a $\alpha$-formula of $S$. If DUM$((S \setminus \{H\}) \cup \mathscr{R}_1(H))$ returns a proof $\pi$, then DUM returns the proof $\frac{S}{\pi}Rule(H)$, otherwise DUM returns NULL;

**6.** If the rule $\mathbf{T}\rightarrow$-cl applies to $S$, then let $H \in S$ be a formula of the kind $\mathbf{T}(A \rightarrow B)$. Let $\pi_1 =$ DUM$((S \setminus \{H\})_{cl} \cup \{\mathbf{T}\neg A\})$ and $\pi_2 =$ DUM$((S \setminus \{H\})_{cl} \cup \{\mathbf{T}B\})$. If $\pi_1$ or $\pi_2$ is NULL, then DUM returns NULL, otherwise DUM returns $\frac{S}{\pi_1 \mid \pi_2}\mathbf{T}\rightarrow$-cl;

**7.** If a $\beta$-rule applies to $S$, then let $H$ be a $\beta$-formula of $S$. Let $\pi_1 =$ DUM$((S \setminus \{H\}) \cup \mathscr{R}_1(H))$ and $\pi_2 =$ DUM$((S \setminus \{H\}) \cup \mathscr{R}_2(H))$. If $\pi_1$ or $\pi_2$ is NULL, then DUM returns NULL, otherwise DUM returns $\frac{S}{\pi_1 \mid \pi_2}Rule(H)$;

**8.** If the rule $\mathbf{F_n}$ applies to $S$, then let $S_{\mathbf{F_n}} = \{\mathbf{F_n}A \in S\}$ and $n = |S_{\mathbf{F_n}}|$. If there exists $i \in \{1, \ldots, n\}$, such that $\pi_i =$ DUM$((S \setminus S_{\mathbf{F_n}})_c \cup \mathscr{R}_i(S_{\mathbf{F_n}}))$ is NULL, then DUM returns NULL. Otherwise $\pi_1, \ldots, \pi_n$ are proofs

and DUM returns $\frac{S}{\pi_1 \mid \ldots \mid \pi_n}\mathbf{F_n}$;

**9.** If the rule $\mathbf{T}\neg\neg$-Atom applies to $S$, then let $H$ be a $\mathbf{T}\neg\neg$-Atom formula of $S$. Let $\pi_1 = \text{DUM}((S \setminus \{H\})_{cl} \cup \mathscr{R}_1(H))$ and $\pi_2 = \text{DUM}((S \setminus \{H\})_{\phi} \cup \mathscr{R}_2(H))$. If $\pi_1$ or $\pi_2$ is NULL, then DUM returns NULL, otherwise DUM returns $\frac{S}{\pi_1 \mid \pi_2}Rule(H)$;

**10.** If none of the previous points apply, then DUM returns NULL.

END FUNCTION DUM.

We emphasize that function DUM respects a particular sequence in the application of the rules: $\mathbf{T}\neg\neg$-Atom is applied if no other rule is applicable and $\mathbf{F_n}$ is applied if no other rule but $\mathbf{T}\neg\neg$-Atom is applicable. As a result no backtracking step is necessary.

By inspecting the rules of the calculus it follows that the procedure terminates. In particular the rightmost conclusion of the rule $\mathbf{F}\rightarrow$ only change the sign of the formula in evidence in the premise, but we remark that the rule $\mathbf{F_n}$ handles these kind of formulas, thus the combined action of $\mathbf{F}\rightarrow$ and $\mathbf{F_n}$ allows to prove that there is no an infinite loop handling $\mathbf{F}\rightarrow$-formulas.

In order to get the completeness of DUM, in the following it is proved that given a set of formulas $S$, if the call of $\text{DUM}(S)$ returns NULL, then there is enough information to build a model $\underline{K} = \langle P,\leq,\rho,\Vdash\rangle$ such that $\rho \rhd S$.

**Lemma 1** (Completeness)**.** *Let $S$ be a set of formulas and suppose that* $\text{DUM}(S)$ *returns* NULL*. Then, there exists a Kripke model $\underline{K} = \langle P,\leq,\rho,\Vdash\rangle$ such that $\rho \rhd S$.*

*Proof.* By induction on the number of nested recursive calls.

*Basis:* There are no recursive calls. Then Step 10 has been performed. We notice that $S$ is not inconsistent (otherwise Step 1 would have been performed) and $S$ does not contain neither $\mathbf{F_n}$-formulas nor $\mathbf{T}\neg\neg$-formulas (otherwise Step 8 or Step 9 would have been performed). Indeed, $S$ only contains atomic formulas signed with $\mathbf{T}$,$\mathbf{F}$,$\mathbf{T_n}$, $\mathbf{F_l}$, formulas of the kind $\mathbf{T}\neg p$, with $p$ atomic, and formulas of the kind $\mathbf{T}(p \rightarrow A)$ with $p$ atomic and $\mathbf{T}p \notin S$ (otherwise, if $\mathbf{T}p \in S$ holds, then Step 2 applies and at least a recursive call to DUM would have been performed). It is easy to prove that the model $\underline{K} = \langle P,\leq,\rho,\Vdash\rangle$, where $P = \{\rho\}$, $\rho \leq \rho$ and $\rho \Vdash p$ iff $\mathbf{T}p \in S$, realizes $S$.

*Step:* By induction hypothesis we assume that the proposition holds for all sets $S'$ such that $\text{DUM}(S')$ requires less than $n$ recursive calls. We prove the proposition holds for a set $S$ such that $\text{DUM}(S)$ requires $n$ recursive calls by inspecting all the possible cases where the procedure returns the NULL value.

NULL *value returned performing Step 4 with $H$ of the kind* $\mathbf{F_l}(\neg A)$. By induction hypothesis there exists a model $\underline{K}$ realizing $S_{cl}, \mathbf{T}A$. Notice that $S$ does not contain $\mathbf{F_n}$-formulas, otherwise $S$ would be inconsistent, and $S_{cl}$ contains only $\mathbf{T}$-formulas. This implies that all the formulas occurring in the subsequent recursive calls are signed with $\mathbf{T}$. Thus no step related to the rules of Fig. 2 is employed. We conclude that $\rho$ is the only element of $\underline{K}$ and $\rho \rhd S_{cl}$ implies $\rho \rhd S$. Moreover, $\rho$ maximal element of $\underline{K}$ and $\rho \rhd \mathbf{T}A$ imply $\rho \rhd \mathbf{F_l}(\neg A)$.

NULL *value returned performing Step 8.* We notice that $S$ is consistent, thus no formula of the kind $\mathbf{F_l}(\neg A)$ belongs to $S$. The set $S$ contains atomic formulas and formulas of the kind $\mathbf{T}(p \rightarrow A)$, $\mathbf{F_n}(A \rightarrow B)$, $\mathbf{F_n}(A \wedge B)$, $\mathbf{F_n}(A \vee B)$. Since the NULL instruction in Step 8 has been performed, at least a $\pi_i$ is NULL, then by induction hypothesis there is a model $\underline{K}' = \langle P,\leq',\rho',\Vdash'\rangle$ realizing $(S \setminus S_{\mathbf{F_n}})_c \cup \mathscr{R}_i(S_{\mathbf{F_n}})$. We define a

Kripke model $\underline{K} = \langle P \cup \{\rho\}, \leq, \rho, \Vdash \rangle$, where

$$\begin{array}{rcl} P \cap \{\rho\} & = & \emptyset, \\ \leq & = & \leq' \cup \{(\rho, \alpha) | \alpha \in P\} \\ \Vdash & = & \Vdash' \cup \{(\rho, p) | \mathbf{T}p \in S\} \end{array}$$

Since $\underline{K}'$ is a Dummett model realizing $(S \setminus S_{\mathbf{F_n}})_c$, it follows that $\underline{K}$ is a Dummett model. As a matter of fact, $\rho'$ is the immediate successor of $\rho$ and $\mathbf{T}A \in S$ implies $\mathbf{T}A \in (S \setminus S_{\mathbf{F_n}})_c$, thus the forcing relation is preserved. Moreover, $\mathbf{F_l}A \in S$ implies $\mathbf{T}A \in (S \setminus S_{\mathbf{F_n}})_c$ and the consistency of $S$ implies $\mathbf{T}A \notin S$. Thus we have proved that $\underline{K}' \rhd (S \setminus S_{\mathbf{F_n}})_c \cup \mathscr{R}_i(S_{\mathbf{F_n}})$. It is an easy task to check that $\underline{K}$ realizes $S$.

`NULL` *value returned performing Step 9.* We notice that when Step 9 is reached $S$ does not contain any $\mathbf{F_n}$-formula. Thus if a $\mathbf{T}\neg\neg$-formula belongs to $S$ the recursive call is performed and it is a correct application of the rule $\mathbf{T}\neg\neg$-Atom. Since Step 9 returns `NULL` one between $\pi_1$ or $\pi_2$ is `NULL`. If $\pi_1$ is `NULL`, then by induction hypothesis there is a model $\underline{K} = \langle P, \leq, \rho, \Vdash \rangle$ such that $\rho \rhd (S \setminus \{\mathbf{T}\neg\neg A\})_{cl} \cup \mathbf{T}A$. By the meaning of $(S \setminus \{\mathbf{T}\neg\neg A\})_{cl}$ immediately we get that $\rho \rhd S$. If $\pi_2$ is `NULL`, then, by induction hypothesis, the recursive call returns a model $\underline{K}' = \langle P, \leq', \rho', \Vdash' \rangle$ realizing $(S \setminus \{\mathbf{T}\neg\neg A\})_{\phi} \cup \{\mathbf{T}A\}$. We define a model $\underline{K} = \langle P \cup \{\rho\}, \leq, \rho, \Vdash \rangle$ as in the above point. Now, $\rho' \rhd \mathbf{T}A$ implies $\rho \rhd \mathbf{T}\neg\neg A$. Moreover, by definition of $(S \setminus \{\mathbf{T}\neg\neg A\})_{\phi}$, it is immediate to prove that $\rho \rhd (S \setminus \{\mathbf{T}\neg\neg A\})$ holds.

□

We conclude the section with a sketch of deduction with our calculus. We consider the provable formula SYJ201.001 of ILTP library:

$$H \equiv ((((p0 \to p1) \wedge (p1 \to p0)) \to C) \wedge (((p1 \to p2) \wedge (p2 \to p1)) \to C) \wedge (((p2 \to p0) \wedge (p0 \to p2)) \to C)) \to C$$

where $C \equiv (p0 \wedge (p1 \wedge p2))$.
The proof starts with $\mathbf{F_l}H$. Since $H$ is implicative, $\mathbf{F_l} \to$ is applied and we get

$$\begin{array}{rcl} S_1 & = & \{\mathbf{T}((((p0 \to p1) \wedge (p1 \to p0)) \to C) \wedge (((p1 \to p2) \wedge (p2 \to p1)) \to C) \wedge (((p2 \to p0) \wedge (p0 \to p2)) \to C)), \\ & & \mathbf{F_l}C, \mathbf{T_n}C\} \end{array}$$

By applying two times $\mathbf{T}\wedge$, two times $\mathbf{T_n}\wedge$ and three times $\mathbf{T} \to \wedge$ we get

$$\begin{array}{rcl} S_2 & = & \{\mathbf{T_n}p1, \mathbf{T_n}p2, \mathbf{T_n}p0, \mathbf{T}((p2 \to p0) \to ((p0 \to p2) \to H)), \mathbf{T}((p1 \to p2) \to ((p2 \to p1) \to H)), \\ & & \mathbf{T}((p0 \to p1) \to ((p1 \to p0) \to H)), \mathbf{F_l}C\} \end{array}$$

We apply to $S_2$ the rule $\mathbf{T} \to\to$, having $\mathbf{T}((p2 \to p0) \to ((p0 \to p2) \to H))$ as main premise. The result are two sets, we only consider the leftmost

$$\begin{array}{rcl} S_3 & = & \{\mathbf{F}(p2 \to p0), \mathbf{T}(p0 \to ((p0 \to p2) \to H)), \mathbf{T_n}p1, \mathbf{T_n}p2, \mathbf{T_n}p0, \\ & & \mathbf{T}((p1 \to p2) \to ((p2 \to p1) \to H)), \mathbf{T}((p0 \to p1) \to ((p1 \to p0) \to H)), \mathbf{F_l}C\} \end{array}$$

We apply to $S_3$ the rule $\mathbf{F} \to_1$ and SIMPLIFICATION as long as possible. The result is

$$\begin{array}{rcl} S_4 & = & \{\mathbf{T}p2, \mathbf{F_l}p0, \mathbf{T}(p1 \to (p0 \wedge p1)), \mathbf{T}(p0 \to (p0 \wedge p1)), \mathbf{T_n}p1, \mathbf{T_n}p0, \\ & & \mathbf{T}((p0 \to p1) \to ((p1 \to p0) \to (p0 \wedge p1))), \mathbf{F_l}(p0 \wedge p1)\} \end{array}$$

We apply to $S_4$ the rule $\mathbf{T} \to\to$. The results are two sets, we only consider the leftmost:

$$\begin{array}{rcl} S_5 & = & \{\mathbf{F}(p0 \to p1), \mathbf{T}(p1 \to ((p1 \to p0) \to (p0 \wedge p1))), \mathbf{T}p2, \mathbf{F_l}p0, \mathbf{T}(p1 \to (p0 \wedge p1)), \\ & & \mathbf{T}(p0 \to (p0 \wedge p1)), \mathbf{T_n}p1, \mathbf{T_n}p0, \mathbf{F_l}(p0 \wedge p1)\} \end{array}$$

We apply to $S_5$ the rule $\mathbf{F} \to_1$ and then SIMPLIFICATION as long as possible. We get

$$S_6 = \{\mathbf{T}p0, \mathbf{T}p2, \mathbf{T}p1, \mathbf{F_l}\top\}$$

which is contradictory. Along the same line we get a contradiction from the sets we left.

# 5 Implementation and Experimental Results

The prolog prototype EPDL-new[1] implements the strategy described in Section 4. The developing effort is not in the implementation of particular data structures but on the strategy to reduce the branching. As a result, the application of the reduction rules requires a quadratic number of steps in the length of the premise. By using the advanced data structures employed in [2], such a number can be reduced to be constant. Since the replacement rules have been proved useful to speed-up the intuitionistic deduction ([2]), beside the replacement rules of Figure 3, EPDL-new implements the replacement rules:

$$\frac{S, \mathbf{T}A, \mathbf{F_l}B}{S, \mathbf{T}A, \mathbf{T_n}A[B/\top], \mathbf{F_l}B}\,\text{Replace}\,\mathbf{F_l}-dup \qquad \frac{S, \mathbf{T}A, \mathbf{T_n}B}{S, \mathbf{T}A, \mathbf{T_n}A[B/\top], \mathbf{T_n}B}\,\text{Replace}\,\mathbf{T_n}-dup$$

provided a $\mathbf{F_n}$-formula is in $S$

The advantage of these two further replacement rules is that they are a mechanism to discover information that will be forced in the next state of knowledge. Since the sign $\mathbf{T}$ subsumes $\mathbf{T_n}$, an alternative is to introduce a $\mathbf{T_n}$-formula for every $\mathbf{T}$-formula in the set, but this would increase the number of $\mathbf{T_n}$-formulas also in cases where this information cannot be exploited. With our choice a $\mathbf{T_n}$-formula is introduced only when a new information about the future is explicitly known. The benefit of discovering as much $\mathbf{T_n}$-formulas as possible is related to the rule $\mathbf{F}\rightarrow_1$, a particular case of $\mathbf{F}\rightarrow$, that has one conclusion. The proviso on the application of the rules guarantees that we do not waste time by applying the replacement rules if there is no witness that a future state exists. Such a condition is necessary to guarantee the correctness of

$$\frac{S, \mathbf{T}\neg A, \mathbf{T_n}B}{S, \mathbf{T}\neg A[B/\top], \mathbf{T_n}B}\,\text{Replace}\,\mathbf{T_n}-special$$

provided a $\mathbf{F_n}$-formula is in $S$

a version of Replace $\mathbf{T_n}-dup$ that does not require duplication of the negated $\mathbf{T}$-formula. Since Replace $\mathbf{T_n}$-dup and Replace $\mathbf{F_l}$-dup copy the premise $\mathbf{T}A$, to avoid an infinite loops in the application of the rules a special labelling on the copied formulas is implemented.

In Fig. 5 we report the comparisons with EPDL ([10]) on the formulas of ILTP library (the ILTP family formulas from SYJ201 to SYJ206 are valid in Dummett logic). On the missing families the behaviour of the implementations is similar. On the the families of formulas SYJ201, SYJ205, SYJ207 and SYJ208 EPDL-new is clearly faster than EPDL and the timings of EPDL-new grow slower than EPDL. As regard the family SYJ202 (the pigeon principle) EPDL-new is clearly slower. This is due to the overhead to manage the $\mathbf{T_n}$-formulas introduced in the deduction that do not give any advantage to decide this formula. The overhead to manage the $\mathbf{T_n}$-formulas is twofold. In the deduction occur formulas of the kind $\mathbf{T_n}(A\wedge B)$, thus the decision procedure applies the rule $\mathbf{T_n}\wedge$. Moreover, the application of the rules in Fig. 3 and 4 requires, for both EPDL and EPDL-new, quadratic time in the number of connectives and atoms in the set of formulas. Since the set managed by EPDL-new is bigger than EPDL we have such an increment in the timings. As we noticed at the beginning of this section, by employing the data structures of [2], the time required to apply the reduction rules can be lowered from quadratic to constant in the number of symbols of the set to be handled. Thus the impact of the overhead is greatly reduced. As regard the family formulas SYJ211, we notice that the different timings have to be charged to the way the reduction rules are implemented. As a matter of fact, to decide the last formula of the family EPDL takes 127 rules and EPDL-new 147, where the difference in the number of the rules is due to the application of $\mathbf{T_n}\wedge$-rule. We also remark that the growing ratio of the two implementations is similar. Finally EPDL decides SYJ212+1.014 by applying 54 rules whereas EPDL-new applies 81. These formulas are

[1] available from http://www.dimequant.unimib.it/~guidofiorino/epdl.jsp

| Formula | EPDL | EPDL-new |
|---|---|---|
| SYJ201+1.002 | 0.21 | 0.03 |
| SYJ201+1.003 | 2.73 | 0.07 |
| SYJ201+1.004 | 33.63 | 0.14 |
| SYJ201+1.005 | 358.07 | 0.26 |
| | | |
| SYJ202+1.004 | 0.12 | 0.23 |
| SYJ202+1.005 | 1.02 | 2.24 |
| SYJ202+1.006 | 9.32 | 27.21 |
| SYJ202+1.007 | 100 | 335 |
| | | |
| SYJ205+1.009 | 12.04 | 2.18 |
| SYJ205+1.010 | 30.44 | 3.00 |
| SYJ205+1.011 | 75.57 | 3.84 |
| SYJ205+1.012 | 187.17 | 5.06 |
| | | |
| SYJ207+1.003 | 0.27 | 0.04 |
| SYJ207+1.004 | 3.33 | 0.10 |
| SYJ207+1.005 | 37.78 | 0.17 |
| SYJ207+1.006 | 419.80 | 0.27 |

| Formula | EPDL | EPDL-new |
|---|---|---|
| SYJ208+1.009 | 6.55 | 1.70 |
| SYJ208+1.010 | 14.75 | 3.05 |
| SYJ208+1.011 | 30.06 | 5.03 |
| SYJ208+1.012 | 71.28 | 8.23 |
| | | |
| SYJ211+1.017 | 0.17 | 1.05 |
| SYJ211+1.018 | 0.20 | 1.16 |
| SYJ211+1.019 | 0.22 | 1.38 |
| SYJ211+1.020 | 0.26 | 1.59 |
| | | |
| SYJ212+1.011 | 0.25 | 0.90 |
| SYJ212+1.012 | 0.49 | 1.90 |
| SYJ212+1.013 | 1.18 | 4.12 |
| SYJ212+1.014 | 2.22 | 8.25 |

Figure 5: EPDL and New-EPDL on ILTP formulas.

huge (SYJ212+1.014 is a formula in 14 variables, containing 131063 connectives and 98296 variable occurrences). Because of the duplication of the calculus, the sets of formulas managed by EPDL-new contains more symbols than those managed by EPDL and the difference in timings is mostly chargeable to the implementation of the reduction rules rather than to the difference on the number of applications of rules. Another clue to support our statement is that EPDL-new requires to apply 63, 69, 75 and 81 rules to decide respectively SYJ212+1.011, SYJ212+1.012, SYJ212+1.013 and SYJ212+1.014, whereas to decide the same formulas EPDL applies respectively 42, 46, 50 and 54 rules. We remark that the number of rules increases of a constant value for both provers, but the timings of the EPDL and EPDL-new approximately double.
Summarizing, on some families of formulas the ideas on which the calculus is based on do not apply, as a result the deduction is slowed-down. This difference is remarkable on the family SYJ202, the pigeon principle, that, apart from the first rule, is decided by a deduction requiring rules for connectives $\wedge$ and $\vee$. We point out that on the families SYJ202, SYJ211 and SYJ212 the multiple premise rule $\mathbf{F_n}$ is not applied. We conjecture that implementing the calculus with better data structures, on the family formulas SYJ211 and SYJ212 the difference in timings between EPDL and EPDL-new can be reduced almost to zero. Finally we remark that on the other families the deduction requires some steps of the multiple premise rule $\mathbf{F_n}$ and in this case both timings and growing ratio of EPDL-new are lower than EPDL.

