Techniche Universität München

Internship 2014

Translation of Proofs Provided by External Provers

More Automatic Prover Support for Isabelle: Two Higher-Order Provers and a SMT Solver

May, 2. to July, 31

Author:
Mathias Fleury
ENS Rennes
Mathias.Fleury@ens-rennes.fr

Supervisor:
Dr. Jasmin Blanchette
TU München
blanchette@in.tum.de

Abstract. Sledgehammer is a powerful interface from Isabelle to automated provers, to discharge subgoals that appear during the interactive proofs. It chooses facts related to this goal and asks some automatic provers to find a proof. The proof can be either reconstructed or just used to extract the relevant lemmas: in both cases the proof is not trusted. We extend the support by adding one first-order prover (Zipperposition), the reconstruction for two higher-order ATPs (Leo-II and Satallax) and an SMT solver veriT. The support of higher-order prover should especially improve Sledgehammer's performance for higher-order goals.

Keywords: Sledgehammer, Isabelle, Leo-II, Satallax, TSTP proof, Zipperposition, veriT, proof reconstruction, higher-order proofs

Acknowledgement I would like to thank Jasmin Blanchette for the internship about a very interesting subject, and the members of the logic and verification chair for the welcome. Then Simon Cruanes and Pascal Fontaine (developer of Zipperposition and veriT) were very helpful and provided many explanations concerning their provers and bug fixes. I want to thank also Amélie Royer, Jasmin Blanchette, Nathanaël Cherrière, Martin Desharnais and Alix Trieu for having read this report and given advice to improve it.

July, 31st 2014

1 Introduction

Interactive proving has become more and more widespread, since Milner's LCF [1]. It has become more powerful especially with more automation thanks to faster decision procedures and faster computers. Interactive proving was able to verify mathematical proofs (like the four-colour theorem by Georges Gonthier [2]) or verify systems and even cryptographic protocols [3]. Much work has been done to improve the readability of the proofs and the ease to use. The interactive provers use a (as small as possible) trusted kernel, around a few thousand lines of code.

On the other side, automatic theorem proving has been largely developed since the proof of Robbins conjecture by the automated prover EQP [4]. It has become more and more powerful, and interactive theorem proving can gain much with the use of such automated programs. The automated provers are built to be fast and efficient, and represent tens of thousands of lines of code.

Although there might be bugs (because of the large code base), automatic theorem provers integration is highly interesting for interactive provers, both to discharge the user from finding trivial proofs and from remembering the name of thousands of lemmas. The integration has been done in the interactive theorem prover Isabelle, with the Sledgehammer tool:

> "I have recently been working on a new development. Sledgehammer has found some simply incredible proofs. I would estimate the improvement in productivity as a factor of at least three, maybe five."
>
> (Larry Paulson, private communication to Jasmin Blanchette)

Sledgehammer is a bridge between two worlds: on one side, Isabelle/HOL, a proof assistant and on the other side, automatic theorem proving. As you should not trust the generated proof (since there might be bugs), Isabelle/HOL tries to replay the proof using some automated tactic and the used facts: the automatic provers is only used as a fact filter. But the performance for higher-order goals were not very good:

> " 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 an actual higher-order theorem prover."
>
> (Larry Paulson in [5])

To solve this problem two higher-order provers Leo-II and Satallax have been integrated to Isabelle/HOL to find the needed facts: there were no proof reconstruction support and the overall performance is not as good as expected, since the used tactics are not very powerful with respect to higher-order logic. This work adds proof reconstruction support for different provers: Zipperposition, veriT, Leo-II and Satallax in Isabelle/HOL. After some background information (section 2) and related works (section 3), we will then see the three first provers whose proof output are similar to each other (section 4). Finally, we will introduce the last prover Satallax, study its proof and the integration in Isabelle/HOL (section 5).

2 Context and Motivation

In this part, we will outline the different technologies used: Isabelle, automatic provers and Sledgehammer.

2.1 Isabelle

Isabelle [6] is a generic framework for interactive theorem proving: the use of a meta-logic *Isabelle/Pure* allows a formalization for object logics. It is based on the ideas of LCF [1]: every proof goes through a trusted kernel. It is written in SML, and has an Isabelle/jEdit interface through the asynchronous PIDE interface [7], allowing parallelism in the execution of the proofs [8].

HOL *higher order logic* [9] is the most developed object logic in Isabelle: it is based on simple type theory, Hilbert choice, polymorphism and axiomatic type classes. Isabelle/HOL's term language is composed of typed λ-terms, with constants and polymorphic types. Functions can be curried $(f\ x\ y)$, and usual binary operators can be used with the infix notation. The main proof method is the *simplifier*, which uses an

equation as oriented rewrite rules on the goals, including contextual and conditional rewriting (like rewriting $ys = Nil \to \text{rev } ys = z$ into $ys = Nil \to \text{rev } Nil = z$ and after that, using the lemma that says $\text{rev } Nil = Nil$ to get $ys = Nil \to Nil = z$). The list of lemmas that are used can be extended by the user's lemmas.

Other object logics includes Isabelle/ZF (based on Zermelo-Fraenkel axiom system) and Isabelle/HOLCF (based on Scott's domain theory, formalized within HOL) [10, 11]. Isabelle has been used for pure mathematics (the Prime Number Theorem[1] verified by Avigad [12]), for system verification (the seL4 micro-kernel [13]), and programming languages (the formalization of Java [14]).

Isabelle/HOL has also an impressive library of proofs, the Archive of Formal Proofs *AFP*. The aim is to provide a "large resource of knowledge". Most of the proofs are written in the Isar language proof: it is designed to produce human understandable proofs [15].

As of now, Sledgehammer is a part of Isabelle/HOL, thus we will now write Isabelle instead of Isabelle/HOL. Isabelle is built around tactics that transform the goals like `simp`, we have described before, including `metis` (it is the built-in resolution prover in Isabelle [30]).

2.2 External provers

There are two types of external provers: *automatic theorem provers* (ATP) and *satisfiability modulo theories* (SMT) solvers; we will distinguish them for historical reasons, but both of them are trying to prove automatically. If we see them as black-boxes, they work in the same manner: they take facts and the conjecture. The latter is negated, and then by combining the facts and the negated conjecture, a proof of $\bot$ (false) is found: this is a proof by contradiction. They are built to be *fast* and *efficient*: thus not using a small trusted kernel, contrary to Isabelle's tactics. ATPs have a more elegant approach for quantifiers, while SMT solvers are faster for ground problems with a high number of facts: both are complementary. Thus using both is quite interesting for a Sledgehammer user. Thankfully both are run in parallel, allowing the user to ignore the difference between the two kinds of systems.

The ATPs world [16] is built around *Thousands of Problem for Theorem Proving* (TPTP). It is not only a vast library of problems [17], but also specifications, hence allowing simpler interactions from Sledgehammer to all of them. Internally the well-known first-order ATP prover E [18], SPASS [19], Vampire [20] are using superposition: they combine the facts trying to get a contradiction from both known and deduced facts [21]. Recently some ATPs have gotten support of higher-order theorems like Leo-II and Satallax.

SMT solvers combine SAT solvers and decision procedures for first-order theories like equality and arithmetic (contrary to ATP solvers). They are especially strong as fact filters: given many facts (a few hundreds), they find efficiently the few needed facts (often less than a dozen). Famous provers include CVC4 (a full rewrite of CVC3) [22], Yices [23] and Z3 [24]. SMT solvers are not as organised as the ATPs, and while there is an input format accepted by nearly all provers (the SMT-LIB format [25]), there is no output standard, though some have been proposed [26, 27]. That can be explained, since the aim of most of them is to find a proof, not to ensure that the proof can be checked, but this is unfortunate for us.

2.3 Sledgehammer

Let us first see a detailed example of a typical Sledgehammer call. First the prover wants to prove that `rev [a, 1 + 1] = [2, a]` where `rev` is the function to reverse list. Then we can call the `sledgehammer` command. First all the lemmas, definition, axioms,... in the Isabelle theories are listed: all these facts are filtered to find those related to the goal, including for example `rev.simps(2)`: for any `x` of type `'a` and `xs` of type `'a list`, `rev (x # xs) = rev xs @ [x]` (`#` is the `cons` operator, and `@` the infix append function). Then these and the goal are translated into the TPTP format in parallel for the provers E, SPASS by default, while there is also a translation for the SMT solver Z3. The provers are started in parallel. When they find a proof, the proof is parsed and depending on the prover and the option, we get either a simple `metis` call (here `by (metis rev_append rev_eq_Cons_iff rev_rev_ident)`) or a full Isar proof (Isabelle proof language). As ATPs do not know arithmetic, E produces a proof including `one_add_one` (`1 + 1 = 2`): the `metis` call is

[1] The number of primes under n, $\pi(n)$, satisfies $\pi(n) \sim \frac{n}{\ln n}$.

`by (metis append.simps one_add_one rev.simps(2) singleton_rev_conv)`, while Z3 does not need it, since it supports arithmetic. Here is the Isar proof produced by Z3:

```
proof -
  have "rev [a, 2] = rev [2] @ rev (rev [] @ [a])" using rev.simps rev_eq_Cons_iff by auto
  hence "rev [a, 2] = [2, a]" using rev_eq_Cons_iff by fastforce
  thus "rev [a, 1 + 1] = [2, a]" by auto
qed
```

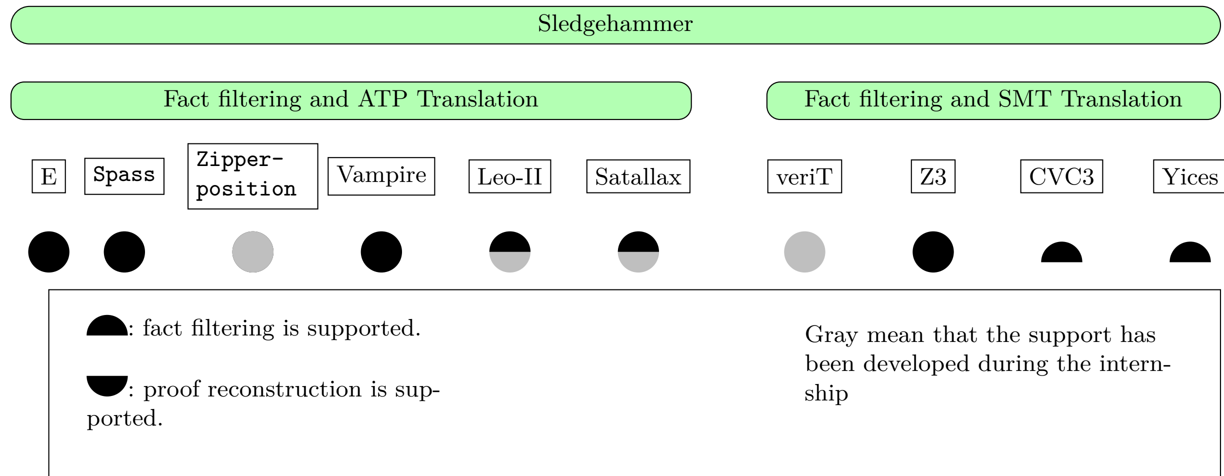

Fig. 1: Sledgehammer organisation with reconstruction support with some of the supported provers

Sledgehammer tries to find a proof by calling an external (untrusted) automated theorem prover. First, the user gives a goal to prove. Then Sledgehammer will filter some facts related to the goal (see figure 1 on this page). Until recently this fact choice was based on the proximity between the lemma and the goal: the more symbols are shared, the more likely the fact were chosen. That works but a better approach has been developed recently, based on machine learning [28]. ATPs and SMT solvers are pretty much the same thing: both try to deduce contradiction from the hypotheses, but they are originating from two different communities. For historical reason, Sledgehammer distinguishes both: at the beginning there was no support for arithmetic in ATPs, whereas SMT solvers do support it, thus the latter do not need lemmas like `1 + 1 = 2` (it is a tautology for them), but ATPs do not know them.