## 6 Conclusions and future work

The main novelty of the calculus presented in this paper is the presence of rules tailored to draw deductions about the (immediate) future and rules exploiting the information about facts known in the future to draw facts about the present state of knowledge. The aim is to reduce the branching of the proof to decide

a formula in Dummett logic. The presence of the signs $\mathbf{F_l}$ and $\mathbf{T_n}$ allow us to introduce specialized rules in particular to handle implicative formulas. Rule $\mathbf{F}\rightarrow_1$ is an example but other specialized rules can be introduced: $\dfrac{S,\mathbf{F}(A\rightarrow B),\mathbf{F_l}A}{S,\mathbf{F_l}A,\mathbf{F_n}B}$ is a further example of single conclusioned rule for $\mathbf{F}\rightarrow$-formulas. Also formulas of the kind $\mathbf{T}((A\rightarrow B)\rightarrow C)$ can be handled by means of specialized rules:

$$\frac{S,\mathbf{T}((A\rightarrow B)\rightarrow C),\mathbf{F_l}(A\rightarrow B)}{S,\mathbf{F_l}(A\rightarrow B),\mathbf{T_n}C}\qquad \frac{S,\mathbf{T}((A\rightarrow B)\rightarrow C),\mathbf{F_l}C}{S,\mathbf{F}(A\rightarrow B),\mathbf{F_l}C}$$

$$\frac{S,\mathbf{T}((A\rightarrow B)\rightarrow C),\mathbf{T_n}(A\rightarrow B)}{S,\mathbf{F_l}(A\rightarrow B),\mathbf{T_n}C|S,\mathbf{T_n}(A\rightarrow B),\mathbf{T}C}\qquad \frac{S,\mathbf{T}((A\rightarrow B)\rightarrow C),\mathbf{T_n}C}{S,\mathbf{F}(A\rightarrow B),\mathbf{T_n}C|S,\mathbf{T}C}$$

These rules are a particular version of $\mathbf{T}\rightarrow\rightarrow$ allowing to reduce branching or the size of the conclusion.

As related works, we quote the implementation LC-cmodels ([16]), compared with EPDL in [10]. We also quote the approach of [3], where propositional Dummett logic is decided via a decision procedure for propositional Intuitionistic logic. Paper [3] introduces the notion of Generalized Tableaux to decide intermediate logics. A Generalized Tableau is a tableau for propositional Intuitionistic logic plus a rule to be applied once as first rule of the deduction. The aim of this rule is to introduce formulas obtained by instantiating the axiom scheme of the logic under consideration. For the case of Dummett logic, to decide a given formula $A$, the special rule introduces the set of formulas obtained by instantiating in every possible way the propositional variables the axiom schemata $(p\rightarrow q)\vee(q\rightarrow p)$ with the formulas in $Rsf(A)$=$\{B|B$ is subformula of $A$ and $B$ is a propositional variable or $B\equiv C\rightarrow D$ or $B\equiv\neg C\}$. Since $|Rsf(A)|=O(|A|)$ and there are $|Rsf(A)|$ choices for $p$ and $q$, it follows that the special rule introduces $O(|A|^2)$ formulas ($|A|$ denotes the cardinality of $A$). Thus the number of connectives to be handled in the deduction is $O(|A|^3)$. Paper [15] proves that propositional intuitionistic logic is decidable in $O(n\lg n)$-SPACE, hence this technique requires $O(n^3\lg n)$-SPACE and the depth of the deductions is $O(|A|^3)$. Another approach is the translation of $A$ into a formula to be decided in classical logic. One can exploit the fact that a formula with $n$ propositional variables is satisfiable in a Kripke model having $n+1$ worlds at most and writing a formula $A'$ expressing that for every $B$ and $C$, subformulas of $A$, $B\rightarrow C$ or $C\rightarrow B$ holds. Moreover also the persistence of the forcing has to be expressed. Thus the size of $A'$ is $O(|A|^3)$.

The ideas presented in this paper for propositional Dummett logic can be applied to intuitionistic deduction too. Here we sketch how to design a calculus for the propositional implicative fragment. The sign $\mathbf{F_n}$ is no longer necessary. Formulas of the kind $\mathbf{F}(A\rightarrow B)$ are handled by the rule $\dfrac{S,\mathbf{F}(A\rightarrow B)}{S,\mathbf{T}A,\mathbf{F_l}B|S_c,\mathbf{T}A,\mathbf{F_l}B}$ and formulas of the kind $\mathbf{T}(A\rightarrow B)$ are handled by the rule $\dfrac{S,\mathbf{T}(A\rightarrow B)}{S,\mathbf{T}B|S,\mathbf{F_l}A,\mathbf{T_n}B|S_c,\mathbf{F_l}A,\mathbf{T_n}B}$. We emphasize that these rules respect the subformula property, thus we are on the way to design a duplication-free calculus (in the sense of [1]) respecting the subformula property, as long as the replacement rules are not employed. As regard the decision procedure equipped with the replacement rules, we expect to improve the performances of fCube ([8]) as EPDL-new improves the performances of EPDL. Our consideration is based on the fact that, since intuitionistic logic is in PSPACE whereas Dummett logic is in NP, the overhead due to the application of the replacement and special rules is less in intuitionistic than in Dummett logic.

# References

[1] A. Avellone, M. Ferrari, and P. Miglioli. Duplication-free tableau calculi and related cut-free sequent calculi for the interpolable propositional intermediate logics. *Logic Journal of the IGPL*, 7(4):447–480, 1999.

[2] A. Avellone, G. Fiorino, and U. Moscato. Optimization techniques for propositional intuitionistic logic and their implementation. *Theoretical Computer Science*, 409(1):41–58, 2008.

[3] A. Avellone, P. Miglioli, U. Moscato, and M. Ornaghi. Generalized tableau systems for intermediate propositional logics. In D. Galmiche, editor, *Proceedings of the 6th International Conference on Automated Reasoning with Analytic Tableaux and Related Methods: Tableaux '97*, volume 1227 of *LNAI*, pages 43–61. Springer-Verlag, 1997.

[4] A. Avron. Simple consequence relations. *Journal of Information and Computation*, 92:276–294, 1991.

[5] A. Avron and B. Konikowska. Decomposition proof systems for gödel-dummett logics. *Studia Logica*, 69(2):197–219, 2001.

[6] M. Baaz, A. Ciabattoni, and C. G. Fermüller. Hypersequent calculi for Gödel logics – a survey. *J. of Logic and Computation*, 13(6):835–861, 2003.

[7] M. Dummett. A propositional calculus with a denumerable matrix. *Journal of Symbolic Logic*, 24:96–107, 1959.

[8] C. Fiorentini M. Ferrari and G. Fiorino. fcube: An efficient prover for intuitionistic propositional logic. In *LPAR 2010*.

[9] G. Fiorino. An $O(n \log n)$-SPACE decision procedure for the propositional Dummett Logic. *Journal of Automated Reasoning*, 27(3):297–311, 2001.

[10] G. Fiorino. Fast decision procedure for propositional dummett logic based on a multiple premise tableau calculus. *Information Sciences*, 180(19):3633 – 3646, 2010.

[11] M.C. Fitting. *Intuitionistic Logic, Model Theory and Forcing*. North-Holland, 1969.

[12] K. Gödel. On the intuitionistic propositional calculus. In S. Feferman et al, editor, *Collected Works*, volume 1. Oxford University Press, 1986.

[13] R. Hähnle. Tableaux and related methods. In John Alan Robinson and Andrei Voronkov, editors, *Handbook of Automated Reasoning*, pages 100–178. Elsevier and MIT Press, 2001.

[14] P. Hajek. *Metamathematics of Fuzzy Logic*. Kluwer, 1998.

[15] J. Hudelmaier. An $O(n \log n)$-SPACE decision procedure for intuitionistic propositional logic. *Journal of Logic and Computation*, 3(1):63–75, 1993.

[16] D. Larchey-Wendling. Graph-based decision for Gödel-Dummett logics. *J. Autom. Reasoning*, 38(1-3):201–225, 2007.

[17] F. Massacci. Simplification: A general constraint propagation technique for propositional and modal tableaux. In Harrie de Swart, editor, *Proc. International Conference on Automated Reasoning with Analytic Tableaux and Related Methods, Oosterwijk, The Netherlands*, volume 1397 of *LNCS*, pages 217–232. Springer-Verlag, 1998.

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

# A Prolog-based Proof Tool for Type Theory $TA_\lambda$ and Implicational Intuitionistic-Logic

L. Yohanes Stefanus
University of Indonesia
Depok 16424, Indonesia
yohanes@cs.ui.ac.id

Ario Santoso*
Technische Universität Dresden
Dresden 01187, Germany
santoso.ario@gmail.com

**Abstract**

Studies on type theory have brought numerous important contributions to computer science. In this paper we present a GUI-based proof tool that provides assistance in constructing deductions in type theory and validating implicational intuitionistic-logic formulas. As such, this proof tool is a testbed for learning type theory and implicational intuitionistic-logic. This proof tool focuses on an important variant of type theory named $TA_\lambda$, especially on its two core algorithms: the principal-type algorithm and the type inhabitant search algorithm. The former algorithm finds a most general type assignable to a given $\lambda$-term, while the latter finds inhabitants (closed $\lambda$-terms in $\beta$-normal form) to which a given type can be assigned. By the Curry–Howard correspondence, the latter algorithm provides provability for implicational formulas in intuitionistic logic. We elaborate on how to implement those two algorithms declaratively in Prolog and the overall GUI-based program architecture. In our implementation, we make some modification to improve the performance of the principal-type algorithm. We have also built a web-based version of the proof tool called $\lambda$-Guru.

## 1 Introduction

In 1902, Russell revealed the inconsistency in naive set theory. This inconsistency challenged the foundation of mathematics at that time. To solve that problem, type theory was introduced [1, 5]. As time goes on, type theory becomes widely used, it becomes a logician's standard tool, especially in proof theory. Since 1970s, the need for robust programming languages has made type theory the center of attention among computer scientists. Many practical programming languages have been developed based on type theory [4, 6].

Type theory has many variants, one of them is called $TA_\lambda$ (for *type-assignment in $\lambda$-calculus*). $TA_\lambda$ is one of the simplest type-theories. Many complex techniques for analyzing the structures of complex type-theories appear in $TA_\lambda$ in an easier form and hence easier to understand. Therefore, $TA_\lambda$ is a good vehicle for learning more complex techniques in type theory. By learning $TA_\lambda$, one can obtain a good foundation to study other type-systems [4].

The importance of $TA_\lambda$ and its relation to type theory in general drive this research project. This work focuses on two most important algorithms in $TA_\lambda$, i.e., the principal-type algorithm and the type inhabitant search algorithm. The former algorithm finds a most general type (principal type) assignable to a given $\lambda$-term, while the latter finds inhabitants (i.e., closed $\lambda$-terms in $\beta$-normal form) to which a given type can be assigned. Those two algorithms are implemented using Prolog and exploiting the ability of definite clause grammars (DCG) for processing linguistic expressions. DCG is a Prolog feature for expressing grammar rules [2]. In addition, a graphical user interface (GUI) is developed for helping users try those two algorithms. Therefore, the resulting software can be used as a testbed for learning $TA_\lambda$ and, by the Curry–Howard correspondence, as a proof assistant for implicational formulas in intuitionistic logic.

*Supported by the European Master's Program in Computational Logic (EMCL)

## 2 The Principal-Type Algorithm Implementation

The principal-type algorithm decides whether a $\lambda$-term is typable or not. It takes a $\lambda$-term as an input. If the given $\lambda$-term is typable, this algorithm will produce its principal (most general) type and its $\mathrm{TA}_\lambda$-deduction tree. If the given $\lambda$-term is not typable, this algorithm will give a correct statement that the given $\lambda$-term is not typable. Here are some examples of input and output of this algorithm:

1. **Input:** $\lambda yz.(yz)z$
   **Output:**

$$\dfrac{\dfrac{\dfrac{\dfrac{y : a{\to}a{\to}b \mapsto y : a{\to}a{\to}b \quad z : a \mapsto z : a}{y : a{\to}a{\to}b, z : a \mapsto yz : a{\to}b}\,(\to E) \qquad z : a \mapsto z : a}{y : a{\to}a{\to}b, z : a \mapsto (yz)z : b}\,(\to E)}{y : a{\to}a{\to}b \mapsto \lambda z.(yz)z : a{\to}b}\,(\to I)}{\mapsto \lambda yz.(yz)z : (a{\to}a{\to}b){\to}a{\to}b}\,(\to I)$$

2. **Input:** $\lambda xz.(xz)x$
   **Output:** The term is not typable

We do not explain the details of the algorithm in this paper due to space limitation. Please check the details in [3, 4].

In this implementation, we use a definite clause grammar (DCG) in Prolog for parsing the input as a $\lambda$-term. While parsing the $\lambda$-term, the parser constructs a tree that represents the given $\lambda$-term. If the given $\lambda$-term is valid, the program will apply the principal-type algorithm to decide whether the given $\lambda$-term is typable or not and produce a deduction for the typable case. Generally the process is described by the flowchart in Figure 1.

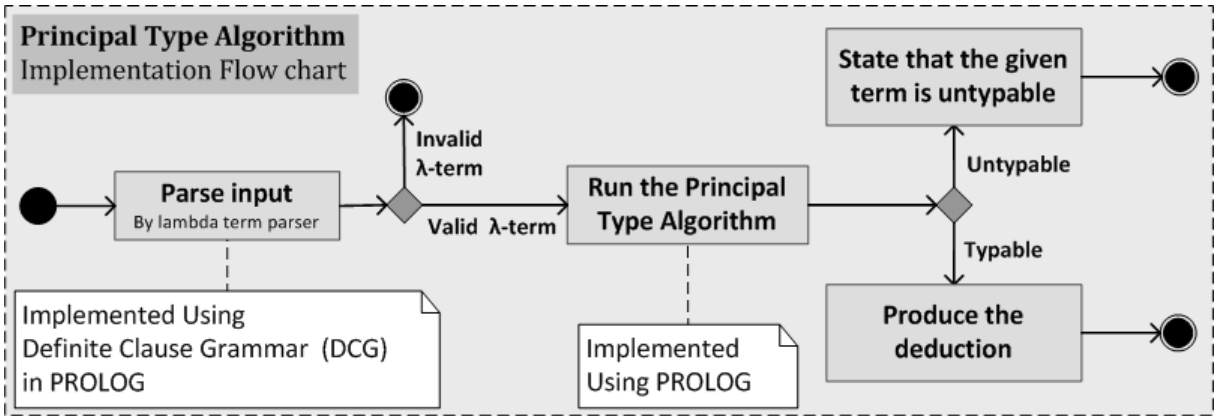



Figure 1: Flowchart for the Implementation of the Principal-Type Algorithm

The principal-type algorithm consists of four cases: (1) when the $\lambda$-term is a term variable, (2) when the $\lambda$-term is an application, (3) when the $\lambda$-term is an abstraction $\lambda x.P$ with $x \in FreeVariable(P)$, and (4) when the $\lambda$-term is an abstraction $\lambda x.P$ with $x \notin FreeVariable(P)$. The four cases are distinguished in the predicate `pta/5` which implements the algorithm by recursively analyzing the structure of the tree which represents the input $\lambda$-term. The five arguments of the predicate are as follows:

1. The $\lambda$-term $M$ which we want to assign a type.
2. The $\mathrm{TA}_\lambda$-deduction for the $\lambda$-term $M$.
3. The result of the algorithm, i.e. the statement which says that the $\lambda$-term $M$ is typable or not.
4. The list of variables used during deduction.
5. The tree which represents the deduction.

The case distinction happens in the first argument. For example, we represent an abstraction $\lambda x.M$ as `abstraction(termVar(X),M)`. Then if the $\lambda$-term in the current recursion has that form, then we apply the algorithm for an abstraction.

### Improvement on the Principal-Type Algorithm

In our implementation, we make some modification to improve the performance of the principal-type algorithm given in [4]. The improvement is obtained by combining case II and case III. This modification eliminates the need for checking whether an abstraction variable is free in the abstraction body or not.

## 3 The Type Inhabitant Search Algorithm Implementation

The type inhabitant search algorithm is a core procedure for answering the question "given a type $\tau$, how many *inhabitants* (closed $\lambda$-terms in $\beta$-normal form) can receive $\tau$ in $\mathrm{TA}_\lambda$?" [8, 4]. This algorithm can be used as a tester whether the number of closed $\lambda$-terms that can receive a certain type $\tau$ is zero or not. As a consequence, by the Curry–Howard correspondence, it can be used to test whether a certain implicational formula in intuitionistic logic is provable or not. The algorithm takes a type as input and searches for closed $\lambda$-terms in $\beta$-normal form that can receive the given type in $\mathrm{TA}_\lambda$ through several steps. For example, given

$$\tau \equiv (((c \to d) \to c \to a) \to b) \to (d \to a) \to b$$

as input, the algorithm will produce the following sets:

$$\mathscr{A}(\tau,0) = \{V^\tau\},$$

$$\mathscr{A}(\tau,1) = \{\lambda x_1^{((c\to d)\to c\to a)\to b} x_2^{d\to a}.x_1^{((c\to d)\to c\to a)\to b} V_1^{(c\to d)\to c\to a}\},$$

$$\mathscr{A}(\tau,2) = \{\lambda x_1^{((c\to d)\to c\to a)\to b} x_2^{d\to a}.x_1^{((c\to d)\to c\to a)\to b} (\lambda x_3^{c\to d} x_4^{c}.x_2^{d\to a} V_2^{d})\},$$

$$\mathscr{A}(\tau,3) = \{\lambda x_1^{((c\to d)\to c\to a)\to b} x_2^{d\to a}.x_1^{((c\to d)\to c\to a)\to b} (\lambda x_3^{c\to d} x_4^{c}.x_2^{d\to a} (x_3^{c\to d} V_3^{c}))\},$$

$$\mathscr{A}(\tau,4) = \{\lambda x_1^{((c\to d)\to c\to a)\to b} x_2^{d\to a}.x_1^{((c\to d)\to c\to a)\to b} (\lambda x_3^{c\to d} x_4^{c}.x_2^{d\to a} (x_3^{c\to d} x_4^{c}))\}$$

as it searches for closed $\lambda$-terms in $\beta$-normal form. The $\lambda$-terms are written here in full typed-notation where each term variable is decorated with its type explicitly at the superscript position. The sequence of $\mathscr{A}$-sets shows the approximation from initially unknown $\lambda$-terms (containing meta-variables $V, V_1, V_2$, etc.) toward closed $\lambda$-terms in $\beta$-normal form. In the example above, type $\tau$ has the $\lambda$-term

$$\lambda x_1^{((c\to d)\to c\to a)\to b} x_2^{d\to a}.x_1^{((c\to d)\to c\to a)\to b} (\lambda x_3^{c\to d} x_4^{c}.x_2^{d\to a} (x_3^{c\to d} x_4^{c}))$$

as its inhabitant. By the Curry–Howard correspondence, the existence of this inhabitant means that the formula

$$(((c \to d) \to c \to a) \to b) \to (d \to a) \to b$$

is valid in intuitionistic logic.

In our implementation, we use definite clause grammars in Prolog for parsing the input as a type expression. The parser constructs a tree that represents the given type expression. If it is valid, the program will run the type inhabitant search algorithm to search for its inhabitants. In general, the process is described by the flowchart in Figure 2.

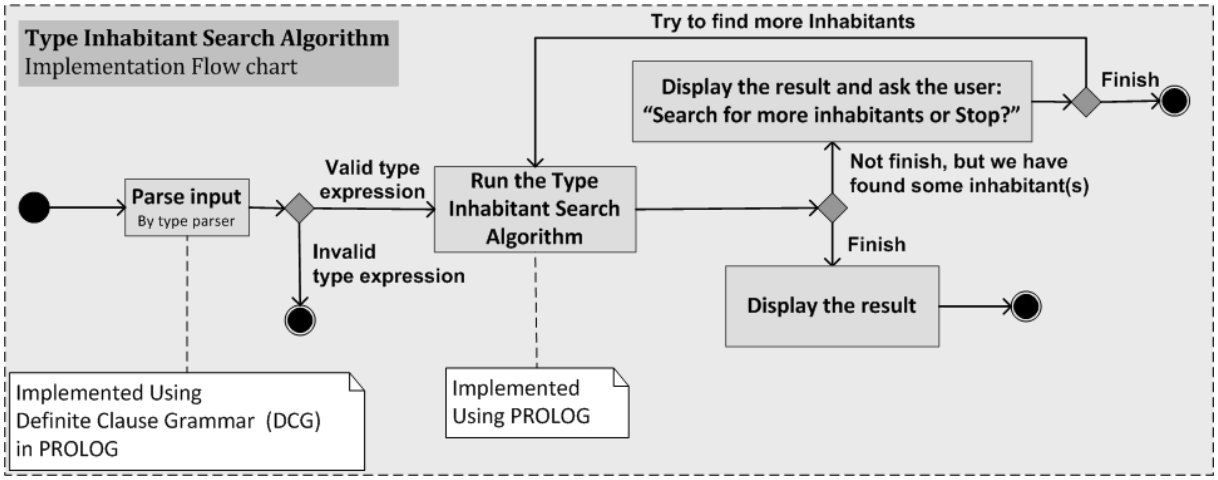



Figure 2: Flowchart for the Implementation of the Type Inhabitant Search Algorithm

The main idea of the type inhabitant search algorithm is to iterate through the approximation of the $\lambda$-term and to produce $\mathscr{A}$-sets with certain depth ($\mathscr{A}(\tau,0)$, $\mathscr{A}(\tau,1)$, ..., $\mathscr{A}(\tau,n)$) during the iteration. In this implementation, we model the iteration as recursion which constructs the $\mathscr{A}$-sets. We define a predicate `searchIhbt/3` with the following arguments:

1. an $\mathscr{A}$-set, i.e. $\mathscr{A}(\tau,d)$ for a certain type $\tau$ and depth $d$.
2. the list of previous $\mathscr{A}$-sets, i.e. the list of $\mathscr{A}(\tau,t)$ with $t = 0,1,\ldots,d-1$.
3. the list of all $\mathscr{A}$-sets that we get during the iteration (searching).

# 4 User Interface

To help users interact with the implementation of those two algorithms, we provide a user-friendly GUI (graphical user interface). The design of our user interface follows the golden rules in designing user interface [7]. Figure 3 and Figure 4 show an overview of our GUI design.

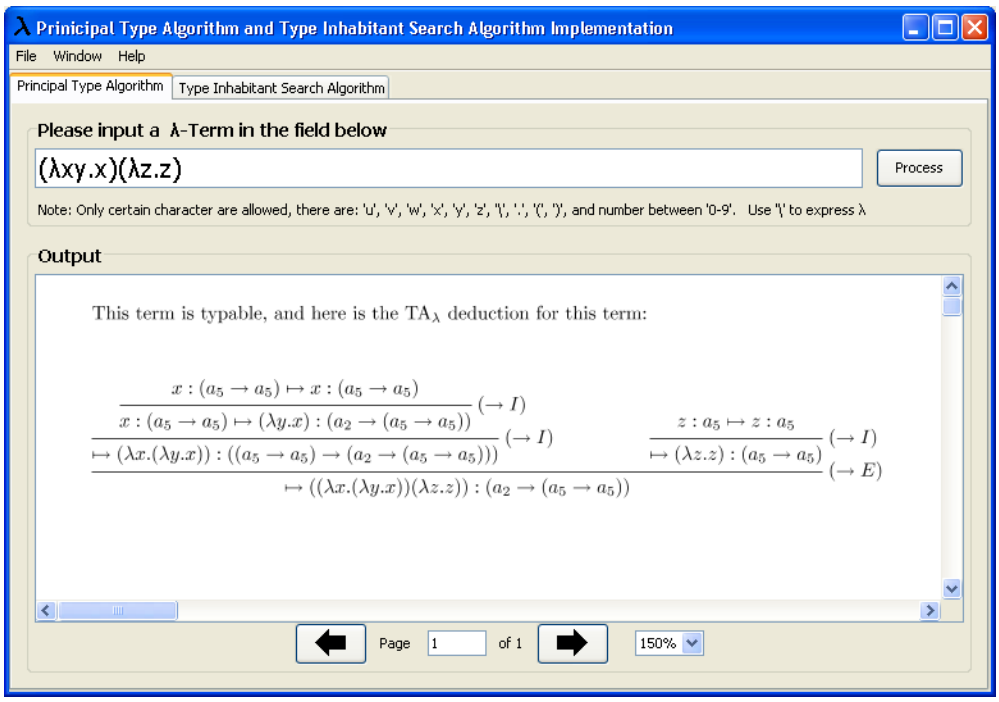



Figure 3: Overview of GUI Design for the Principal-Type Algorithm

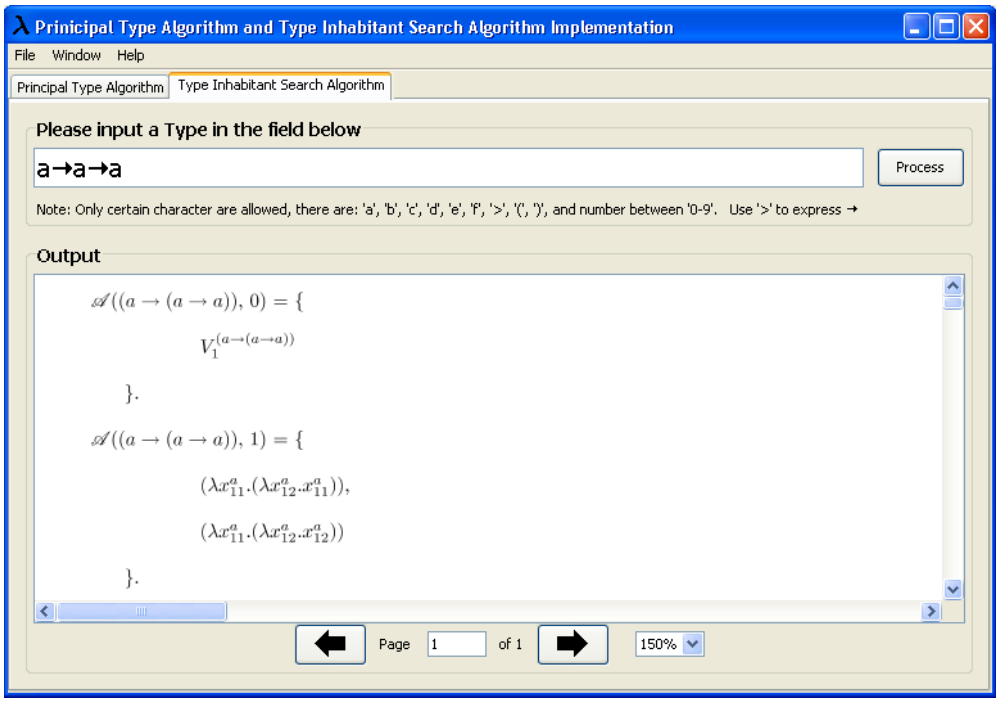



Figure 4: Overview of GUI Design for the Type Inhabitant Search Algorithm

The user interface consists of two main tabs: Principal-Type Algorithm tab and Type Inhabitant Search Algorithm tab. In the Principal-Type Algorithm tab, there is a textbox where user can input a $\lambda$-term and a panel where the type deduction is shown. In the Type Inhabitant Search Algorithm tab,

there is a textbox where user can input a type and a panel where the algorithm result is shown.

This GUI is implemented using Java with Prolog in the back end. We also use LaTeX to help generating tidy output from the algorithms, i.e. deduction trees and steps of type inhabitants searching. In general, the program works as follows. The Java program takes the user input and calls the Prolog programs implementing the algorithms declaratively. The Prolog programs generate the output of the algorithms as LaTeX code. After the algorithms finish execution, the Java program calls LaTeX to compile the algorithm output into a PDF file. Lastly, the Java program reads the generated PDF file and displays it to the user as output.

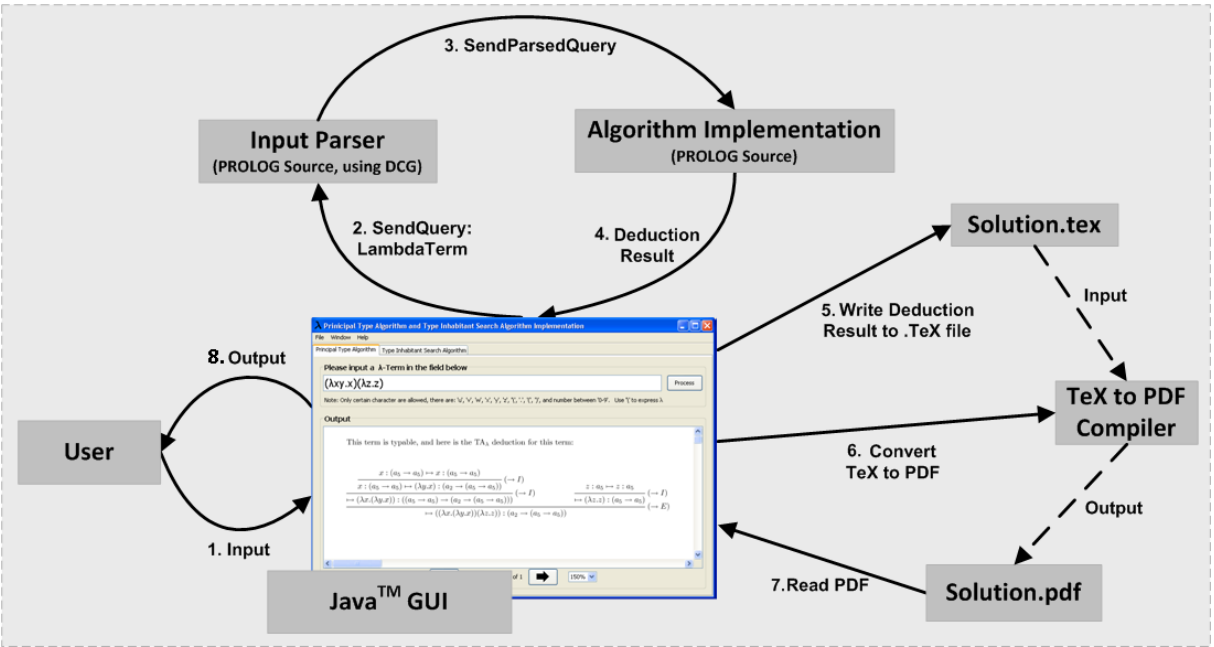



Figure 5: Illustration of the Scenario for the Principal-Type Algorithm Program

The flowchart of this architecture is shown in Figure 5 and the explanation is as follows:

1. User writes a $\lambda$-term into the input textbox.
2. The Java GUI program parses the input $\lambda$-term using the parser implemented in Prolog.
3. The parser passes the parsed $\lambda$-term to the program implementing the principal-type algorithm.
4. The Java GUI program gets back the deduction result for the given $\lambda$-term.
5. The Java GUI program writes the deduction in LaTeX format into a file.
6. The Java GUI program calls the LaTeX program to convert the LaTeX file into a PDF file.
7. The Java GUI program reads the PDF file.
8. The Java GUI program displays the solution to the user.

For the program of the type inhabitant search algorithm implementation, the flowchart is similar. The difference is only on the parser and the main Prolog program for computing the final result.

As a package, this GUI-based implementation of the principal-type algorithm and the type inhabitant search algorithm can play the role of a proof tool that provides assistance in constructing deductions in type theory and establishing the validity of implicational intuitionistic-logic formulas. The intuitive interface helps people study and understand the two algorithms and the related theories.

A web-based version of the software, called $\lambda$-Guru, has also been built. It can be accessed at **http://fmse.cs.ui.ac.id/LambdaGuru**.

Generally, to construct the $\lambda$-Guru, we convert the Java GUI program, which is developed under the Java Swing library, into a JApplet. We also create a PHP program which accepts HTTP POST requests from the JApplet. This PHP program then calls the Prolog program which parses the input and applies the principal-type algorithm or the type inhabitant search algorithm. After the PHP program gets the result back from the Prolog program, it then forwards the algorithm result to the JApplet program. Finally, the JApplet program displays the result to the user. The flowchart of this process is shown in Figure 6.

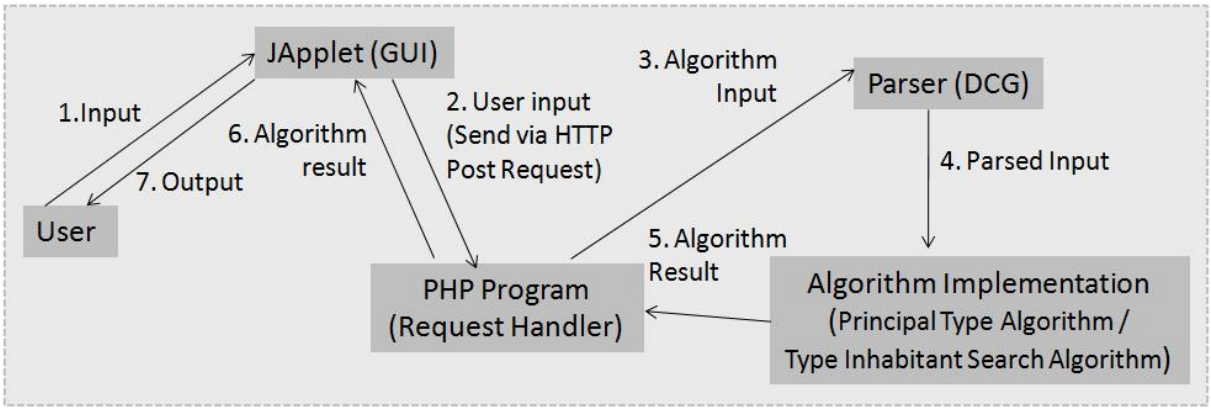



Figure 6: $\lambda$-Guru Web-based Architecture

## 5 Conclusion and Future Work

We have presented a Prolog-based implementation of a proof tool that provides assistance in constructing deductions in type theory and establishing the validity of implicational intuitionistic-logic formulas. With this functionality, this proof tool is useful for learning type theory. This proof tool focuses on an important variant of type theory named $\mathrm{TA}_\lambda$, especially on its two core algorithms: the principal-type algorithm and the type inhabitant search algorithm. We have elaborated on how to implement those two algorithms declaratively in Prolog and the overall GUI-based program architecture. We have also described the graphical user interface that can help users interact with the program. Furthermore, we have built a web-based version of the proof tool called $\lambda$-Guru.

The next step of this work is to implement declaratively the converse principal-type algorithm. It is shown in [4] that if a type $\tau$ is assignable to a closed $\lambda$-term $M$ but is not the principal type of $M$, then it is the principal type of another closed $\lambda$-term $M^*$. The task of the converse principal-type algorithm is to construct $M^*$ when $\tau$ and $M$ are given.

## Acknowledgments

The authors would like to thank the anonymous reviewers of this paper for their helpful comments and suggestions.

# References

[1] A. N. Whitehead and B. Russell. *Principia Mathematica*. Cambridge University Press, Cambridge, revised edition, 1925-1927.

[2] L. Sterling and E. Shapiro. *The Art of PROLOG*. MIT Press, Massachusetts, second edition, 1994.

[3] J. R. Hindley. *The Principal Type-Scheme of an Object in Combinatory Logic*. Trans. American Math. Soc., 1969.

[4] J. R. Hindley. *Basic Simple Type Theory*. Cambridge University Press, Cambridge, first edition, 1997.