The number of facts that are translated depends on the prover: as a user you want the provers to have as many facts as possible, but the more facts, the less efficient they become (because the search space increases). Thus, a compromise has to be found (see for example [29] for Leo-II and Satallax). After that the facts are translated to a format supported by the prover. There are two formats depending on whether the prover is an SMT solver or an ATP prover.

We will distinguish the *facts* (Isabelle's theorem that are exported) from the *hypotheses* (the hypotheses of the theorem, that the user wants to prove) and the negated conjecture. For example if we want to prove that under the assumptions even p and prime p, we have $p < 3$, even p is one of the hypothesis, the negated conjecture is $\neg(p < 3)$, and one related fact is for example that prime $p \to p > 2 \to$ odd p (called `prime_odd_nat` in Isabelle).

The prover uses the negated conjecture, the hypotheses and the facts and tries to derive false. If they obtain it, the output depends on the prover:

– the prover can give all facts (and not only the needed facts), like CVC3, either because it cannot output the proof (i.e. only outputs `sat` or `unsat` — unprovable or provable) or produces an unsupported proof.

Then the needed facts are minimized to find out only those needed: the prover is restarted with only half of the facts, and so after a few steps, we can get only the used facts. Minimizing is not only interesting in that case, but can also be interesting for all provers, since they often use more facts than actually needed.

- it can give only the needed facts (called *unsatisfiable core*). Then, because the provers often use more facts than needed, Sledgehammer tries to minimize the number of facts as described before. This is slightly faster than the previous approach, since we are minimizing a few facts, and not a few hundreds.
- the prover can also give a real proof, like Z3. Then the proof can be translated in Isar, or we can extract only the needed facts.

Most provers that can output a proof can also output the unsatisfiable core, like Satallax. The core can easily be deduced from the proof, since it makes no sense to be able to reconstruct a proof without finding the facts, because the proof needs the facts. Whether printing the proof depends on Isabelle support for the given format.

Given the facts, a `metis` proof is then tested by giving all the needed facts as arguments: `metis` is the built-in resolution prover in Isabelle [30] (given an infinite amount of time and memory, it is complete for a first-order fragment of higher-order logic). Sometimes the replaying fails or is quite long, especially for higher-order goals. Then a reconstructed proof is the only way to get an Isabelle proof.

Before being proposed to the user, the proof is redirected (to obtain a direct proof, and not a refutation proof) and compressed (provers often give much more details and proof steps, than needed by Isabelle proofs' tactics) [31], thus allowing shorter proofs and for the user, easier to understand.

Sadly there are a few problems for reconstructing proofs: some provers give no proof at all. Others give only a proof sketch that is useless, and does not even contain enough information to find out which facts have been used, like CVC4. What's more, they use a proof format LFSC [27], that does not remind the proof term (the term that is produced at each step), thus forbidding any reconstruction without "reverse-engineering" the proof. Contrary to their input, there is no standard format and every SMT prover uses either its own, either do not produce proofs. Often there is no documentation about the meaning of the proofs. This is specially important when Satallax interprets the TPTP format on its own very different way (see section 5 on page 8).

2.4 Motivation

For higher-order logic, the extracted facts are often not enough to find the proof, as both tactic used to find the proof are incomplete and inefficient for higher-order functions. A single example: with the definition of $\circ$, $f \circ g = \lambda x.\ f(g\ x)$, `metis` is unable to prove that $P\ (f \circ g)\ (\lambda x.\ f(g\ x)) = P\ (\lambda x.\ f(g\ x))\ (f \circ g)$ where P, f and g are some function with the correct type.

One of the solution is to reconstruct the proof step-by-step in Isabelle, instead of extracting only the facts: "reconstructing" means to output an Isabelle proof with the steps, instead of a call to a tactic using only the needed facts. As the steps are little enough, thus at least one of the tactics should work. If the reconstruction fails, it means that there is a bug in the reconstruction or that there is a bug in the prover; while if the call to the tactic with the facts fails, it does only mean that the tactic is not powerful enough (or that you have not waited long enough). Adding new provers is also interesting, since different provers can have different strengths, thus allowing the user to care less about the details, and more about the proof globally. A simple proof like the one above is compressed in a one-line proof (not calling `metis`): different tactics work (here `simp`), but in longer proofs, they are not powerful enough.

Now we have introduced the background information needed to understand what we are working on, we can speak about the new support of three provers, after describing the other work that exists on the subject.

3 Related Work

3.1 In Isabelle

Our work to add new provers is closely linked to Sledgehammer (as you can see on figure 1 on the preceding page), that has been developed by Larry Paulson at Cambridge and then expanded at Munich. Some ATPs

have been added and are even part of the Isabelle distribution like E and SPASS. As some provers provide an internet access to a server like Vampire: it allows their usage without having to install them.

On this ATP side, many first-order solvers are supported for both reconstruction and fact filtering. Our work was adding proof reconstruction for the higher order provers Leo-II and Satallax, since only reconstruction for first order provers was supported before.

Some SMT solvers have already been integrated like CVC3, CVC4 and Yices, but only the Z3 proof output is supported for reconstruction (because the others have either no proof output, or the reconstruction is too hard). We added the support for the SMT solver veriT, Böhme had already added the support for the SMT solver Z3 [32]. It is important to notice that integrating ATPs is very different from integrating SMT solvers, e.g. because of the support for arithmetic.