[5] C. Munoz. Type theory and its applications to computer science. *Quarterly News Letter of Institute for Computer Application in Science and Engineering (ICASE), Vol 8, No 4*, 2007.

[6] Benjamin C. Pierce. *Types and Programming Languages*. The MIT Press, Cambridge, Massachusetts, 2002.

[7] B. Shneiderman and C. Plaisant. *Designing The User Interface*. Pearson Education, fourth edition, 2005.

[8] B. Yelles. Type-assignment in the lambda-calculus; syntax and semantic. Technical report, Mathemathics Dept. University of Wales Swansea, Swansea SA2 8PP, UK, 1979.

# On Implementing Modular Complexity Analysis*

Harald Zankl and Martin Korp
Institute of Computer Science, University of Innsbruck, 6020 Innsbruck, Austria

**Abstract**

We recall the recent approach by (Zankl and Korp, 2010) to prove upper bounds on the (derivational) complexity of term rewrite systems modularly. In this note we show that this approach is suitable to tighten bounds after they have been established. The idea is to replace proof steps with a large bound by (new) proofs that yield smaller bounds. An evaluation of the approach shows the benefits.

## 1 Introduction

Term rewriting is a Turing-complete model of computation. The objects are modeled as terms and computation is mimicked by reduction steps. For terminating rewrite systems Hofbauer and Lautemann [4] consider the length of derivations as a measurement for the complexity of rewrite systems. The resulting notion of *derivational complexity* relates the length of a rewrite sequence to the size of its starting term. Thereby it is, e.g., a suitable metric for the complexity of deciding the word problem for a given confluent and terminating rewrite system (since the decision procedure rewrites terms to normal form). To show (feasible) upper bounds on the derivational complexity of a rewrite system currently few techniques are known. Match-bounds [2] and arctic matrix interpretations [5] induce linear upper bounds on the derivational complexity and triangular matrix interpretations [7] imply at most polynomially long derivations (the dimension of the matrices yields the degree of the polynomial). Recently in [10] a new approach has been introduced that allows to prove (upper) bounds on the (derivational) complexity of rewrite systems modularly. The idea is based on relative rewriting. In this note we present a method that enables this modular setting to infer tighter bounds.

## 2 Modular Complexity Analysis

We assume familiarity with (relative) term rewriting [1,3,9]. Let $\mathcal{F}$ be a signature and $\mathcal{V}$ a disjoint set of variables. By $\mathcal{T}(\mathcal{F},\mathcal{V})$ we denote the set of terms over $\mathcal{F}$ and $\mathcal{V}$. The size of a term $t$ is denoted $|t|$. A *rewrite rule* is a pair of terms $(l,r)$, written $l \to r$, such that $l$ is not a variable and all variables in $r$ are contained in $l$. A *term rewrite system* (TRS for short) is a set of rewrite rules. For complexity analysis we assume TRSs to be finite and terminating. A TRS $\mathcal{R}$ is said to be *duplicating* if there exists a rewrite rule $l \to r \in \mathcal{R}$ and a variable $x$ that occurs more often in $r$ than in $l$. A *rewrite relation* is a binary relation on terms that is closed under contexts and substitutions. For a TRS $\mathcal{R}$ we define $\to_{\mathcal{R}}$ to be the smallest rewrite relation that contains $\mathcal{R}$. As usual $\to^*$ denotes the reflexive and transitive closure of $\to$ and $\to^n$ the $n$-th iterate of $\to$. A *relative* TRS $\mathcal{R}/\mathcal{S}$ is a pair of TRSs $\mathcal{R}$ and $\mathcal{S}$ with the induced rewrite relation $\to_{\mathcal{R}/\mathcal{S}} = \to_{\mathcal{S}}^* \cdot \to_{\mathcal{R}} \cdot \to_{\mathcal{S}}^*$. In the sequel we will sometimes identify a TRS $\mathcal{R}$ with the relative TRS $\mathcal{R}/\varnothing$ and vice versa.

The *derivation height* of a term $t$ with respect to a relation $\to$ is defined as follows: $\mathsf{dh}(t,\to) = \mathsf{sup}\{m \mid \exists u\ t \to^m u\}$. The *derivational complexity* of a relation $\to$ computes the maximal derivation height of all terms up to size $n$ and is defined as $\mathsf{dc}(n,\to) = \mathsf{sup}\{\mathsf{dh}(t,\to) \mid t \in \mathcal{T}(\mathcal{F},\mathcal{V}) \text{ and } |t| \leqslant n\}$. Sometimes we say that a TRS $\mathcal{R}$ (relative TRS $\mathcal{R}/\mathcal{S}$) has linear, quadratic, etc. or polynomial derivational complexity if $\mathsf{dc}(n,\to_{\mathcal{R}})$ ($\mathsf{dc}(n,\to_{\mathcal{R}/\mathcal{S}})$) can be bounded by a linear, quadratic, etc. function or

* This research is supported by FWF (Austrian Science Fund) project P22467.

polynomial in $n$. For functions $f,g\colon \mathbb{N}\to\mathbb{N}$ we write $f(n)=\mathcal{O}(g(n))$ if there are constants $M,N\in\mathbb{N}$ such that $f(n)\leqslant M\cdot g(n)+N$ for all $n\in\mathbb{N}$. Finally, we use $\mathsf{zero}$ to denote the constant zero function, i.e., $\mathsf{zero}\colon \mathbb{N}\to\mathbb{N}$ with $\mathsf{zero}(n)=0$

Next we recall the recent approach for modular complexity analysis via relative rewriting [10].

**Definition 1.** A *complexity pair* $(\succ,\succeq)$ consists of two finitely branching rewrite relations $\succ$ and $\succeq$ that are *compatible*, i.e., $\succeq\cdot\succ\subseteq\succ$ and $\succ\cdot\succeq\subseteq\succ$. We call a relative TRS $\mathcal{R}/\mathcal{S}$ *compatible* with a complexity pair $(\succ,\succeq)$ if $\mathcal{R}\subseteq\succ$ and $\mathcal{S}\subseteq\succeq$.

The next lemma states that we can use complexity pairs to measure derivational complexity.

**Lemma 2.** *Let $\mathcal{R}/\mathcal{S}$ be a relative TRS compatible with a complexity pair $(\succ,\succeq)$. Then for any term $t$ we have $\mathsf{dh}(t,\succ)\geqslant\mathsf{dh}(t,\to_{\mathcal{R}/\mathcal{S}})$.*

Because we are especially interested in feasible upper bounds we state the next corollary.

**Corollary 3.** *Let $\mathcal{R}/\mathcal{S}$ be a terminating relative TRS compatible with a complexity pair $(\succ,\succeq)$. If the derivational complexity of $\succ$ is linear, quadratic, etc. or polynomial then the derivational complexity of $\mathcal{R}/\mathcal{S}$ is linear, quadratic, etc. or polynomial.*

This corollary allows to investigate the complexity of (compatible) complexity pairs instead of the complexity of the underlying relative TRS. The next theorem is the key observation from [10].

**Theorem 4.** *Let $(\mathcal{R}_1\cup\mathcal{R}_2)/\mathcal{S}$ be a terminating relative TRS. Then the equality $\mathsf{dc}(n,\to_{(\mathcal{R}_1\cup\mathcal{R}_2)/\mathcal{S}})=\mathcal{O}(\mathsf{dc}(n,\to_{\mathcal{R}_1/(\mathcal{R}_2\cup\mathcal{S})})+\mathsf{dc}(n,\to_{\mathcal{R}_2/(\mathcal{R}_1\cup\mathcal{S})}))$ holds.*

Theorem 4 allows to split a relative TRS $(\mathcal{R}_1\cup\mathcal{R}_2)/\mathcal{S}$ into *smaller* components $\mathcal{R}_1/(\mathcal{R}_2\cup\mathcal{S})$ and $\mathcal{R}_2/(\mathcal{R}_1\cup\mathcal{S})$ and evaluate the complexities of these components (e.g., by different complexity pairs) independently. Note that this approach is not restricted to relative rewriting. To estimate the complexity of a (non-relative) TRS $\mathcal{R}$ just consider the relative TRS $\mathcal{R}/\varnothing$. Proofs in the modular setting can be viewed as trees, as shown in the next example.

**Example 5.** Consider the complexity proof in Figure 1b on page 4, for the abstract TRS $\mathcal{R}=\{1,2,3,4,5\}$ consisting of five rewrite rules. The root node of the tree is the TRS of interest and the other nodes are relative rewrite systems and represent intermediate complexity problems. The edges indicate the (derivational) complexity of the proof steps. It is possible to apply Theorem 4 explicitly to split a problem into two (or more) problems as demonstrated in the second node. Such situations do not effect the complexity of the given problem which justifies the labels $\mathcal{O}(1)$. The remaining proof steps rely on an implicit application of Theorem 4 and measure the complexity of the rewrite rules that are moved from the first into the second component (relative to the remaining rules). For instance in the proof shown in Figure 1b there is an edge from $\{1,3,5\}/\{2,4\}$ to $\{1\}/\{2,3,4,5\}$ labeled $\mathcal{O}(n^3)$, stating that the (derivational) complexity of $\{3,5\}/\{1,2,4\}$ is at most cubic. This step is sound because Theorem 4 states that computing an upper bound on $\{1\}/\{2,3,4,5\}$ and $\{3,5\}/\{1,2,4\}$ suffices to get an upper bound on $\{1,3,5\}/\{2,4\}$. Since the leaves in the tree give rise to constant complexity, the complexity of the original problem can be overestimated by summing up the complexities annotated to the edges; yielding a cubic upper bound for the proof in Figure 1b.

In the next section we revisit this example and show how existing bounds can be tightened.

# 3 Implementation

First we present the procedure for obtaining *some* complexity proof. Afterwards Section 3.2 is concerned with tightening the bounds starting from an existing complexity proof.

### 3.1 Establishing Bounds

To estimate the complexity of a TRS $\mathcal{R}$ we first transform $\mathcal{R}$ into the relative TRS $\mathcal{R}/\varnothing$. Obviously $\mathsf{dc}(n, \to_{\mathcal{R}}) = \mathsf{dc}(n, \to_{\mathcal{R}/\varnothing})$. If the input already is a relative TRS this step is omitted. Afterwards for a relative TRS $\mathcal{R}/\mathcal{S}$, we try to establish a bound on the complexity of $\mathcal{R}_1/(\mathcal{R}_2 \cup \mathcal{S})$ and continue with the relative TRS $\mathcal{R}_2/(\mathcal{R}_1 \cup \mathcal{S})$. Here $\mathcal{R}_1$ and $\mathcal{R}_2$ form a decomposition of $\mathcal{R}$. This process is repeated until the remaining problem equals $\varnothing/(\mathcal{R} \cup \mathcal{S})$. Finally the complexity of $\mathcal{R}/\mathcal{S}$ is obtained by summing up all intermediate bounds. In order to establish a maximal number of complexity proofs we run all techniques that can be derived from Corollary 3 (cf. also Theorem 7) in parallel and the first technique that can shift some rules is used to achieve progress.

Note that the procedure sketched above contains an implicit application of Theorem 4, i.e., some method immediately proves a bound for $\mathcal{R}_1/(\mathcal{R}_2 \cup \mathcal{S})$ and leaves $\mathcal{R}_2/(\mathcal{R}_1 \cup \mathcal{S})$ as open proof obligation. In contrast to an explicit application of Theorem 4, here the method that establishes the bound on $\mathcal{R}_1/(\mathcal{R}_2 \cup \mathcal{S})$ can select the decomposition of $\mathcal{R}$ into $\mathcal{R}_1$ and $\mathcal{R}_2$ which is beneficial for performance. As an immediate consequence, proof trees degenerate to lists (see Figure 1a on page 4).

In the following we describe the presented approach more formal and refer to it as the *complexity framework*. In this context we call a relative TRS $\mathcal{R}/\mathcal{S}$ *complexity problem* (CP problem for short). To operate on CP problems so called *complexity processors* are used.

**Definition 6.** A *complexity processor* (CP processor for short) is a function that takes a CP problem $\mathcal{R}/\mathcal{S}$ as input and returns a set of pairs $\bigcup_{1 \leqslant i \leqslant m} \{(\mathcal{R}_i/\mathcal{S}_i, f_i)\}$ as output. Here $\mathcal{R}_i/\mathcal{S}_i$ is a complexity problem and $f_i \colon \mathbb{N} \to \mathbb{N}$ for each $1 \leqslant i \leqslant m$. A complexity processor is *sound* if the equality $\mathsf{dc}(n, \mathcal{R}/\mathcal{S}) = \mathcal{O}(f_1(n) + \cdots + f_m(n) + \mathsf{dc}(n, \to_{\mathcal{R}_1/\mathcal{S}_1}) + \cdots + \mathsf{dc}(n, \to_{\mathcal{R}_m/\mathcal{S}_m}))$ holds.

Next we list some CP processors. The first one is based on complexity pairs. In [10] powerful methods are presented that yield complexity pairs and allow to implement this processor.

**Theorem 7.** *The CP processor that maps the CP problem $\mathcal{R}/\mathcal{S}$ to $\{(\mathcal{R}_1/(\mathcal{R}_2 \cup \mathcal{S}), f)\}$ if $\mathcal{R}_2/(\mathcal{R}_1 \cup \mathcal{S})$ is compatible with a complexity pair $(\succ, \succeq)$ and to $\{(\mathcal{R}/\mathcal{S}, \mathsf{zero})\}$ otherwise is sound. Here $\mathcal{R} = \mathcal{R}_1 \cup \mathcal{R}_2$ and $f(n) = \mathsf{dc}(n, \succ)$.*

The next CP processor is not implemented explicitly (cf. the discussion at the beginning of the section) but very suitable to tighten existing bounds (see Section 3.2).

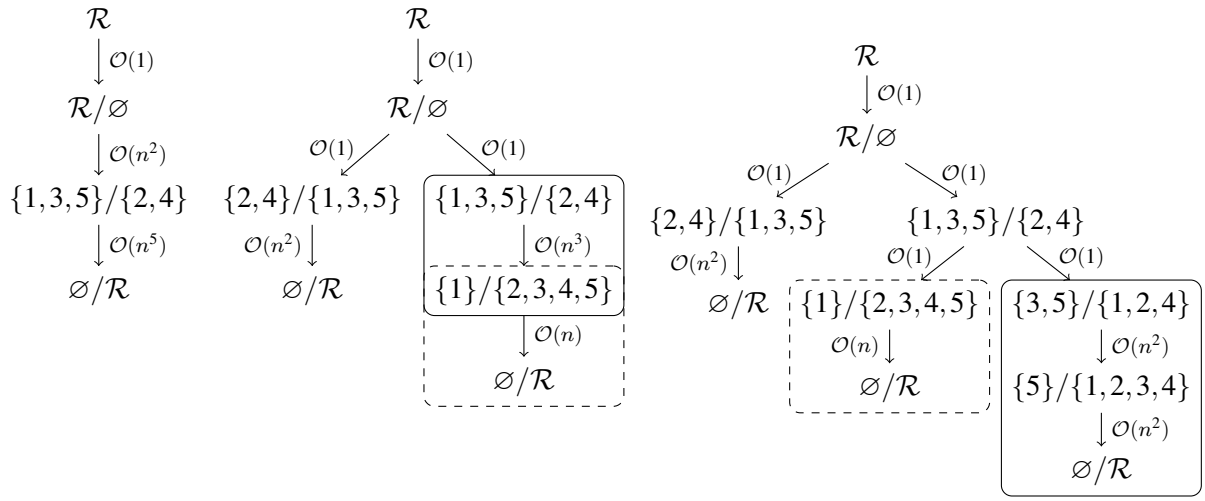
**Theorem 8.** *The CP processor $(\mathcal{R}_1 \cup \mathcal{R}_2)/\mathcal{S} \mapsto \{(\mathcal{R}_1/(\mathcal{R}_2 \cup \mathcal{S}), \mathsf{zero}), (\mathcal{R}_2/(\mathcal{R}_1 \cup \mathcal{S}), \mathsf{zero})\}$ is sound.*

Further CP processors are presented in [10]. They are not essential for our purposes here. Finally, the main theorem states that the complexity framework is suitable for complexity analysis. We say that $P$ is a complexity proof for a relative TRS $\mathcal{R}/\mathcal{S}$ if all leaves in the proof are of the shape $\varnothing/(\mathcal{R} \cup \mathcal{S})$.

**Theorem 9.** *Let $\mathcal{R}/\mathcal{S}$ be a relative TRS, $P$ a complexity proof for $\mathcal{R}/\mathcal{S}$, and $f_1, \ldots, f_m$ the complexities occurring in $P$. If all CP processors in $P$ are sound then $\mathsf{dc}(n, \to_{\mathcal{R}/\mathcal{S}}) = \mathcal{O}(f_1(n) + \cdots + f_m(n))$.*

### 3.2 Tightening Bounds

In contrast to termination, which is a plain YES/NO question, complexity corresponds to an optimization problem. Hence the tools should try to establish as tight bounds as possible. In the direct setting (where all base methods are used stand-alone) all methods can be executed in parallel and after a fixed amount of time the tightest bound is reported. In the modular setting this simple idea does not work because two problems emerge. The first problem is that the tool does not know how much time it may spend on a single proof step. If it spends too much then it may not finish the proof within the global time limit and




(a) Original proof (b) Optimized proof (first step) (c) Optimized proof (second step)

Figure 1: Tightening bounds

if it spends too little then it can miss a low bound. The second problem is that in the modular setting (according to Theorem 4) separate criteria may make statements about the complexity of different rules. The question is then to identify the *better* bound. The following idea overcomes both problems: First we establish *some* complexity proof according to the procedure described at the beginning of Section 3.1 to obtain a bound for as many systems as possible. Afterwards we *optimize* this bound. The next example shows how the latter works.

**Example 10.** In Figure 1, three complexity proofs for the TRS $\mathcal{R}$ of Example 5 are given. We show how the proof in Figure 1b can be optimized to the one in Figure 1c on an abstract level (note that a similar reasoning has been applied to obtain the proof shown in Figure 1b). In Figure 1b one part in the proof, highlighted by a solid box, is overestimated by a cubic upper bound. Hence the complexity of the whole system is at most cubic. We remark that this proof step estimates the complexity of $\{3,5\}/\{1,2,4\}$. Now assume that the cubic bound is not optimal, i.e., there exists a proof (that may be longer and harder to find) that induces a quadratic upper bound on the complexity of $\{3,5\}/\{1,2,4\}$. Then the proof can be optimized by splitting $\{1,3,5\}/\{2,4\}$ into the problems $\{1\}/\{2,3,4,5\}$ and $\{3,5\}/\{1,2,4\}$ as illustrated in Figure 1c. After that, the proof of $\{1\}/\{2,3,4,5\}$ is reused in the optimized proof (cf. the dashed boxes in Figure 1b and Figure 1c) whereas the original proof of $\{3,5\}/\{1,2,4\}$ is replaced by the new one, as indicated by the solid box in Figure 1c. Now, the proof in Figure 1c establishes a quadratic upper bound on the complexity of $\mathcal{R}$.