3.2 In other provers assistants

HOL(y)Hammer[33] (inspired by Sledgehammer) and Sledgehammer have the same aim: being fully automatic (no work from the user), constructing a proof (no assumption that the prover is correct) and source code delivery (after finding a proof, there is no need to have the solver installed). As far as we know, these are the only work with these goals.

*Ω*mega is another proof assistant that has been expressively developed for proof planning and support for theorem translation and proofs translation from one system to another. Sledgehammer has inspired Miz$\mathbb{AR}$ for Mizar (based on Vampire and `SInE`) [34], and the integration of the equational prover `Waldmeister` in Agda has been developed [35]. On the SMT side, PVS uses Yices as an oracle (i.e. trusting the tool). HOL Light integrates CVC Lite and reconstruct the proof. There is work to support the SMT solver veriT in Coq. Satallax is able to translate the proof into a Coq script: the main difference is that Satallax reconstruct the proof itself. The interactive theorem prover should reconstruct the proof and not the contrary, since each automatic prover would have to make a translation to every possible interactive prover.

4 Adding a new Prover: a simple Case

While we have said before that SMT solvers do not have a common proof format contrary to ATPs, the situation is not that simple: TPTP defines a *guideline* and ATPs can interpret this differently: they do not share a semantic, that would allow us to verify the proofs of all of them (see section 5 on page 8 for Satallax proofs). In this part, we will give general ideas about how to add a prover to Isabelle and some special rules used.

4.1 Presentation of the three provers

Before speaking about the support in Isabelle, we will present them briefly.

Zipperposition[2] is a recent prover developed by Cruanes. It is a superposition prover written in OCaml and had participated to the CADE 2013 (Conference on Automated Deduction) competition and performed correctly (for a prover written for experimenting, rather that speed): it solved 91 of the 300 problems (Vampire solved 281) at the 2013 CADE ATP System Competition [36]. The prover is able to produce a standard TSTP output. Since the competition, some code has been rewritten and at the beginning we have found some bugs in the prover: the prover derived false from a consistent set of axioms because of unsound inferences. These bugs have since been corrected, after we contacted the author.

Leo-II[3] is a higher-order resolution solver: it works on higher-order clauses and it calls periodically a first-order prover, usually E, that tries to find out a proof of false with the first-order clauses. It implements a relevance filter if there are more than one hundred axioms. It is developed by Benzmüller, Theiss and Sultana in OCaml. Some work has been done to explain the proof format and the applied rules by Sultana

[2] Available at `https://www.rocq.inria.fr/deducteam/zipperposition/`.
[3] Available at `http://page.mi.fu-berlin.de/cbenzmueller/leo/`.

and Benzmüller [37]. Leo-II took part at the 2013 CADE ATP Competition (in a different class than Zipperposition): it solved fewer goals than Satallax (76 out of 150, while Satallax that we will study in the next section, solved 119).

veriT is an SMT solver developed by P. Fontaine et al. [38] at the LORIA in Nancy. Similarly, to Leo-II, it uses a first-order logic solver, that is periodically called. As an SMT solver it supports arithmetic. At the 2014 SMT contest, it performed the best (but Z3 did not take part to that contest), and did not mark false theorem as proved contrary to some other provers (like Yices). It is programmed in C++ (around 67 000 lines of code).

4.2 ATPs: Leo-II and Zipperposition

While both Leo-II and Zipperposition are ATPs, they do not use the same output format, since Leo-II is higher order, while Zipperposition is not.

TPTP language hierarchy The TPTP infrastructure defines a hierarchy of languages, starting from first-order form FOF (first-order logic, with equality over untyped types), over typed first-order form TFF0, to typed higher-order form THF0. There is a strict inclusion FOF $\subsetneq$ TFF0 $\subsetneq$ THF0, except for few minor syntactic differences. THF0 types can be *type constants* κ or functions between type constants $\sigma \to \tau$. The intended semantics is Henkin's with extensionality and Hilbert choice. Zipperposition uses TFF0 while Leo-II THF0. Both format are supported by Sledgehammer since Blanchette's work on supporting Leo-II and Satallax (without proof reconstruction).

Output format The output format is a standard, and both use a common interpretation of the standard. The output is a proof starting from the facts and goes to the conclusion (see 5.2 for a different structure). As this is the most used way for proofs, Sledgehammer supports it, if the output is parsed and transformed with respect to the prover's peculiarities. Moreover, the output is partly *typed* like `p=^[X1:int]:X1`: all the types have not to be reconstructed to be accepted by Isabelle.

As the input format is different (not the same things have to be expressed), so is the output format, since one use higher-order logic wile the other does not. That changes the output (THF, TFF), but the difference is in the term encoding (mostly), and not in the proof output. The output is like that:

⟨*step*⟩ ::= ⟨*description*⟩(⟨*name*⟩, definition, ⟨*proof term*⟩, file(⟨*file access*⟩, ⟨*fact_number*⟩)).
  | ⟨*description*⟩(⟨*number*⟩, ⟨*role*⟩, ⟨*proof term*⟩, ⟨*dependency list*⟩*).

⟨*description*⟩ ::= 'cnf' | 'thf' | 'tff'

where ⟨*description*⟩ depends on the format. The proof is done by contradiction, thus the final conclusion is always $\bot$.

In Isabelle After parsing the higher-order terms, like the following: `p=^[X1:int]:X1` corresponding to $p = \lambda x :: \text{int}.\ x$, Isabelle has a standard reconstruction for ATPs proofs as said above. Most steps can be replayed with `metis` and most of the time, more than one step is done at a time ; but as described above `metis` is not complete for higher-order logic, thus other tactics might have been called.

We have to cope with the ATPs' bugs or unintended behaviours for the reconstruction:

- Zipperposition writes the steps in the opposite direction than all others: we had to reverse them;
- Leo-II does not write all the labels correctly: `thf(comp, definition, (comp = ...), file('prob_leo2_1', comp)).` while the name of the fact was given in the following definition: `thf(fact_0_o__def, definition, ((comp = ...))).`
- Leo-II is higher order and uses E to find contradiction: sometimes E finds a proof, but Leo-II is not able to give the facts needed. In this every single step is given as an assumption to the final $\bot$-step. That leads to more complex proof after the reconstruction, or even steps, that are impossible to show with an

Isabelle tactic: in this case, *all* the assumptions are marked as used, while this is wrong. This is also a fact in favour of minimisation: as we restart the provers with only half of the needed facts, there might be no bug in the interaction between Leo-II and E.

That is all we have to do to add the support for a new prover in Isabelle, thanks to the standard output format. The work to do is very different for SMT solvers.

4.3 SMT solver: veriT

veriT's output format has been proposed as a standard in [26], it is thus well described. The information included in the output format are close to that of the TPTP format:

$\langle step \rangle$ ::= (set $\langle label \rangle$ $\langle rule \rangle$:clause($\langle$*dependency label*$\rangle$*) :args($\langle$*arg*$\rangle$*) :concl($\langle$*proof terms*$\rangle$))).

The proof step is close to the THF0 format with more or less the same information: $\langle label \rangle$ is of the form `.c` followed by a number or if it used as a dependency it can also be the name of one the facts of Sledgehammer. $\langle rule \rangle$ is the applied rules to get the conclusion using the assumptions. $\langle$*dependency label*$\rangle^\star$ is the list of used assumptions. It can be either one of the introduced clauses or one the facts. Contrary to Z3, veriT uses the name of the fact defined by the `:name`-tag defined by the SMT-LIB2 format. If there are not named, then the used facts are reminded at the beginning. As our facts are named, we can avoid recognizing the term to find out which facts are used. `:args` is the information about how variables are bound after an instantiation. This information is not used in by Isabelle. `:concl` is the conclusion by the application of the rule ⟨`rule`⟩ on the assumption. It can be empty (then it means `false`) or it can be a list of different formulae: as internally, veriT uses multi-sets, the $\vee$ are not always written (the rewriting is done by the `or`-rule, but can also be done when hypotheses are recalled).

This work is related to Böhme's reconstruction of Z3 proofs [32]: both works are linked to reconstructing the proof output of an SMT solver, but the output format are quite different. The proof formulae are written in the same format, so that we could reuse part of Böhme's code in Isabelle, including his work on finding the types (the output is untyped, but Isabelle needs the type information).

As every single step is a very little step, most rules can be compressed and several steps can be reconstructed by a single `metis-` call in Isabelle. Some rules have to be treated in a special manner:

ite_elim: transforms the facts like ($\texttt{if}\ x\ \texttt{then}\ u\ \texttt{else}\ v$) $= z$ into $\alpha = z \wedge \texttt{if}\ x\ \texttt{then}\ \alpha = u\ \texttt{else}\ \alpha = v$. This step can be either an introduction for a new variable or the reuse of an existing variable: in the first case, we have to tell Isabelle that this is a skolemisation, in the second case, we have to add the dependency to the step that introduced the variable.

sub-proofs: they are described in [39]. Subproofs are like proofs a logical set with assumptions, proof steps and a conclusion, here is an example:

```
(set .c15
  (subproof
    (set .c1 (input :conclusion ((=> (p 0 @sk0) (p (ite (<= 0 1) 1 0) @sk0)))))
    (set .c2 (tmp_ite_elim :clauses (.c1) :conclusion
      ((and (=> (p 0 @sk0) (p @ite0 @sk0)) (ite (<= 0 1) (= @ite0 1) (= @ite0 0))))))
  :conclusion
    (or (not (=> (p 0 @sk0) (p (ite (<= 0 1) 1 0) @sk0)))
        (and (=> (p 0 @sk0) (p @ite0 @sk0)) (ite (<= 0 1) (= @ite0 1) (= @ite0 0))))))
```

There is `ite_elim`, we have described before. As Sledgehammer does not support them (because they are not present in any other prover), we inline the assumption (here `.c1` in `.c2`).

This kind of subproofs sometimes appears especially related to the `ite_elim` rule. The assumptions have to be discharged when the subproof is used (more precisely, the prover uses this just in that way). The main problem is that Isabelle's Sledgehammer does not support those subproofs. Our solution was to inline the

assumption, removing the now unused assumption and renumber this steps. Remark that the `:conclusion` of the subproof is correct unconditionally, (i.e. without having to inline the assumptions).

All the provers of this section (Leo-II, Satallax and veriT) share two important properties: the proof terms are written (and we need this to be able to reconstruct an Isar proof), and the proofs start with the assumption. Not all provers work like that: Satallax is another higher order prover, but works quite differently.

5 Understanding Satallax Proofs

Satallax[4] and Leo-II are two higher order ATP provers, but their proof output is very different. We have not described the proof produced by Leo-II in details, but we have seen that it is the same as other ATP solvers.

5.1 Satallax

Satallax [40] is a higher-order prover based on Church's simple type theory with extensionality and choice operators. It uses the *tableau* [41] method internally. Like Leo-II, it generates propositional clauses and does periodically call a first-order prover (recently E [18]). If the first-order prover finds a refutation, we obtain a proof. It is programmed in OCaml (around 18 000 lines of code). Satallax can produce TSTP-like proofs (but that does not follow the other ATP provers' format rules), or Coq proofs.