As the previous example demonstrates the basic idea is to replace single proof steps by new proofs that induce tighter bounds. This procedure is repeated until either the global time limit is reached or none of the bounds can be tightened further. Note that the transformation is sound by Theorem 9.

# 4 Experimental Results

The technique described in the preceding section is implemented in the complexity analyzer GaT [10] which is developed by the authors. Below we report on the experiments we performed on the 1172 non-duplicating TRSs in version 7.0.2 of the termination problems database without strategy or theory annotation. Note that duplicating systems induce at least exponential derivational complexity. All tests

Table 1: Derivational complexity of 1172 TRSs

| | $\mathcal{O}(n^k)$ | $\mathcal{O}(n)$ | $\mathcal{O}(n^2)$ | $\mathcal{O}(n^3)$ | time |
|---|---|---|---|---|---|
| modular | 334/334 | 208/221 | 228/321 | 261/329 | 2.6/10.6 |
| CaT | 328/328 | 216/219 | 310/317 | 319/324 | 4.2/11.5 |

have been performed on a server equipped with eight dual-core AMD Opteron® processors 885 running at a clock rate of 2.6 GHz and on 64 GB of main memory. If the tool did not report an answer within 60 seconds, its execution was aborted.

Our results are summarized in Table 1. As base methods we use the match-bounds technique [6,10] as well as polynomially bounded matrix interpretations [7,5,8] of dimensions one to five. The row modular refers to the implementation as in [10] where all base methods are run in parallel and started upon program execution. For reference we also list the data for the 2010 competition version of CaT, which was the winner in the derivational complexity category. In the table the columns $\mathcal{O}(n)$, $\mathcal{O}(n^2)$, etc. give the number of TRSs whose derivational complexity could be shown to be at most linear, quadratic, etc. We also list the total number of TRSs for which a polynomial bound could be established (cf. column $\mathcal{O}(n^k)$) and the average time (in seconds) needed for finding a bound. The numbers before (after) the slash correspond to the setting which does not (does) make use of tightening bounds (cf. Section 3.2).

The results in the table indicate that refining bounds is beneficial, especially if all criteria are started synchronously, which is essential to maximize the total number of upper bounds. The 2010 version of CaT did not use tightening of bounds. To maximize the number of low bounds CaT executes criteria that yield larger complexity bounds slightly delayed. This explains why for CaT tightening bounds increases the global performance less compared to modular. On the contrary, CaT misses some proofs compared to modular since (costly) criteria are not executed for up to 60 seconds.

For further comparison with other tools we refer the reader to the international termination competition (`http://termcomp.uibk.ac.at`). Since 2008, when the complexity categories have been installed in the termination competition, CaT won the division for derivational complexity every year.

# References


[1] Baader, F., Nipkow, T.: Term Rewriting and All That. Cambridge University Press, Cambridge (1998)

[2] Geser, A., Hofbauer, D., Waldmann, J., Zantema, H.: On tree automata that certify termination of left-linear term rewriting systems. I&C 205(4), 512–534 (2007)

[3] Geser, A.: Relative termination. PhD thesis, Universität Passau, Germany (1990). Available as: Report 91-03, Ulmer Informatik-Berichte, Universität Ulm, 1991

[4] Hofbauer, D., Lautemann, C.: Termination proofs and the length of derivations (preliminary version). In: RTA 1989. LNCS, vol. 355, pp. 167–177 (1989)

[5] Koprowski, A., Waldmann, J.: Max/plus tree automata for termination of term rewriting. AC 19(2), 357–392 (2009)

[6] Korp, M., Middeldorp, A.: Match-bounds revisited. I&C 207(11), 1259–1283 (2009)

[7] Moser, G., Schnabl, A., Waldmann, J.: Complexity analysis of term rewriting based on matrix and context dependent interpretations. In: FSTTCS 2008. LIPIcs, vol. 2, pp. 304–315 (2008)

[8] Neurauter, F., Zankl, H., Middeldorp, A.: Revisiting matrix interpretations for polynomial derivational complexity of term rewriting. In: LPAR 17. LNCS ARCoSS, vol. 6397, pp. 550–564 (2010)

[9] TeReSe: Term Rewriting Systems. vol. 55 of Cambridge Tracts in Theoretical Computer Science. Cambridge University Press (2003)

[10] Zankl, H., Korp, M.: Modular complexity analysis via relative complexity. In: RTA 2010. LIPIcs, vol. 6, pp. 385–400 (2010)

# Implementing an Efficient SAT Solver for a Probabilistic Description Logic

Pavel Klinov
University of Manchester
Manchester, United Kingdom
pklinov@cs.man.ac.uk

Bijan Parsia
University of Manchester
Manchester, United Kingdom
bparsia@cs.man.ac.uk

**Abstract**

This paper presents an optimized algorithm for solving the satisfiability problem (PSAT) in the probabilistic description logic P-$\mathcal{SROIQ}$. In P-$\mathcal{SROIQ}$ and related Nilsson-style probabilistic logics the PSAT problem is typically solved by reduction to linear programming. This straightforward approach does not scale well because the number of variables in linear programs grows exponentially with the number of probabilistic statements. In this paper we demonstrate an algorithm to cope with this problem which is based on column generation. Although column generation approaches to PSAT have been known for the last two decades, this is, to the best of our knowledge, the first algorithm which also works for a non-propositional probabilistic logic. We report results of an empirical investigation which show that the algorithm can handle probabilistic knowledge bases of about 1000 probabilistic statements in addition to even larger number of classical $\mathcal{SROIQ}$ axioms.

## 1 Introduction

There are many proposed formalisms for combining Description Logics (DLs) with various sorts of uncertainty, although, to our knowledge, none have been used in a real application. We believe that this is due to two reasons: 1) there is comparatively little knowledge about how to use these formalisms effectively (or even, which are best suited for what purposes) and 2) there is a severe lack of tooling, in particular, there have been no sufficiently effective reasoners.

This paper describes our work on the second problem. We present the satisfiability algorithm implemented in Pronto,[1] our reasoner for the probabilistic extension of DL $\mathcal{SROIQ}$ (named P-$\mathcal{SROIQ}$) [15]. This logic can be viewed either as a generalization of the Nilsson's propositional probabilistic logic [18] or as a fragment of first-order probabilistic logic of Halpern and Bacchus [5] [2] (with certain non-monotonic extensions which are unimportant in the context of this paper). One attractive feature of these probabilistic logics is that they allow modelers to declaratively describe their uncertain knowledge without fully specifying any probability distribution in contrast to, for example, Bayesian networks. Several applications of probabilistic DLs have been described, in particular, automated validation of uncertain ontology alignments [3] and, very recently, an analysis of a large medical expert system CADIAG-2 [14]. In the latter case uncertain rules used in medical diagnosis were translated into P-$\mathcal{SROIQ}$ and Pronto was used to discover *all* probabilistic inconsistencies in the system.

In spite of their attractive features Nilsson-style logics have been criticized, partly for the intractability of probabilistic inference. Reasoning procedures are typically implemented via reduction to linear programming but it is well known that corresponding linear programs are exponentially large so the scalability is very limited. Over the last two decades there have been several attempts to overcome that issue in the propositional case which led to some promising results, such as solving the probabilistic satisfiability problem (PSAT) for 800-1000 formulas

[1] http://www.cs.manchester.ac.uk/%7Eklinovp/research/pronto

[6]. It has been unclear whether the methods used to solve large propositional PSATs can be directly applied to PSAT in probabilistic DLs (see Section 5).

To the best of our knowledge, Pronto is the first reasoner for a Nilsson-style probabilistic DL which scalability is comparable to (and often better than) scalability of propositional solvers. In particular, it can solve propositional PSATs of the same size but i) can also handle probabilistic statements over arbitrary (i.e. non-propositional) $\mathcal{SROIQ}$ expressions and ii) can efficiently deal with KBs containing large bodies of non-probabilistic knowledge *in addition to* roughly 1000 probabilistic statements.

## 2 Preliminaries

This section provides a brief background on the Description Logic $\mathcal{SROIQ}$ and its probabilistic extension P-$\mathcal{SROIQ}$.

### 2.1 Description Logic

Description Logics is a family of logics which are typically decidable fragments of first-order logic developed specifically for representing structural background knowledge [1]. $\mathcal{SROIQ}$ is one of the most expressive representatives of that family. It is a formal basis of the Web Ontology Language (OWL 2) which is a W3C standard for representing ontologies. We introduce DLs using $\mathcal{ALCOQ}$— a subset of $\mathcal{SROIQ}$ which provides all the syntactic features used in this paper. Full presentation of $\mathcal{SROIQ}$ is space consuming and is not required since there are no important differences between probabilistic extensions to $\mathcal{ALCOQ}$ and to $\mathcal{SROIQ}$. We refer to [1, 9] for full details on syntax and semantics of expressive DLs including $\mathcal{SROIQ}$.

**Syntax of $\mathcal{ALCOQ}$** We assume fixed finite sets $N_C, N_R$ and $N_I$ of concept names (atomic concepts), role names and individuals respectively. Concept expressions (or concepts) in $\mathcal{ALCOQ}$ have the following syntactic form:

$$C ::= A|\{o\}|\neg C|C \sqcap D|\exists R.C|\geq nR.C|\leq nR.C \tag{1}$$

where $A \in N_C, R \in N_R$, $C$ and $D$ are concepts, $n$ is a natural number, $o \in N_I$. The following are the standard abbreviations: $C \sqcup D \equiv \neg(\neg C \sqcap \neg D)$, $\forall R.C \equiv \neg\exists R.\neg C$, $\bot \equiv \neg A \sqcap A$, $\top \equiv \neg\bot$, $\{o_1, \ldots, o_k\} \equiv \{o_1\}\sqcup, \ldots, \sqcup\{o_k\}$ and $= nR.C \equiv\geq nR.C\sqcap \leq nR.C$. Expressions of the form $\{o_1, \ldots, o_k\}$ are called nominals.

Usually a knowledge base (or *ontology*) in DLs is considered to be a tuple $\mathcal{K} = (\mathcal{T}, \mathcal{A}, \mathcal{R})$ where $\mathcal{T}$ is a terminological box (TBox), $\mathcal{A}$ is an assertional box (ABox) and $\mathcal{R}$ is a role box (RBox). In this paper RBoxes are irrelevant while nominals make TBoxes strictly more expressive than ABoxes [1]. Therefore we will only consider TBoxes which are sets of *concept subsumption* axioms. Each subsumption axiom is an expression of the form $C \sqsubseteq D$ where $C$ and $D$ are concepts. $C \equiv D$ abbreviates $\{C \sqsubseteq D, D \sqsubseteq C\}$.

**Semantics of $\mathcal{ALCOQ}$** Semantics of DLs is standardly based on interpretations $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$, where $\Delta^{\mathcal{I}}$ is a non-empty set (the domain) and $\cdot^{\mathcal{I}}$ is an interpretation function that maps each $A \in N_C$ to a subset $A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$, each $R \in N_R$ to a relation $R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$ and each $o \in N_I$ to an element $o^{\mathcal{I}} \in \Delta^{\mathcal{I}}$. It is extended to concept expressions as follows:

$$(\neg C)^{\mathcal{I}} = \Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$$
$$(\{o\})^{\mathcal{I}} = o^{\mathcal{I}}$$
$$(C \sqcap D)^{\mathcal{I}} = C^{\mathcal{I}} \cap D^{\mathcal{I}}$$
$$(\exists R.C)^{\mathcal{I}} = \{x \in \Delta^{\mathcal{I}} | \exists y \in \mathcal{C}^{\mathcal{I}} \text{ s.t. } (x,y) \in R^{\mathcal{I}}\}$$
$$(\geq nR.C)^{\mathcal{I}} = \{x \in \Delta^{\mathcal{I}} || \{y \in \mathcal{C}^{\mathcal{I}} \text{ s.t. } (x,y) \in R^{\mathcal{I}}\}| \geq n\}$$
$$(\leq nR.C)^{\mathcal{I}} = \{x \in \Delta^{\mathcal{I}} || \{y \in \mathcal{C}^{\mathcal{I}} \text{ s.t. } (x,y) \in R^{\mathcal{I}}\}| \leq n\}$$

An interpretation $\mathcal{I}$ satisfies (or is a model of) a subsumption axiom $C \sqsubseteq D$ if $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$. It is a model of a TBox $\mathcal{T}$ if it is a model of each axiom in $\mathcal{T}$. A TBox is called satisfiable if it has a model. Given a TBox $\mathcal{T}$ concept $C$ is called satisfiable if there exists an interpretation $\mathcal{I}$ which is a model of $\mathcal{T}$ and $C^{\mathcal{I}} \neq \emptyset$. An axiom $C \sqsubseteq D$ is entailed by a TBox $\mathcal{T}$ if it is satisfied by every model of $\mathcal{T}$.

### 2.2 Probabilistic Description Logic P-$\mathcal{SROIQ}$

P-$\mathcal{SROIQ}$ is a probabilistic generalization of the DL $\mathcal{SROIQ}$ [15]. It supports probabilistic subsumptions between arbitrary $\mathcal{SROIQ}$ concepts and a certain class of probabilistic concept assertions. Any $\mathcal{SROIQ}$ ontology can be used as a basis for a P-$\mathcal{SROIQ}$ ontology which facilitates transition from classical to probabilistic ontologies.

**Syntax of P-$\mathcal{SROIQ}$** The syntactic constructs of P-$\mathcal{SROIQ}$ include those of $\mathcal{SROIQ}$ together with *conditional constraints*. Conditional constraints are expressions of the form $(D|C)[l,u]$ where $D, C$ are $\mathcal{SROIQ}$ concept expressions (called *conclusion* and *evidence* respectively) and $[l,u] \subseteq [0,1]$ is a closed real-valued interval. Unconditional constraints are the special case of conditional ones when the evidence class is equivalent to $\top$.

For the purpose of this paper it is sufficient to consider only probabilistic TBoxes (or PTBoxes). A PTBox is a pair $PT = (\mathcal{T}, \mathcal{P})$ where $\mathcal{T}$ is a classical $\mathcal{SROIQ}$ TBox and $P$ is a finite set of conditional constraints. Informally, a PTBox axiom $(D|C)[l,u]$ means that "if a randomly chosen individual is an instance of $C$, the probability of it being an instance of $D$ is in $[l,u]$". In what follows we call $\mathcal{T}$ and $\mathcal{P}$ the classical and the probabilistic part of a PTBox respectively.

**Semantics of P-$\mathcal{SROIQ}$** Semantics of P-$\mathcal{SROIQ}$ is standardly explained using the notion of *possible world* which is defined with respect to a set of concepts $\Phi$ (called *basic concepts* or *probabilistic signature*[2]) [15]. A possible world $I$ is a subset of $\Phi$ such that the set of axioms $\{\{o\} \sqsubseteq C | C \in I\} \cup \{\{o\} \sqsubseteq \neg C | C \notin I\}$ is satisfiable for a fresh individual $o$ (in other words, possible worlds correspond to *realizable* concept types). A basic concept $C$ occurs *positively* in a possible world $I$ if $C \in I$, otherwise it occurs *negatively*. The set of all possible worlds with respect to $\Phi$ is denoted as $\mathcal{I}_\Phi$. A world $I$ satisfies a basic concept $C$ denoted as $I \models C$ if $C$ occurs positively in $I$. Satisfiability of basic concepts is inductively extended to $\mathcal{SROIQ}$ concept expressions according to the semantics of $\mathcal{SROIQ}$ (see Section 2.1), for example, $I \models C \sqcap D$ if $I \models C$ and $I \models D$.

For the reason which will become clear in Section 3.2 we assume a *linear order* of basic concepts in $\Phi$. Since $\Phi$ is a finite set we can denote the $i$-th basic concept in $\Phi$ by $C_i$. For a

[2]Note that basic concepts need not be atomic.

given possible world $I$ we also use the notation $I_i$ to denote either $C_i$ if $C_i$ occurs positively in $I$ or $\neg C_i$ if it occurs negatively. For a given PTBox the order of basic concepts is fixed across all possible worlds.

A world $I$ is said to be a model of a TBox axiom $\alpha$ denoted as $I \models \alpha$ if $\alpha \cup \{\{o\} \sqsubseteq C | C \in I\} \cup \{\{o\} \sqsubseteq \neg C | C \notin I\}$ is satisfiable for a new individual $o$. A world $I$ is a model of a $\mathcal{SROIQ}$ TBox $\mathcal{T}$ denoted as $I \models \mathcal{T}$ if it is a model of all axioms of $\mathcal{T}$. A world $I$ that satisfies a TBox $\mathcal{T}$ exists iff $\mathcal{T}$ has a model $(\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ [15].

A probabilistic interpretation $Pr$ is a probability distribution over $\mathcal{I}_\Phi$. $Pr$ is said to *satisfy* a $\mathcal{SROIQ}$ TBox $\mathcal{T}$ denoted as $Pr \models \mathcal{T}$ if for all $I \in \mathcal{I}_\Phi, Pr(I) > 0$ implies that $I \models \mathcal{T}$. The probability of a concept $C$, denoted as $Pr(C)$, is defined as $\sum_{I \models C} Pr(I)$. $Pr(D|C)$ is used as an abbreviation for $Pr(C \sqcap D)/Pr(C)$ given $Pr(C) > 0$. A probabilistic interpretation $Pr$ satisfies a conditional constraint $(D|C)[l, u]$, denoted as $Pr \models (D|C)[l, u]$, if $Pr(C) = 0$ or $Pr(D|C) \in [l, u]$. $Pr$ satisfies a set of conditional constraints $F$ if it satisfies each of the constraints. A PTBox $PT = (\mathcal{T}, \mathcal{P})$ is called *satisfiable* if there exists an interpretation that satisfies both $\mathcal{T}$ and $\mathcal{P}$.

The probabilistic satisfiability problem (PSAT) is a problem of deciding if a PTBox $(\mathcal{T}, \mathcal{P})$ has a model $Pr$. It is decidable and its complexity class is `N2ExpTime`-complete, i.e. the same as the complexity of reasoning in $\mathcal{SROIQ}$ [10]. We refer to [15] for a more detailed presentation of P-$\mathcal{SROIQ}$ semantics, reasoning problems and procedures, and complexity results.

## 3 The Probabilistic Satisfiability Algorithm

The first contribution of this paper is the novel PSAT algorithm implemented in Pronto (see Section 5 for some relationships to the previously developed methods). For the sake of clarity we will consider a special case of PSAT where the PTBox is of the form $PT = (\mathcal{T}, \{(C_i|\top)[p_i, p_i]\})$ (i.e. all probabilistic statements are unconditional constraints with point-valued probabilities and all $C_i$ are concept names). It is straightforward, but technically awkward, to generalize the procedure to handle conditional interval statements over arbitrary concept expressions.

A PTBox $PT = (\mathcal{T}, \{(C_i|\top)[p_i, p_i]\})$ is satisfiable iff the following system of linear inequalities is *feasible*, i.e. admits at least one solution (by generalization from propositional PSAT [6]):

$$\begin{aligned} &\sum_{I \models C_i} x_I = p_i, \text{ for each } (C_i|\top)[p_i, p_i] \in \mathcal{P} \\ &\sum_{I \in \mathcal{I}_\Phi} x_I = 1 \text{ and all } x_I \geq 0 \end{aligned} \tag{2}$$

where $\mathcal{I}_\Phi$ is the set of all possible worlds for the set of concepts $\Phi$ in $\mathcal{T}$. Observe, that $\mathcal{I}_\Phi$ is finite but exponential in the size of $\Phi$, so it is not feasible to explicitly generate this system to check whether it admits a solution.

One successful approach to dealing with linear systems having an exponential number variables is *column generation*. It is based on the fundamental property of linear programming: any feasible (i.e. admitting at least one solution) program always has an optimal solution in which only a linear number of variables have non-zero values. Column generation exploits this property by trying to avoid an explicit representation of variables (columns) which will not have positive values in the finally discovered solution. The method is briefly presented in the next subsection.

### 3.1 Column Generation Basics

Consider the standard form of a linear program (3). Any linear program, in particular, a version (2) with intervals can be reduced to it by adding slack variables.

$$\max\ z = cx \tag{3}$$
$$\text{s.t. } Ax = b \tag{4}$$
$$x \geq 0$$

$A$ denotes a $m \times n$ matrix of linear coefficients of (3). At every step of the simplex algorithm, $A$ is represented as a combination $(B, N)$ where $B$ and $N$ are submatrices of the *basic* and *non-basic* variables, respectively. Values of non-basic variables are fixed to zero, and the solver proceeds by replacing one basic variable by a non-basic one until the optimal solution is found. Variables are represented as indexed columns of $A$. The index of a non-basic column which enters the basis is determined according to the following expression [6]:

$$j \in \{1, \ldots, |N|\} \text{ s.t. } c_j - u^T A^j \text{ is maximal} \tag{5}$$

where $c_j$ is the objective coefficient for the new variable and $u^T$ is the current dual solution of (3). The expression $c_j - u^T A^j$ is called *reduced cost*. At every iteration the column having the highest positive reduced cost is selected. If no such column exists the linear program is at an optimum and the simplex algorithm stops.

If the size of $N$ is exponential, as is the case for the program (2), one should compute the index of the entering column according to (5) without examining all columns in $N$. This is done using the column generation technique in which (5) is treated as an optimization problem with the following objective function:

$$max\ (c_j - \sum_{i=1}^{m+1} u_i a_i^j),\ A^j = (a_i^j) \in \{0,1\}^{m+1} \tag{6}$$

where $a_i^j$ are binary variables that represent linear coefficients of the entering column.

It is important to note that except of the way the entering column is obtained (i.e. generated vs. selected) the simplex algorithm works along the same lines. Whether the column generation technique is successful is contingent upon the following criteria: i) there exists an efficient algorithm for the optimization problem (6), ii) an optimal solution of the program (3) can be found without generation of an excessive number of columns. Such number characterizes *convergence* of the algorithm. In the next subsection we demonstrate an optimized algorithm for generating columns for the PSAT system (2) and will later demonstrate its effectiveness.

### 3.2 Possible World Generation

We first rewrite the system (2) as the following linear program:

$$\begin{aligned} \max \ & \sum_{I \in I_\Phi} x_I \\ \text{s.t.} \ & \sum_{I \models C_i} x_I = p_i \times \sum_{I \in I_\Phi} x_I, \text{ for each } (C_i|\top)[p_i, p_i] \in \mathcal{P} \\ & \sum_{I \in I_\Phi} x_I \leq 1 \text{ and all } x_I \geq 0 \end{aligned} \tag{7}$$

This program has the optimal objective value of 1 iff the system (2) is feasible. The advantage of using this program is that it is feasible even if the PTBox is not satisfiable. This property facilitates use of the column generation technique.

Consider $a_i^j$, the $i$-th coefficient of some column $A^j$. The column corresponds to some possible world $I^j = \{C_i\}$, therefore $a_i^j = 1$ implies that $C_i$ occurs positively in $I^j$ while $a_i^j = 0$ implies that it occurs negatively (or equivalently, $I^j \models \neg C_i$). Thus it is possible to represent $I^j$ as a *conjunctive* concept expression in $\mathcal{SROIQ}$ assuming a fixed linear ordering of concept names $\{C_i\}$ in $\Phi$ (see Section 2.2). More formally, we define the following function $\eta$ which maps columns, i.e. binary vectors, to conjunctions of basic concepts from $\Phi$:

$$\eta(A^j) = \bigsqcap X_i, \text{ where } X_i = \begin{cases} C_i, & \text{if } a_i^j = 1 \\ \neg C_i, & \text{if } a_i^j = 0 \end{cases} \tag{8}$$

$X_i$ are literal concepts that denote either a basic concept or its negation.

*Soundness* of the PSAT algorithm strongly depends on whether every solution of the optimization problem (6), which is added as a column to the main linear program (7), corresponds to a concept expression that is satisfiable w.r.t. $\mathcal{T}$, i.e. is a *possible* world. If this condition is true then soundness trivially follows because one may simply enumerate the set of all solutions (since the set of possible worlds is finite), so (7) will be equivalent to the original linear system (2). *Completeness* requires that every possible world for the given PTBox corresponds to some solution of (6). Therefore, for ensuring both soundness and completeness it is crucial to construct a set of constraints $\mathcal{H}$ for the problem (6) such that its set of solutions is in one-to-one correspondence with the set of all possible worlds $\mathcal{I}_\Phi$.

In what follows we will call columns which correspond to satisfiable expressions *valid* and others — invalid. More formally, given a $\mathcal{SROIQ}$ TBox $\mathcal{T}$, a column $A^j$ is valid if $\mathcal{T} \not\models \eta(A^j) \sqsubseteq \bot$ and is invalid otherwise.

Validity can easily be ensured in the propositional case where each $C_i$ is a clause. One possibility is to employ a well known formulation of SAT as a mixed-integer linear program (MILP) [7]. For example, if $C_i = c_1 \vee \neg c_2 \vee c_3$ then (6) will have the constraint $a_i = x_{c1} + (1 - x_{c2}) + x_{c3}$ where all variables are binary. In that case soundness and completeness follow from the reduction of SAT to MILP. Previously developed propositional PSAT algorithms take full advantage of that (see Section 5).

In the case of an expressive language, such as $\mathcal{SROIQ}$, there appears to be no easy way of determining the set of constraints $\mathcal{H}$. Furthermore, it is unclear whether such a set is polynomial in the size of $\mathcal{T}$. Informally, $\mathcal{H}$ must capture every entailment, such as $\mathcal{T} \models C_i \sqcap, \ldots, \sqcap C_j \sqsubseteq \bot$ in order to prevent generation of any column $A^j$ such that $C_i \sqcap, \ldots, \sqcap C_j$ is a conjunctive subexpression of $\eta(A^j)$. All such entailments can be computed in a naive way by checking satisfiability of all conjunctions $C_i \sqcap, \ldots, \sqcap C_j$ over $\Phi$ but this is no better than trying to construct the full linear system (2).

Instead, Pronto implements a novel *hybrid*, *iterative* procedure to compute $\mathcal{H}$ which can be summarized as follows:

```
   Input: PTBox PT = (T, P), current dual solution u^T of (7)
   Output: New column A^j or null
 1 Initialize (6) using u^T, H ← ∅
 2 while A^j ≠ null do
 3 |   Solve (6) to optimality, A^j ← next optimal solution
 4 |   if A^j ≠ null then
 5 |   |   if satisfiable(η(A^j), T) then
 6 |   |   |   return A^j
 7 |   |   else
 8 |   |   |   add constraints to H that block A^j
 9 |   |   end
10 |   end
11 end
12 return null;
```

**Algorithm 1**: Possible world generation algorithm

The key steps are 5 and 8. On step 5 the algorithm invokes a $\mathcal{SROIQ}$ reasoner (in our case, Pellet [21]) to determine if the computed column corresponds to a *possible* world. This is critical for soundness. If yes, the column is valid and returned. If no, the current set of constraints $\mathcal{H}$ needs to be extended to exclude $A^j$ from the set of solutions to (6) (a similar technique is used in All-SAT solvers to block clauses [17]). This step deserves a more detailed explanation which we present by first defining the notion of the minimal unsatisfiable core for an unsatisfiable conjunctive concept expression.

**Definition 1** (Unsatisfiable Core)**.** *Given a TBox $\mathcal{T}$ and unsatisfiable (w.r.t. $\mathcal{T}$) concept expression $\bigsqcap X_i$ represented as a set of conjuncts $X = \{X_i\}$, a minimal unsatisfiable subexpression (MUS) is a subset $X' = \{X'_i\} \subseteq \{X_i\}$ such that $\bigsqcap X'_i$ is unsatisfiable w.r.t. $\mathcal{T}$ and any $X'' = \{X''_i\} \subset \{X'_i\}$ is satisfiable w.r.t. $\mathcal{T}$. The unsatisfiable core (UC) of $\bigsqcap X_i$ is the set of all its MUSes.*

Intuitively, our notion of UC for conjunctive $\mathcal{SROIQ}$ concepts corresponds to the standard notion of unsatisfiable core for propositional formulas in conjunctive normal form [16].

Each MUS can be regarded as a one "laconic justification" of the unsatisfiability of the original concept expression [8] (here "laconic" means that it contains no superfluous conjuncts). The UC is the set of all laconic justifications of the unsatisfiability. Clearly, it is sufficient to add a constraint that rules out *any* of the MUSes to exclude the current column from the set of solutions to (6).

Next, we show how to translate MUSes into linear inequalities. A MUS is a set of conjuncts $\{X'_i\}$ each of which corresponds to a binary variable (observe that $\eta$, as defined in (8), is a bijective function). By a slight abuse of notation we write $a_i = \eta^{-1}(X'_i)$ to denote the variable that corresponds to $C_i$. Then given a MUS $X' = \{X'_i\}_{i=1}^k$ we add the following linear constraint:

$$\sum_{i=1}^{k} a_i \leq k-1, \text{ where } a_i = \begin{cases} \eta^{-1}(X'_i), & X'_i = C_i \\ 1 - \eta^{-1}(X'_i), & X_i = \neg C_i \end{cases} \tag{9}$$

If a conjunctive concept contains $\bigsqcap X_i$ as a subexpression then all binary expressions $b_i$, i.e. either $a_i$ or $1 - a_i$ depending on whether $X_i$ is a positive or a negative literal, are equal to 1.

Therefore $\sum_{i=1}^{k} a_i = k$ where $k$ is the size of $\{X_i\}$. Constraining $\sum_{i=1}^{k} b_i$ to be less or equal to $k-1$ is equivalent to requiring at least one $b_i$ to be equal to 0. According to the definition of $\eta$ this is equivalent to removing of at least one conjunct from $X'$ which makes it satisfiable (due to minimality of $X'$, see Definition 1). Therefore, each of the constraints (9) is sufficient to exclude all columns, which correspond to concept expressions containing $X'$, from the set of solutions to (6). Observe that the constraints do not exclude any columns which do *not* include $X'$ since it is necessary to ensure completeness.

On step 8 the algorithm computes the unsatisfiable core for a concept expressions that corresponds to the current solution of (6). Then it transforms each of the MUSes into a linear inequality according to (9) and adds them to the binary program (6).

We call our PSAT algorithm (Algorithm 1) "hybrid" because it combines invocations of LP solver (to optimize (7)), MILP and $\mathcal{SROIQ}$ solvers (to optimize (6) and check satisfiability of concept expressions respectively). It is *iterative* because it iteratively tightens the set of solutions to (6) until either a valid column is found or provably no such column exists.

Finally, we give a short example demonstrating our iterative technique for computing valid columns.

**Example 1.** *Consider a PTBox where* $\mathcal{T} = \{A \sqsubseteq \exists R.C, B \sqsubseteq \exists R.\neg C, \geq 2R.\top \sqsubseteq D\}$ *and* $\mathcal{P}$ *contains some probabilistic constraints over the ordered set* $\Phi = \{A, B, D\}$*. Algorithm 1 starts out with an empty set of linear constraints for (6). The list of binary variables for (6) is* $(x_A, x_B, x_D)$*. Assume that at some iteration the algorithm generates the following column:* $A^j = (1, 1, 0, 1)$ *(the last component of any column is always equal to* 1 *because all coefficients in the last row of (7) are equal to* 1*). Then* $\eta(A^j) = A \sqcap B \sqcap \neg D$*.*

*It is not hard to see that* $\mathcal{T} \models \eta(A^j) \sqsubseteq \bot$*. The reason is that any instance* $o$ *of* $A \sqcap B$ *must have two R-successors (domain elements which are connected to* $o^{\mathcal{I}}$ *by* $R^{\mathcal{I}}$*). Moreover, they are necessarily distinct because one is an instance of* $C$ *and another is an instance of* $\neg C$*. Therefore,* $o$ *is an instance of* $\geq 2R.\top$ *and consequently is an instance of* $D$*. This is a contradiction with* $\neg D$ *in* $\eta(A^j)$*.*

*The unsatisfiable core of* $\eta(A^j)$ *is* $\{A, B, \neg D\}$*. This MUS is converted into the following linear inequality* $x_D \geq x_A + x_B - 1$ *which is then added to the binary program (6). As a result, no invalid column containing this MUS will be computed on subsequent iterations.*

### 3.3 Main Optimizations

We next describe several optimization techniques which play key roles for our implementation of the possible world generation algorithm.

#### 3.3.1 Exploiting Concept Hierarchy

The first optimization stems from a natural observation that many inequalities for the binary program (6) can be added simply by examining the structure of a TBox. Virtually all modern DL reasoners can efficiently construct a so called *classification hierarchy* by finding all subsumptions between concept names that are logically entailed by the TBox. Such hierarchy can be used to construct an initial set of inequalities $\mathcal{H}_0$.

Consider the following TBox $\mathcal{T} = \{A \sqcup B \sqsubseteq C\}$. The classified version of $\mathcal{T}$ should include subsumptions $A \sqsubseteq C$ and $B \sqsubseteq C$. Therefore they can be directly translated to inequalities $x_A \leq x_C$ and $x_B \leq x_C$ before computing some invalid column containing either $A \sqcap \neg C$ or $B \sqcap \neg C$ as subexpressions and converting these subexpressions into inequalities.

This idea helps to reduce the number of concept satisfiability tests. The effectiveness of this technique depends on axiomatic richness of the TBox. For axiomatically weak TBoxes, where almost all subsumptions can be discovered by traversing the concept hierarchy, most of the set $\mathcal{H}$ is computed up front. More complex TBoxes may have non-trivial entailments involving concept expressions on both left-hand and right-hand sides which can only be discovered when checking validity of some column candidate. One such example is the subsumption $A \sqcap B \sqsubseteq D$ from Example 1.

#### 3.3.2 Optimistic Inequality Generation

One issue with a naive implementation of Algorithm 1 is that computing unsatisfiability cores may appear impractical for certain concept expressions and TBoxes. This may especially happen for long expressions which contain MUSes with little or no overlap. It is well known from the model diagnosis theory that finding all minimal unsatisfiable sets may require an exponential (in the size of all unsatisfiable sets) number of SAT tests [20].

To address this issue the algorithm imposes a time limit on the procedure that computes the UC. If at least one MUS has been found but finding others exceeds the timeout the procedure is interrupted. The found MUSes are then converted to linear inequalities and the algorithm moves on as if the full UC was computed.

This optimization does not cause a loss of either soundness or completeness. Completeness is trivially preserved because not adding some inequalities to the program (6) can only expand its set of solutions, so no possible world could be missed. Soundness is preserved because each computed column is still valid (SAT tests are never interrupted). The only possible negative impact of missing some MUSes is that they can appear in some future column candidates, so the algorithm might go through additional iterations. However, they do not *have to* appear because the optimal basis for the main program (7) can often be found before considering column candidates containing those MUSes. Intuitively, the algorithm behaves *optimistically* by hoping that additional iterations will not be required.

#### 3.3.3 Multiple Column Generation and Stabilization

We use several techniques aimed at improving convergence of the column generation process. The most important are generating several optimal solutions of (6), i.e. column candidates, and introducing additional variables for the main linear program (7) to stabilize dual space and reduce degeneracy.

It is known that adding multiple columns into basis at every simplex iteration can improve convergence of column generation. For instance, Hansen and Perron add up to 50 column per iteration [6]. This is made possible by their heuristic method of generating columns (the variable neighborhood heuristics) which allows them to quickly generate many sub-optimal columns having positive reduced cost. In our case, when columns are generated by a MILP solver, there are the following two possibilities.

First, some MILP solvers, in particular, CPLEX, support *solution pools* which store multiple optimal or sub-optimal solutions for an MILP problem instance. Such solvers can either return the set of sub-optimal solutions which they recorded during the optimization process or continue the branch-and-cut search after the optimum until the required number of solutions has been found. Second, if the solver does not support solution pools one may still force it generate multiple solutions. This can be done by "cutting off" the optimal solution and re-optimizing or by hooking inside the solver to continue the search after the optimum.

Pronto is highly modular and can be used with different LP/MILP solvers. Using a solution pool is the first choice strategy if it is available. If it is not available, as is the case with GLPK, then the second option is used. However, since it has implications for performance fewer columns are generated (usually up to 5).

Similarly to [6] we also use specific techniques to stabilize the values of the dual variables and reduce degeneracy.[3] This involves adding extra variables to the main linear program so it looks as follows:

$$\begin{aligned} \max \ & \sum_{I \in I_\Phi} x_I - \delta^+ y^+ - \delta^- y^- \\ \text{s.t.} \ & \sum_{I \models C_i} x_I = p_i + y^+ - y^- \\ & \sum_{I \in I_\Phi} x_I \leq 1, \ x_I, y^+, y^- \geq 0 \end{aligned} \tag{10}$$

where $y^+$ and $y^-$ are the column vectors $(y_1^+, \ldots, y_m^+)^T$, $(y_1^-, \ldots, y_m^-)^T$ respectively while $\delta^+$,$\delta^-$ are fixed positive coefficients. Observe that the original PSAT program (7) has the optimal value of 1 *iff* the augmented program has the optimal value of 1 (so soundness and completeness are preserved). However, the extra variables allow the objective function to vary smoothly between 0 and 1 which generally improves convergence of the column generation algorithms.

Our technique is simpler than the iterative method used by Hansen and Perron [6] but appears to be more efficient in most of our tests. At the same time one may combine different techniques or switch between them in the process of generating columns.

#### 3.3.4 Early Unsatisfiability Detection

Automated reasoners often implement heuristic approaches to quickly detect obvious reasons for unsatisfiabilty of logical formulae. One good example of such techniques is the early clash detection methods employed by virtually all mature DL reasoners. Probabilistic reasoners are no exception, for instance, propositional PSAT solvers sometimes use incomplete rule-based reasoning methods which prove to be tremendously effective at proving unsatisfiability [6].