Satallax and Satallax-`MaLeS` (the same code base, but the strategy scheduler is based on machine learning) won the two first place at the 2013 CADE ATP System Competition [36], while Leo-II was fifth place, behind Isabelle that internally does *not* call Satallax and Leo-II through Sledgehammer.

5.2 Satallax proofs

Forward and backward proof Satallax proof works like a Coq proof *backwards*[5], contrary to (as far as we know) all other prover's TSTP output: instead of going from the hypothesis to the conclusion, the backward proof goes from the conclusion (here `false`), and introduces hypotheses. Then it has to prove $\bot$ *with* these newly introduced hypotheses. To explain the difference in a more formal way:

forward proofs: you want to show that $[A_1, \ldots, A_n] \to G$ (G is the goal, $A_1, \ldots, A_n$ are the assumption), you also know that $[A_i, \ldots, A_j] \to H$, thus we add H to our hypothesis and it remains to show that $[A_1, \ldots, A_n, H] \to G$. The proof is finished when G is added to the assumptions. This is the kind of proofs produced by Leo-II.

backward proofs: we will use a different representation for backward proof: instead of $[A_1, \ldots, A_n] \to G$, we will write $\dfrac{A_1, \ldots, A_n}{G}$, but it means the same (we have to prove G under assumptions $A_1, \ldots, A_n$). If you know for example that $[B_1, \ldots, B_j] \to G$, thus it remains to show j subgoals: $\dfrac{A_1, \ldots, A_n}{B_1}$, ..., $\dfrac{A_1, \ldots, A_n}{B_j}$. This is the kind of proof produced by Satallax (probably because of Coq).

For example, to prove that $2 < 3$ with a forward proof: if we use Peano's integer construction and write S for the successor and define inductively $<$ as: $A := 0 < \mathrm{S}_$ is true and $B := \forall (m, n) \in \mathbb{N}.\ n < m \to \mathrm{S}\, n < \mathrm{S}\, m$. The assumptions are $[A; B]$:

$[A; B]$:	as $A \to 0 < 1$, we can add $0 < \mathrm{S}\,0$ to the hypotheses
$[A; B; 0 < \mathrm{S}\,0]$:	as $B \to 0 < \mathrm{S}\,0 \to \mathrm{S}\,0 < \mathrm{S}\,\mathrm{S}\,0$, we have new assumptions
$[A; B; 0 < \mathrm{S}\,0; \mathrm{S}\,0 < \mathrm{S}\,\mathrm{S}\,0]$:	finally, as $B \to \mathrm{S}\,0 < \mathrm{S}\,\mathrm{S}\,0 \to \mathrm{S}\,\mathrm{S}\,0 < \mathrm{S}\,\mathrm{S}\,\mathrm{S}\,0$
$[A; B; 0 < \mathrm{S}\,0; \mathrm{S}\,0 < \mathrm{S}\,\mathrm{S}\,0; \mathrm{S}\,\mathrm{S}\,0 < \mathrm{S}\,\mathrm{S}\,\mathrm{S}\,0]$:	$2 < 3$ is true.

In contrast for the backward proof using the same formalism: we start with the goal $\dfrac{A, B}{\mathrm{S}\,\mathrm{S}\,0 < \mathrm{S}\,\mathrm{S}\,\mathrm{S}\,0}$:

[4] Available at `satallax.com`.

[5] The word "backwards" sometimes refers to proof by contradiction, but it is not the case here.

1 $\dfrac{A, B}{\mathrm{S\,S\,0} < \mathrm{S\,S\,S\,0}}$: as we know that $B \to \mathrm{S\,0} < \mathrm{S\,S\,0} \to \mathrm{S\,S\,0} < \mathrm{S\,S\,S\,0}$, we have two new goals, numbered 2 and 3

2 $\dfrac{A, B}{B}$ is straighford because B is an assumption: it remains to show that 3 is true

3 $\dfrac{A, B}{\mathrm{S\,0} < \mathrm{S\,S\,0}}$ as $B \to 0 < \mathrm{S\,0} \to \mathrm{S\,0} < \mathrm{S\,S\,0}$, we have two new goals, numbered 4 and 5

4 $\dfrac{A, B}{B}$ same proof as before, it remains to show that 5 is true

5 $\dfrac{A, B}{0 < \mathrm{S\,0}}$ is true thanks to A: no goal remains, the proof is finished.

The main difference between the two proof methods is that it is easy to produce case distinction in the backward mode. For example if you assume $A \lor B$ to show G; you only have to prove G once under the assumption A and once under the assumption B. To express this in a forward proof style, we have to prove it $A \to G$ and $B \to G$: it is a bit less natural, since for backward proof it is a deduction rule, while we have to change the goal for forward proofs.

Proof representation We introduce now a representation of a proof based on graphs.

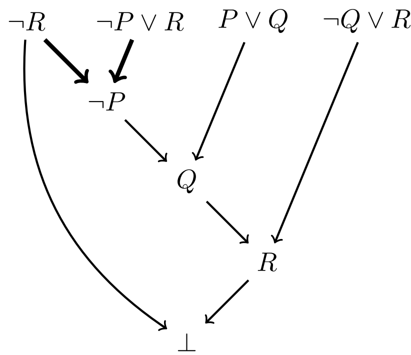

Fig. 2: Example of a forward proof

First, forward proofs (without case distinction, since Sledgehammer does not support them) can be represented as a dependency graph (it is a directed acyclic graph) where the nodes go from the hypothesis to the new hypotheses.

For example, let's see a proof that $[\neg R, \neg Q \lor R, P \lor Q] \to \neg(\neg P \lor R)$. As we are proving by contradiction, it is the same as showing that $[\neg R; (\neg P \lor R); P \lor Q; \neg Q \lor R] \to \bot$ (backward and forward are not linked with contradiction or not, but we are speaking of automatic provers that always prove by contradiction).