Since rule sets have only been formulated in the propositional case we take another approach. It is based on the observation that if the PSAT program has an optimal objective value of 1 (i.e. the KB is satisfiable) then the optimal dual solution is of the form $(0, \ldots, 0, 1)$. Components of the dual solution show the rate at which the objective value of the primal program will improve in response to a small relaxation of row bounds. If the optimal value of the program (7) is 1 then the only row whose relaxation can improve the objective is the last one, i.e. $\sum_{I \in I_\Phi} x_I \leq 1$. Its coefficients are exactly the same as the objective coefficients, thus its dual value is 1 while the other dual values are zeros. On the other hand, if the optimal objective value is below 1 then the indexes of non-zero dual variables indicate conflicting inequalities and, consequently, conflicting conditional constraints.

Our algorithm maintains history of dual solutions over a fixed number of column generation iterations. The history helps to spot the situation when the *same* dual variables repeatedly take on non-zero values while the primal objective is improving very slowly. More precisely, if over the last 10 iterations:

[3] Degeneracy is the situation when some basic variables have zero values. It is known to be a problem for simplex since it can lead to multiple column swaps without an improvement of the objective value.

- The objective function improved by less than 1% and,
- More than 10% of dual values that were non-zero on *some* iterations had non-zero values on *all* iterations

then it is a good indication that the set of conditional constraints, which correspond to those dual variables, is not satisfiable. In that case, satisfiability of such constraints is tested separately. If they are indeed unsatisfiable, the whole process stops and unsatisfiability is reported, otherwise the main column generation loop continues.

The technique preserves *soundness* because of monotonicity: if some subset of a KB is unsatisfiable, then the whole KB is unsatisfiable. It also preserves *completeness* since for each satisfiable KB the column generation process will necessarily continue until satisfiability is proved (all extra tests, if any, must be negative).

The heuristic happens to be very effective for unsatisfiable KBs. In our experiments more than 90% of unsatisfiable KBs have been proved unsatisfiable by this method. It is effective mostly because conflicting sets of constraints tend to be small so it can easily be spotted if a small number of duals progress towards non-zero values. However, if the KB is satisfiable then the heuristic can slow the column generation down by introducing extra tests but this has so far happened in less than 15% of the cases.

## 4 Experimental Evaluation

Since P-$\mathcal{SROIQ}$ was designed as an extension of the DLs behind OWL and compatibility with OWL is declared as one of its major advantages, it is critical to ensure that the reasoning algorithms can successfully deal with probabilistic extensions of real OWL ontologies, not just randomly generated probabilistic clauses (as in [6]) or propositional taxonomies. We used the the following criteria to choose ontologies from a large variety of currently available ones:

- One of our long-term goals is to provide tools for reasoning over probabilistic extensions of OWL ontologies, so it is reasonable to evaluate the performance on ontologies which use as many features of OWL 2 as possible. At the same time ontologies represented in a lightweight fragment of OWL 2, such as OWL EL (based on logic $\mathcal{EL}^{++}$), also need to be included, especially given that they are getting increasingly widespread in some important domains, such as medical informatics.
- The ontologies should have reasonably large TBoxes with at least few hundred concept subsumption, equivalence or disjointness axioms, and some object roles.
- The ontologies should have at least 500 concepts in the TBox to show that the reasoner can handle large probabilistic signatures.
- Ideally, the ontologies should be "in service", i.e. have been created for and be in use by real applications as opposed to educational or experimental purposes. In that case they are more likely to encompass common and useful modeling patterns.

We selected the following five ontologies from different domains. For all ontologies we use versions stored in the TONES repository.[4]

[4] http://owl.cs.manchester.ac.uk/repository/

**The Subcellular Anatomy Ontology** (SAO) is the ontology from the neuroinformatics domain describing cellular and subcellular structures, supracellular domains, and macromolecules.[5] It contains 737 concepts, 915 subsumption, 4 equivalence, and 1580 concept disjointness axioms, 36 object properties and 47 data properties.

**The Process Ontology** is a part of the SWEET (Semantic Web for Earth and Environmental Terminology) collection of ontologies developed by NASA to provide semantic support for various Earth science projects.[6] It contains 1537 concepts, 1922 subsumption, 84 equivalence, and 1 concept disjointness axioms, 102 object properties and 19 data properties.

**The Sequence Ontology with Composite Terms** (SO-XP) defines terms and relationships used to describe features and attributes of biological sequence as well as cross-product definitions for composite terms.[7] It is a deliverable of the Gene Ontology Project and the Open Biomedical Ontologies (OBO) experiment. It contains 1660 concepts, 1709 subsumption, 198 equivalence, and 21 concept disjointness axioms and 22 object properties.

**The Teleost Anatomy Ontology** (TAO) is a multi-species anatomy ontology for teleost fishes.[8] It contains 2229 concepts, 3 object properties and 3406 concept subsumption axioms.

**The Cell Type Ontology** (CO) is a structured controlled vocabulary for cell types constructed for model organism and other Bioinformatics databases.[9] It contains 857 concepts, 1 object property and 1263 concept subsumption axioms.

Neither of these ontologies are propositional or small and simple enough to consider their propositionalization and a subsequent use of a propositional probabilistic SAT solver as a feasible alternative. None of the previously developed PSAT algorithms is capable of dealing with thousands of classical axioms in addition to a comparable number of probabilistic formulas.

PSAT instances over these ontologies were generated by the random selection of pairs of concept names. The proportion of unconditional constraints was kept at approximately 10%. Although there is not yet an established best practice for modeling in P-$\mathcal{SROIQ}$, our initial investigation [11] suggested that most PTBox constraints are conditional. This is also the case with Bayesian modeling. Unconditional constraints are mostly statements about individuals but since they are restricted (see [13]) their number is usually limited.

It is known that satisfiable probabilistic KBs are typically more difficult for PSAT algorithms [6, 4]. We have also observed that on average our algorithm generates less than half the columns for unsatisfiable instances as compared to satisfiable instances. This is mostly due to the early unsatisfiability detection technique (see Section 3.3.4). Therefore we implemented a specific technique to generate *satisfiable* KBs which follows the generic methodology used in [6]. It is based on generating two sets of possible worlds of fixed size with two probability distributions. Possible worlds are generated using a $\mathcal{SROIQ}$ reasoner such that each concept in the probabilistic signature has an approximately equal chance of being selected for the next world. Two probability functions $Pr_1$ and $Pr_2$ are then randomly selected from the set of normal probability distributions over the set of possible worlds. For each constraint $(D|C)[l, u]$ we take $l$ (resp. $u$) as the minimum (resp. maximum) probability which is assigned to $(D|C)$ by $Pr_1$ and $Pr_2$. Intuitively, the existence of these interpretations guarantees satisfiability of the KB because they are some of its models. The early unsatisfiability detection heuristic has been kept on during the experiments to account for any loss of performance it may cause.

[5] http://ccdb.ucsd.edu/CCDBWebSite/sao.html
[6] http://sweet.jpl.nasa.gov/ontology/
[7] http://wiki.geneontology.org/index.php/SO:Composite_Terms
[8] https://www.nescent.org/phenoscape/Teleost_Anatomy_Ontology
[9] http://obolibrary.org/cgi-bin/detail.cgi?id=cell

IBM CPLEX solver was used to solve all LP and MILP problem instances. Other solvers can be easily plugged into Pronto. We have experimented with GLPK[10] and the results tend to be 25%-50% worse than those presented below.

All the experiments were conducted on a desktop PC with 2GHz CPU and 2GB RAM with JRE 1.6 running under Windows XP SP3. For each PSAT test we measured the following parameters: total CPU time (in seconds), total number of columns generated, and the average time to generate a new column (CG time, in milliseconds). CPU time and the number columns are averaged over 10 PSAT instances for each KB's size while CG time is also averaged over all columns generated during a single PSAT. The results are presented in Table 1.

Table 1: Results of PSAT evaluation on satisfiable PTBoxes

| Ontology | Language | TBox size | Signature size | PTBox size | Total time (s) | CG time (ms) | # columns |
|---|---|---|---|---|---|---|---|
| SAO | $\mathcal{SHIN}$ | 2499 | 125 | 250 | 78.85 | 449.86 | 138.8 |
| | | | 250 | 500 | 161.16 | 450.24 | 261 |
| | | | 325 | 750 | 360.97 | 1023.86 | 351.4 |
| | | | 500 | 1000 | 1275.85 | 3062.52 | 424.8 |
| Process | $\mathcal{SHOF}$ | 2007 | 125 | 250 | 48.48 | 337.38 | 87.2 |
| | | | 250 | 500 | 114.68 | 380.76 | 180.6 |
| | | | 325 | 750 | 224.42 | 480.26 | 280.8 |
| | | | 500 | 1000 | 416.66 | 509.26 | 408.8 |
| SO-XP | $\mathcal{SHI}$ | 1928 | 125 | 250 | 58.62 | 423.08 | 74.4 |
| | | | 250 | 500 | 186.49 | 741.76 | 184.8 |
| | | | 325 | 750 | 424.66 | 986.8 | 318.4 |
| | | | 500 | 1000 | 799.38 | 1524.4 | 422.4 |
| TAO | $\mathcal{EL}^{++}$ | 3406 | 125 | 250 | 48.63 | 280.96 | 91.4 |
| | | | 250 | 500 | 123.36 | 379.68 | 191 |
| | | | 325 | 750 | 224.69 | 414.16 | 281.4 |
| | | | 500 | 1000 | 430.69 | 466.54 | 449.02 |
| Cell Type | $\mathcal{EL}^{++}$ | 1263 | 125 | 250 | 53.8 | 321.1 | 87.2 |
| | | | 250 | 500 | 128.2 | 386.54 | 180.8 |
| | | | 325 | 750 | 236.85 | 428.4 | 280.8 |
| | | | 500 | 1000 | 420.93 | 484.78 | 394.6 |

The major outcome is that our algorithm exhibits a very good convergence on probabilistic extensions of real ontologies. This appears to be mostly due to rich TBoxes which effectively shrink the set of all possible world thus allowing the algorithm to faster arrive at the optimal PSAT program. For instance, in the case of the SAO ontology our algorithm was able to solve all instances of PSAT of size 1000 never generating more than 600 columns out of the space of $2^{500}$ (excluding invalid ones). A likely explanation is a high number of concept disjointness axioms in the SAO ontology.

Due to the space restriction we do not include the results on propositional KBs. In that case Pronto's performance is comparable to that of propositional PSAT algorithms [6, 4]. The algorithm tends to have a worse convergence on such KBs but columns are generated faster because the optimization program (6), that is used to produce columns, has fewer inequalities. This is a direct consequence of simpler TBoxes.

Our plan is to continue the experiments with probabilistic extensions of real ontologies. Apart from posing interesting challenges to the PSAT algorithm, such ontologies are also more valuable with respect to developing useful methodologies for probabilistic modeling. They present examples of interesting interactions between classical and probabilistic knowledge which can be

[10]The GNU Linear Programming Kit, http://www.gnu.org/software/glpk/

useful for augmenting real ontologies with, for example, domain statistics.

## 5 Related Work

There is extensive work on solving the PSAT problem in propositional probabilistic logic (see esp. [6, 19, 4, 7]). Those algorithms are similar in spirit since they are also based on the column generation technique but are different in scope, design, and implementation.

The major difference between propositional and non-propositional PSAT problems in the context of column generation is that propositionality of the KB allows encoding of all its structure in a polynomial number of linear inequalities over a polynomial number of binary variables. This follows directly from the well known reduction of propositional SAT to integer programming [7]. Therefore, the column generation problem (6) is much easier to handle, either as a standard MILP instance [7, 4] or as a non-linear 0-1 program [6, 19]. To the best of our knowledge, this work is the first to describe an algorithm and evaluation results for non-propositional PSAT.

Our method does *not* try to capture the classical part of the KB at once. By interacting with a classical SAT solver for the target logic we capture only those parts which are essential for solving a particular PSAT or entailment problem. While this may be less efficient on KBs with weak classical part (or expressed in a restricted language, such as propositional logic) it has important advantages. First, it allows us to handle essentially any target logic for which a SAT solver is available. Second, this makes our algorithm more scalable with respect to the amount of classical knowledge. For example, modern DL reasoners can efficiently solve concept satisfiability problems even for very large TBoxes containing thousands of axioms (a characteristic example is the NCI Thesaurus). Even if they were propositional (or could be propositionalized) and fully translated into constraints for the problem (6), the latter would be drastically larger. Instead, as our results show, we can often solve PSAT by capturing only some relevant parts of TBoxes.

The PSAT algorithm presented in this paper is a substantial improvement of the earlier described algorithm [12] which is based on a similar idea but generates columns by solving a generic constraint optimization problem. The MILP formulation, presented in this paper, appears to be much more tractable and amenable to optimizations which resulted in the scalability improvements of the order of magnitude.

## 6 Conclusion

The primary conclusion of this work is that it is highly likely that P-$\mathcal{SROIQ}$ is as "practical" an ontology language, at least from the computational point of view, as $\mathcal{SROIQ}$. PSAT and the various entailment problems for P-$\mathcal{SROIQ}$ are `N2ExpTime`-complete, so, as with any logic in the $\mathcal{SROIQ}$ family, practicality, efficiency, and scalability claims must be carefully qualified. However, our current system handles all our randomly generated problems, which stress characteristics that we have good reason to believe to be present in future ontologies, to scales that are reasonable for hand built probabilistic ontologies in the next few years. As with $\mathcal{SROIQ}$, it is certainly possible to generate — or to find naturally — KBs that will utterly defeat our current optimizations, but the transformation from being barely able to handle 10-50 probabilistic statements to being handle thousands is a game changing improvement. Of course, $\mathcal{SROIQ}$ reasoners have a much larger suite of optimizations to cope with a broader set of ontology types. But this is a difference in degree, not kind.

For example, prior to the current PSAT algorithm the breast cancer risk ontology (BCRA) [11] was nigh impossible to work with. We were reduced to working only with subsets of the

ontology and had to abandon the idea of importing it into a new ovarian cancer ontology. This is clearly an unacceptable development situation for probabilistic ontology modelers. But now it makes sense for ontology modelers to experiment seriously with P-$\mathcal{SROIQ}$: Not only is it likely that Pronto can handle their needs, but we now have confidence that additional optimizations (or workarounds) can be designed as we encounter new difficult problems. In this sense, we are now on the same footing as the $\mathcal{SROIQ}$ family of logics: Modelers keep breaking reasoners and reasoners keep adapting to the new challenges.

## References


[1] F. Baader, D. Calvanese, D. McGuiness, D. Nardi, and P. F. Patel-Schneider. *Description Logic Handbook*. Cambridge University Press, 2003.

[2] F. Bacchus. *Representing and reasoning with probabilistic knowledge*. MIT Press, 1990.

[3] S. Castano, A. Ferrara, D. Lorusso, T. H. Näth, and R. Möller. Mapping validation by probabilistic reasoning. In *European Conference on Semantic Web*, pages 170–184, 2008.

[4] P. S. de Souza Andrade, J. C. F. da Rocha, D. P. Couto, A. da Costa Teves, and F. G. Cozman. A toolset for propositional probabilistic logic. In *Encontro Nacional de Inteligencia Artificial*, pages 1371–1380, 2007.

[5] J. Halpern. An analysis of first-order logics of probability. *Artificial Intelligence*, 46:311–350, 1990.

[6] P. Hansen and S. Perron. Merging the local and global approaches to probabilistic satisfiability. *Int. Journal of Approximate Reasoning*, 47(2):125–140, 2008.

[7] J. Hooker. Quantitative approach to logical reasoning. *Decision Support Systems*, 4:45–69, 1988.

[8] M. Horridge, B. Parsia, and U. Sattler. Laconic and precise justifications in OWL. In *International Semantic Web Conference*, pages 323–338, 2008.

[9] I. Horrocks, O. Kutz, and U. Sattler. The even more irresistible SROIQ. In *Knowledge Representation and Reasoning*, pages 57–67, 2006.

[10] Y. Kazakov. SRIQ and SROIQ are harder than SHOIQ. In *Knowledge Representation and Reasoning*, pages 274–284, 2008.

[11] P. Klinov and B. Parsia. Probabilistic modeling and OWL: A user oriented introduction into P-$\mathcal{SHIQ}$(D). In *OWL: Experiences and Directions (OWLED-2008)*, 2008.

[12] P. Klinov and B. Parsia. On improving the scalability of checking satisfiability in probabilistic description logics. In *International Conference on Scalable Uncertainty Management*, volume 5785/2009 of *Lecture Notes in Computer Science*, pages 138–149. Springer, 2009.

[13] P. Klinov and B. Parsia. Relationships between probabilistic description and first-order logics. In *International Workshop on Uncertainty in Description Logics*, 2010.

[14] P. Klinov, B. Parsia, and D. Picado. The consistency of the CADIAG-2 knowledge base: A probabilistic approach. In *Logic for Programming, Artificial Intelligence and Reasoning*, 2010.

[15] T. Lukasiewicz. Expressive probabilistic description logics. *Artificial Intelligence*, 172(6-7):852–883, 2008.

[16] I. Lynce and J. P. M. Silva. On computing minimum unsatisfiable cores. In *International Conference on Theory and Applications of Satisfiability Testing*, 2004.

[17] K. McMillan. Applying SAT methods in unbounded symbolic model checking. In *Computer Aided Verification*, pages 250–264, 2002.

[18] N. J. Nilsson. Probabilistic logic. *Artificial Intelligence*, 28(1):71–87, 1986.

[19] Z. Ognjanovic, U. Midic, and N. Mladenovic. A hybrid genetic and variable neighborhood descent for probabilistic SAT problem. In *Hybrid Metaheuristics*, pages 42–53, 2005.

[20] R. Reiter. A theory of diagnosis from first principles. *Artificial Intelligence*, 32:57–95, 1987.

[21] E. Sirin, B. Parsia, B. Cuenca Grau, A. Kalyanpur, and Y. Katz. Pellet: A practical OWL-DL reasoner. *Journal of Web Semantics*, 5(2):51–53, 2007.

# Optimizing the AES S-Box using SAT*

Carsten Fuhs (fuhs@informatik.rwth-aachen.de)
LuFG Informatik 2, RWTH Aachen University, Germany
Peter Schneider-Kamp (petersk@imada.sdu.dk)
IMADA, University of Southern Denmark, Denmark

## 1 Introduction

In this paper we describe the implementation of a technique for minimizing XOR circuits used in cryptographic algorithms. More precisely, we present our work from [4] for encoding this synthesis problem to SAT with a focus on the case study of optimizing an important component of the Advanced Encryption Standard (AES) [8]. In addition to these previously published contributions, we report on novel encouraging experimental results that allow us to actually prove optimality of the results obtained.

The AES algorithm consists of the (repeated) application of four steps. The main step for introducing non-linearity is the SubBytes step that is based on a so-called S-box. This S-box is a transformation based on multiplicative inverses in GF($2^8$) combined with an invertible affine transformation. This step can be decomposed into two linear parts and a minimal non-linear part. We focus on the optimization of the linear parts, in particular the first one (called the "top matrix" in [2]).

In this paper, we assume that we have $n$ inputs $x_1,\ldots,x_n$ and $m$ outputs $y_1,\ldots,y_m$. Then the linear function to be computed can be specified by $m$ equations of the form $y_\ell = a_{\ell,1}\cdot x_1 \oplus a_{\ell,2}\cdot x_2 \oplus \ldots \oplus a_{\ell,n}\cdot x_n$ for $1 \leq \ell \leq m$. We call each equation a *linear form*. Note that each $a_{\ell,j}$ is a constant from GF(2) $= \{0,1\}$, each $x_j$ is a variable over GF(2), and $\oplus$ and $\cdot$ denote standard addition and multiplication on GF(2).

Our goal is to come up with an algorithm that computes these linear forms given $x_1,\ldots,x_n$ as inputs. More specifically, we would like to express this algorithm via a *linear straight-line program* (or, for brevity, just *program*). Here, every line of the program has the shape $u := e\cdot v \oplus f\cdot w$ with $e,f \in$ GF(2) and $v,w$ variables. Some lines of the program will contain the output, i.e., assign the value of one of the desired linear forms to a variable. The *length* of a program is the number of lines the program contains. A program is *optimal* if there is no shorter program that computes the same linear forms.

**Example 1.** *Consider the following linear forms:*

$$y_1 = x_1 \oplus x_2 \oplus x_3 \qquad y_2 = x_1 \oplus x_2 \oplus x_3 \oplus x_4 \qquad y_3 = x_1 \oplus x_2 \oplus x_4$$

*A shortest linear program for computing these linear forms has length* 4*. The following linear program is an optimal solution for this example (and demonstrates cancellation).*

$$\begin{array}{lllll} v_1 = x_1 \oplus x_2 & & & v_3 = x_4 \oplus v_2 & [y_2] \\ v_2 = x_3 \oplus v_1 & [y_1] & & v_4 = x_3 \oplus v_3 & [y_3] \end{array}$$

*For each output $y_\ell$ there is a variable $v_i$ that contains the linear form for $y_\ell$. This mapping from variables to outputs is given by annotating the program lines with the associated output in square brackets.*

The goal is to find an optimal linear straight-line program for a given set of linear forms both automatically and efficiently. In [4] we first present an encoding of the associated decision problem to SAT:

> Given $n$ variables $x_1,\ldots,x_n$ over GF(2), $m$ linear forms $y_\ell = a_{\ell,1}\cdot x_1 \oplus \ldots \oplus a_{\ell,n}\cdot x_n$ and a natural number $k$, decide if there exists a linear program of length $k$ that computes all $y_\ell$.

Of course, if the answer to this question is "Yes", we do not only wish to get this answer, but we would also like to obtain a corresponding program of length (at most) $k$. To this end, we ensure that each model of the propositional formula describes a program of lenth (at most) $k$.

*Supported by the G.I.F. grant 966-116.6 and the Danish Natural Science Research Council.

**Matrix Representation**

To facilitate the description of our encoding, we reformulate the problem via matrices over GF(2). Here, given a natural number $k$, we represent the given coefficients of the $m$ linear forms over $n$ inputs with $y_\ell = a_{\ell,1} \cdot x_1 \oplus a_{\ell,2} \cdot x_2 \oplus \ldots \oplus a_{\ell,n} \cdot x_n$ ($1 \leq \ell \leq m$) as rows of an $m \times n$-matrix $A$. The $\ell$-th row thus consists of the entries $a_{\ell,1} a_{\ell,2} \ldots a_{\ell,n}$ from GF(2). Likewise, we can also express the resulting program via two matrices over GF(2): $B = (b_{i,j})_{k \times n}$, where $b_{i,j} = 1$ iff in line $i$ of the program the input variable $x_j$ is read, and $C = (c_{i,j})_{k \times k}$, where $c_{i,j} = 1$ iff in line $i$ of the program the intermediate variable $v_j$ is read. To represent for example the program line $v_3 = x_4 \oplus v_2$ from Example 1, all $b_{3,j}$ except for $b_{3,4}$ and all $c_{3,j}$ except for $c_{3,2}$ have to be 0. Thus, the third row in $B$ is $\begin{pmatrix} 0 & 0 & 0 & 1 \end{pmatrix}$ while in $C$ it is $\begin{pmatrix} 0 & 1 & 0 & 0 \end{pmatrix}$. Now, for the matrices $B$ and $C$ to actually represent a legal linear straight-line program, for any row $i$ there must be exactly two non-zero entries in the combined $i$-th row of $B$ and $C$. That is, the vector $\begin{pmatrix} b_{i,1} & \ldots & b_{i,n} & c_{i,1} & \ldots & c_{i,k} \end{pmatrix}$ must contain exactly two 1s.

Furthermore, for the represented program to actually compute our linear forms, we have to demand that for each desired output $y_\ell$, there is a line $i$ in the program (and the matrices) such that $v_i = y_\ell$ where $y_\ell = a_{\ell,1} \cdot x_1 \oplus \ldots \oplus a_{\ell,n} \cdot x_n$ and $v_i = b_{i,1} \cdot x_1 \oplus \ldots \oplus b_{i,n} \cdot x_n \oplus c_{i,1} \cdot v_1 \oplus \ldots \oplus c_{i,i-1} \cdot v_{i-1}$. Finally, to represent the mapping of intermediate variables to outputs, we use a function $f : \{1, \ldots, m\} \to \{1, \ldots, k\}$.

**Example 2.** *Consider again the linear forms from Example 1. They are represented by the following matrix A. Likewise, the program is represented by the matrices B and C and the function f.*

$$A = \begin{pmatrix} 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 \end{pmatrix} \quad B = \begin{pmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad C = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \quad f = \begin{cases} 1 & \mapsto & 2 \\ 2 & \mapsto & 3 \\ 3 & \mapsto & 4 \end{cases}$$

*Obviously, all combined rows of B and C contain exactly two non-zero elements. Moreover, by computing the $v_i$ and the $y_\ell$, we see that each of the linear forms from A is computed by the program in B and C.*

**Encoding to Propositional Logic**

In [4] we give an encoding of the decision problem as a logical formula in second order logic and perform a stepwise refinement of that encoding such that we can finally reduce it to satisfiability of propositional logic. For the sake of brevity, we make use of cardinality constraints represented by predicates like $\text{exactly}_k$ that take a list of variables and check that the number of variables that are assigned 1 is exactly $k$. The final encoding to SAT can be performed by using [3, 1]. With these constraints, we can ensure that $B$ and $C$ represent a legal linear straight-line program. This is encoded by the following formula $\beta_1$:

$$\beta_1 = \bigwedge_{1 \leq i \leq k} \text{exactly}_2(b_{i,1}, \ldots, b_{i,n}, c_{i,1}, \ldots, c_{i,i-1})$$

For $1 \leq j \leq n$ and $1 \leq i \leq k$, we introduce the auxiliary formulae $\psi(j,i)$, which denotes the dependence of the value for $v_i$ with respect to $x_j$ (i.e., whether the value of $v_i$ toggles if $x_j$ changes or not):

$$\psi(j,i) = b_{i,j} \oplus \bigoplus_{1 \leq p < i} c_{i,p} \wedge \psi(j,p)$$

We get the following encoding $\delta$ which expresses that $f$ is a function and the $i$-th intermediate variable indicated by $f(\ell)$ actually computes the $\ell$-th linear form.

$$\delta = \beta_1 \wedge \bigwedge_{1 \leq \ell \leq m} \left( \bigwedge_{1 \leq i \leq k} \left( f_{\ell,i} \to \bigwedge_{1 \leq j \leq n} \left( \psi(j,i) \leftrightarrow a_{\ell,j} \right) \right) \wedge \text{exactly}_1(f_{\ell,1}, \ldots, f_{\ell,k}) \right)$$

For the implementation of $\delta$ we use the SAT framework in the verification environment AProVE [5] and the Tseitin implementation from SAT4J [7]. As shown in [4], given a decision problem with an $m \times n$ matrix and a natural number $k$ (where w.l.o.g. $m \leq k$ holds since for $m > k$, we could just set $\delta = 0$), our encoding $\delta$ has size $\mathcal{O}(n \cdot k^2)$ if the cardinality constraints are encoded in linear space [3].

## 2 Case Study: Advanced Encryption Standard

As mentioned in the introduction, a major motivation for our work is the minimization of circuits for implementing cryptographic algorithms. For our case study, we consider the first of the linear parts (called the "top matrix" in [2]) which is represented by the following $21 \times 8$ matrix $A$:

$$A = \begin{pmatrix}
0&1&1&0&0&0&0&1\\
1&1&1&0&0&0&0&1\\
1&1&1&0&0&1&1&1\\
0&1&1&1&0&0&0&1\\
0&1&1&0&0&0&1&1\\
1&0&0&1&1&0&1&1\\
0&1&0&0&1&1&1&1\\
1&0&0&0&0&1&0&0\\
1&0&0&1&0&0&0&0\\
1&1&1&1&1&0&1&0\\
0&1&0&0&1&1&1&0\\
1&0&0&1&0&1&1&0\\
1&0&0&0&0&0&1&0\\
0&0&0&1&0&1&0&0\\
1&0&0&1&1&0&1&0\\
0&0&1&0&1&1&1&0\\
1&0&1&1&0&1&0&0\\
1&0&1&0&1&1&1&0\\
0&1&1&1&1&1&1&0\\
1&1&0&1&1&1&1&0\\
1&0&1&0&1&1&0&0
\end{pmatrix}
\quad
B = \begin{pmatrix}
0&0&1&0&0&0&0&1\\
1&0&0&0&0&0&1&0\\
0&0&1&0&0&0&0&0\\
0&0&0&0&1&0&0&0\\
0&1&0&0&0&0&0&0\\
1&0&0&0&0&0&1&0\\
1&0&0&1&0&0&0&0\\
0&0&0&0&0&0&1&0\\
0&0&0&1&0&0&0&0\\
0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0\\
0&0&0&1&0&0&0&0\\
0&0&0&0&0&0&0&0\\
0&0&0&1&0&0&0&0\\
0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&1&0\\
0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&1\\
0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0
\end{pmatrix}$$

$$C = \begin{pmatrix}
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&1&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&1&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&1&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&1&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0\\
0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0\\
0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0\\
0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0
\end{pmatrix}$$

Here, the matrices $B$ and $C$ represent a solution with length $k = 23$. This solution was found in less than one minute using our decision procedure from Section 1 with MiniSAT v2.1 as backend on a 2.67 GHz Intel Core i7. We now know that $k_{\min} = 23$ and, indeed, the shortest previously known linear straight-line program for the linear forms described by the matrix $A$ has length $k = 23$ [2]. This shows that our SAT-based optimization method is able to find very good solutions in reasonable time. Unfortunately, proving the unsatisfiability for $k = 22$ proves to be much more challenging. Indeed, we have run many different SAT solvers (including but not limited to glucose, ManySat, MiniSat, MiraXT with 8 threads, OKsolver, PicoSAT, PrecoSAT, RSat, SAT4J) on the CNF file for this instance of our decision problem. Some promising solvers were run for more than 40 days without terminating.

In an effort to prove unsatisfiability of this instance and thereby prove optimality of the solution with $k = 23$, we have asked for and received a lot of support and good advice from the SAT community (see the Acknowledgements at the end of this paper). At SAT'10, we were pointed to CryptoMiniSat [9], a solver featuring special support for dealing with (implicit) XOR gates, which are often used for cryptographic functions. Using version 2.5.1 of this solver, we were able to prove optimality of the corresponding instance with the improvements of Section 3 on an Opteron 848 at 2.2 GHz within less than 106 hours. Using pre-processing techniques, the number of variables of this instance can be reduced from more than 45000 to less than 5000 in a matter of minutes.[1]

To analyze how difficult these problems really are, we consider a small subset of the linear forms to be computed for the top matrix. The table to the right shows how the runtimes in seconds of the SAT solver are affected by the choice

| $k$ | result | time |
|---|---|---|
| 8 | UNSAT | 0.4 |
| 9 | UNSAT | 0.5 |
| 10 | UNSAT | 1.2 |
| 11 | UNSAT | 5.0 |
| 12 | UNSAT | 76.8 |
| 13 | SAT | 1.0 |
| 14 | SAT | 3.4 |
| 15 | SAT | 2.8 |
| 16 | SAT | 1.5 |
| 17 | SAT | 4.3 |
| 18 | SAT | 2.7 |
| 19 | SAT | 2.5 |

[1] The reader is cordially invited to try his favorite SAT solver on one of the instances available from: `http://aprove.informatik.rwth-aachen.de/eval/slp.zip`

of $k$ for the case that we consider only the first 8 out of 21 linear forms from $A$. In order to keep runtimes manageable we already incorporated the symmetry breaking improvement described in Section 3. Note that unsatisfiability for $k = 12$ is still much harder to show than satisfiability for $k_{min} = 13$.

To conclude this case study, we see that while finding (potentially) minimal solutions is obviously feasible, proving their optimality (i.e., unsatisfiability of the associated decision problem for $k = k_{min} - 1$) is challenging. This observation confirms observations made in [6]. In the following section we present some of our attempts to improve the efficiency of our encoding for the UNSAT case.

# 3 Symmetry Breaking

Satisfiability of propositional logic is an NP-complete problem and, thus, we can expect that at least some instances are computationally expensive. While SAT solvers have proven to be a Swiss army knife for solving practically relevant instances of many different NP-complete problems, our kind of program synthesis problems seems to be a major challenge for today's SAT solvers even on instances with "just" 1500 variables. In this section we discuss an approach based on symmetry breaking.

In general, having many solutions is considered good for SAT instances as the SAT solver is more likely to "stumble" upon one of them. For UNSAT instances, though, having many potential solutions usually means that the search space to exhaust is very large.

One of the main reasons for having many solutions is symmetry. For example, it does not matter if we first compute $v_1 = x_1 \oplus x_2$ and then $v_2 = x_3 \oplus x_4$ or the other way around. Limiting these kinds of symmetries can be expected to significantly reduce the runtimes for UNSAT instances. In our concrete setting, being able to reorder independent program lines is one of the major sources of symmetry. Two outputs in a straight-line programs are said to be *independent* if neither of them depends on the other (directly through the matrix $C$ or indirectly).

Now, the idea for breaking symmetry is to impose an order on these lines: the line which computes the "smaller" linear form (w.r.t. a total order on linear forms, which can e.g. be obtained by lexicographic comparison of the coefficient vectors) must occur before the line which computes the greater linear form.

We can encode the direct dependence of $v_i$ on $v_p$:

$$\bigwedge_{1 \le i \le k} \bigwedge_{1 \le p < i} c_{i,p} \rightarrow dep_{i,p}$$

Likewise, the indirect dependence of $v_i$ on $v_p$ can be encoded by transitivity:

$$\bigwedge_{1 \le i \le k} \bigwedge_{1 \le p < i} \bigwedge_{p < q < i} c_{i,q} \wedge dep_{q,p} \rightarrow dep_{i,p}$$

We also need to encode the reverse direction, i.e.:

$$\bigwedge_{1 \le i \le k} \bigwedge_{1 \le p < i} \left( dep_{i,p} \rightarrow \left( c_{i,p} \vee \bigvee_{p < q < i} (c_{i,q} \wedge dep_{q,p}) \right) \right)$$

Now for $i > p$, the output $v_i$ should depend on the output $v_p$ or encode a greater linear form than $v_p$:

$$\bigwedge_{1 \le i \le k} \bigwedge_{1 \le p < i} (dep_{i,p} \vee [\psi(1,i), \ldots, \psi(n,i)] >_{lex} [\psi(1,p), \ldots, \psi(n,p)])$$

Here lexicographic comparison of formula tuples is encoded in the usual way (cf. the encoding in [3]).

While this approach eliminates some otherwise valid solutions of length $k$ and thus reduces the set of admissible solutions, obviously there is at least one solution of length $k$ which satisfies our constraints

whenever solutions of length $k$ exist at all. This way, we greatly reduce the search space by breaking symmetries that are not relevant for the result, but may slow down the search considerably.

Consider again the restriction of our S-box top matrix to the first 8 linear forms. With symmetry breaking, we can show unsatisfiability for the "hard" case $k = 12$ in 76.8 seconds. In contrast, without symmetry breaking, we cannot show unsatisfiability within 4 days.

# 4 Conclusion

In this paper we have shortly reiterated how shortest linear straight-line programs for given linear forms can be synthesized using SAT solvers. Then we have evaluated the feasibility of this approach by a case study where we minimize an important part of the S-box for the Advanced Encryption Standard. This study shows that our SAT-based approach is indeed able to synthesize shortest programs for realistic problem settings within reasonable time.

Proving optimality of the programs found by showing unsatisfiability of the associated decision problem leads to very challenging SAT problems. We have shown that breaking some symmetries of our problem significantly reduces runtimes in the UNSAT case, and using CryptoMiniSat recently we showed optimality in our case study. In future work, we consider to apply our method to other problems from cryptography. Also, we plan to further enhance our encoding and specialize existing SAT solvers to further improve performance in the UNSAT case.

# Acknowledgements

Our sincere thanks go to Erika Ábrahám, Daniel Le Berre, Armin Biere, Youssef Hamadi, Marijn Heule, Oliver Kullmann, Matthew Lewis, Lakhdar Saïs, Laurent Simon, and Mate Soos for input on and help with the experiments. We also want to thank the anonymous referees for helpful comments.

Furthermore, we thank Joan Boyar and René Peralta for providing us with information on their work and Michael Codish for pointing out similarities to common subexpression elimination.

# References

[1] R. Asín, R. Nieuwenhuis, A. Oliveras, and E. Rodríguez-Carbonell. Cardinality networks and their applications. In *Proc. SAT '09*, volume 5584 of *LNCS*, pages 167–180, 2009.

[2] J. Boyar and R. Peralta. A new combinational logic minimization technique with applications to cryptology. In *Proc. SEA '10*, volume 6049 of *LNCS*, pages 178–189, 2010.

[3] M. Codish, V. Lagoon, and P. Stuckey. Solving partial order constraints for LPO termination. *Journal on Satisfiability, Boolean Modeling and Computation (JSAT)*, 5:193–215, 2008.

[4] C. Fuhs and P. Schneider-Kamp. Synthesizing shortest linear straight-line programs over GF(2) using SAT. In *Proc. SAT '10*, volume 6175 of *LNCS*, pages 71–84, 2010.

[5] J. Giesl, P. Schneider-Kamp, and R. Thiemann. AProVE 1.2: Automatic termination proofs in the dependency pair framework. In *Proc. IJCAR '06*, volume 4130 of *LNAI*, pages 281–286, 2006.

[6] A. Kojevnikov, A. S. Kulikov, and G. Yaroslavtsev. Finding efficient circuits using SAT-solvers. In *Proc. SAT '09*, volume 5584 of *LNCS*, pages 32–44, 2009.

[7] D. Le Berre and A. Parrain. The SAT4J library, release 2.2. *Journal on Satisfiability, Boolean Modelling and Computation (JSAT)*, 7:59–64, 2010.

[8] Federal Information Processing Standard 197. The advanced encryption standard. Technical report, National Institute of Standards and Technology, 2001.

[9] M. Soos, K. Nohl, and C. Castelluccia. Extending SAT Solvers to Cryptographic Problems. *Proc. SAT '09*, volume 5584 of *LNCS*, pages 244–257, 2009.