In this graph, the thicker arrows means that using the two known facts $\neg R$ and $\neg P \lor R$, we have $\neg P$. A fact can be used more than only once.

This is the kind of representation that is supported in Sledgehammer: to represent this kind of proofs, in each step you have the term, the step number and the dependencies. It is also the most natural way to write proves, starting from the assumptions and going to the conclusion.

We have not here indicated the rule that is applied on each arrow since it is simple first order logic; but in the general case, this rule is needed for the reconstruction. Remark that not all the assumptions have to be used: the prover is called with many facts and not all are useful.

Backward proofs cannot be represented that way: the destination cannot be $\bot$. We will use a representation as a directed graph, close to the representation from the tableau method: the nodes are the fact and the conclusion, and the arrows are the proof steps. As we can introduce new subgoals several arrows can start from a given node, see figure 3. The arrows are labelled by the used rule. The different subgoals are numbered from 1 to 7. We apply here two different rules:

(i)**:** $(A \lor B) \to (A \to \bot) \to (B \to \bot) \to ((A \lor B) \to \bot)$
(ii)**:** $(\neg (A \land B)) \to (\neg A \to \bot) \to (\neg B \to \bot) \to (\neg (A \land B) \to \bot)$

We have simplified a bit contrary to our previous example: as $A \lor B$ is *always* in the assumptions (otherwise applying the rule make no sense), we have not written the goal $\dfrac{\mathcal{H}'}{A \lor B}$ for some hypothesis $\mathcal{H}'$ where $A \lor B$ is included in $\mathcal{H}'$. As you can see, on each step we introduce new assumptions. The label on the left of the goal means is the rule on which we are applying (i), the label on the right are the contradicting hypotheses. At each step we introduce the name of the new assumption to be close to the proofs generated by Satallax (given in annex B).

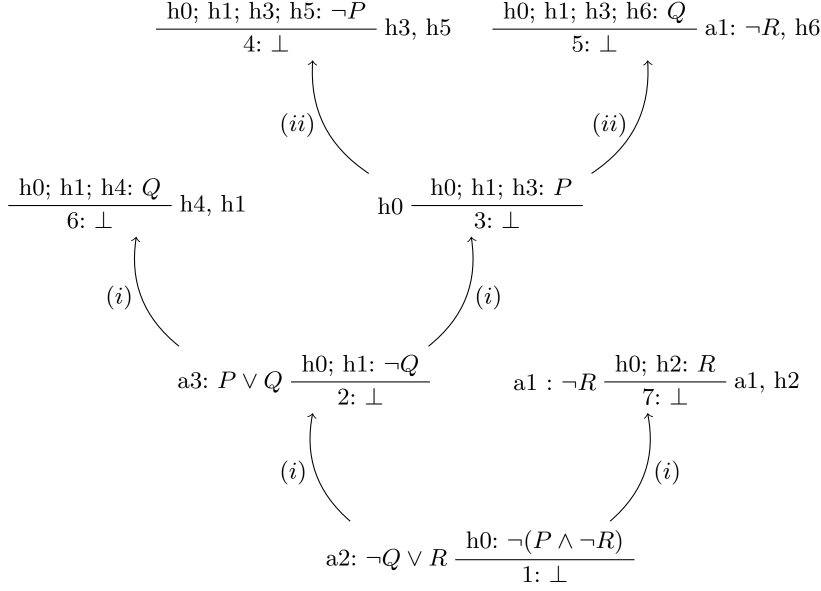

Fig. 3: Backward proof example

Organisation of the backward proof As Satallax is using a backward approach, each step adds a new hypothesis and changes the goal (although `false` is *always* the goal). In the TSTP proofs generated, we will distinguish two kinds of steps: the hypothesis and the goals. Hypotheses are the steps with identifier beginning with `h`, like `h1`, `h2` and so on. These steps are marked with `assumption` or `axiom`. The subgoals are numbered 1, 2,... They contain the indication about the proof itself, as we will describe it. There is a third type of steps: the types indicating that the type is inhabited (necessary for creating variables bound by ∃). There seems to be no rule in the order of appearance in the two types of steps. The goal 1 is the goal we want to prove, from this the goal 2 is introduced, and so on. They are in reversed order (i.e. 1 is at the end). Satallax even recall as the step 0 the theorem it has proved.

TSTP proofs The assumption steps are of the following form:

⟨*assumption_steps*⟩ ::= ⟨*hypothesis_step*⟩ | ⟨*axiom_step*⟩

⟨*hypothesis_step*⟩ ::= thf(⟨*hypothesis rule number*⟩, assumption, ⟨*proof term*⟩, introduced(assumption, [])).

⟨*axiom_step*⟩ ::= thf(⟨*fact_name*⟩, axiom, ⟨*proof term*⟩).

The ⟨*rule number*⟩ is of the form `h` followed by a number (as said above). The ⟨*proof term*⟩ is a higher order term following the THF0 norm. Here is an extract of the example of assumptions steps that Satallax can produce (`~` means not, `&` and, `|` or). The proved theorem is reminded as step 0 (see annex B for the full proof):

```
thf(conj_0,conjecture,(p & (~(r)))).
thf(h0,negated_conjecture,(~((p & (~(r))))),inference(assume_negation,
   [status(cth)],[conj_0])).
thf(h1,assumption,(~(q)),introduced(assumption,[])).
thf(a1,axiom,(~(r))).
```

The proving steps follow the same format, but look different:

⟨*proving_step*⟩ ::= thf(⟨*proving_step_number*⟩, $false, inference(⟨*rule name*⟩), [⟨*status*⟩, assumptions(⟨*hypothesis rule number*⟩*), ⟨*rule list with assumption*⟩, ⟨*optional extra arguments*⟩], ⟨*premises list*⟩).

where:

⟨*proving_step_number*⟩ is the number of the goal.

⟨*rule_name*⟩ is the name of the rule that is applied. It does only give information about how to get the hypothesis, the precise rule is not stated here.

⟨*assumption_list*⟩ is the list of all the hypotheses of the current goal.

⟨*rule_list_with_assumption*⟩ this contains the rules applied to get the contradiction. It is a list of element of `rule(discharge,[used_assumptions])` like elements. The rule indicates what is deduced from the ⟨*used_assumption*⟩. The rules are described below in section 5.4.

⟨*premises_list*⟩ gives a lot of important information. It can be divided into three parts: the identifier of the used hypotheses, the number of the generated subgoals and the new hypothesis. First, there are some hypothesis steps' name or a theorem. This gives the rule that is applied or the assumption on which work is done; in that case it also contains information on how the variable are bind: `h0:[bind(X1,$thf(a))]` means that `X1` of (for example) `h0` will bind as `a` in the next step. Then there are the number of the new generated subgoals, and finally the number of the new hypotheses

It is important to remember that sometimes there is no new goal nor new hypothesis generated. More than one single goal can also be generated.

An extract of the proof steps of the previous example are the following (see annex B for the full file):

```
thf(1,plain,$false,inference(tab_or,[status(thm),assumptions([h0]),
  tab_or(discharge,[h1]),tab_or(discharge,[h2])],[a2,2,7,h1,h2])).
thf(0,theorem,(p & (~(r))),inference(contra,[status(thm),contra(discharge,[h0])],[1,h0])).
```

Let's see for example the step 1: the number of the step is 1, the role (as defined by the TPTP format) is `plain`, the theorem is `$false`, the rule to produce the new subgoals is `tab_or` and this produce two new hypotheses as described in `tab_or(discharge,[h1]),tab_or(discharge,[h2])`. In the premises list there are three parts: the assumption on which we apply the rule, here `a2`, the rule generates two new subgoals `2` and `7`, the rules generates two new assumptions `h1` and `h7`.

The last type of steps is of the following form:

⟨*inhabited_step*⟩ ::= tab_inh(⟨*type*⟩) ___⟨*number*⟩.

For example `tab_inh(a) __11.` means that the variable `eigen__11` that had been created before of type `a` has a direction, since we cannot create variable in an uninhabited type. As one cannot define an empty type in Isabelle, we can safely ignore this kind of steps.

5.3 Proof transformations

Now we have understood the proofs, we need to transform the backward proof into forward proof. One possible solution is to inline all the assumptions and to change the sense of the proof: instead of creating a subgoal, we assume it. And in each step we inline every assumptions $\dfrac{A, B, C}{\bot}$ will become $A \to B \to C \to \bot$. See the figure 4 below, for the example described above.

This kind of proof is hard to read, hard to understand and hard to maintain. Moreover, replaying the proof step will become hard when there are many assumptions, especially since it is hard helping Isabelle to find the contradiction.

To simplify the proofs, the first remark is that we do not need to inline every assumptions, but only the one that are needed. An assumption is used in a step if it is used by the rule, i.e. if it is part of the premises list. More generally an assumption is needed, if it is part of the assumptions and if it is used in either the step itself or one of the produced subgoals. This produces a simpler proof with less implications in it (see in

Number	Term	Dependencies
4	$h0 \to h1 \to h3 \to h5 \to \bot$	
5	$h0 \to h1 \to h3 \to h6 \to \bot$	a1
3	$h0 \to h1 \to h3 \to \bot$	4, 5
6	$h0 \to h1 \to h4 \to \bot$	
2	$h0 \to h1 \to \bot$	3, 6, a3
7	$h0 \to h2 \to \bot$	
1	$h0 \to \bot$	3, 7, a2
0	$\bot$	$h0$, 1

Number	Term	Dependencies
4	$h3 \to h5 \to \bot$	
5	$h6 \to \bot$	a1
3	$h3 \to \bot$	4, 5
6	$h1 \to h4 \to \bot$	
2	$h0 \to h1 \to \bot$	3, 6, a3
7	$h2 \to \bot$	a1
1	$h0 \to \bot$	3, 7, a2
0	$\bot$	$h0$, 1

Fig. 4: Forward proof corresponding to the proof described in figure 3. On the left the most naive version where every assumption have been inlined, and on the right the version where only the needed assumptions are inlined.

figure 4). But we can do better since for example we do not need to inline $h0$ since we know that it is true. We do not change the semantic of the proof, since we have removed only the assumptions we do not use.

In the general case, the proofs are composed of two parts, the *linear parts* (thicker arrows) and the *split part*.

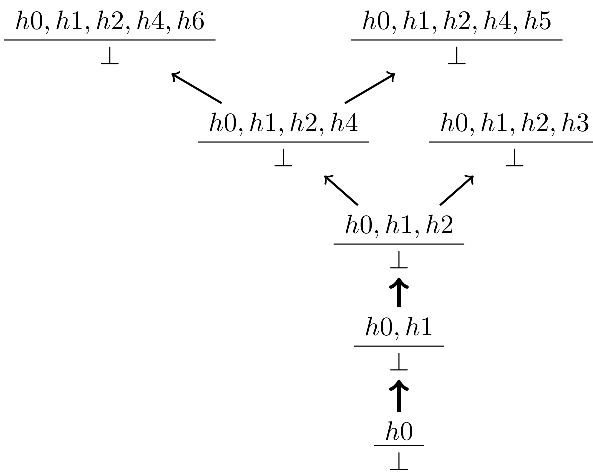

Fig. 5: General form of a backward proof

The linear part can be proven independently and we do not need to inline this part: in the previous example, we do not need to add $h0$ in each step, since we know that $h0$ is true (we can derive it from the assumptions). Thus, the linear part (composed of steps that introduce a single new hypotheses), can be transformed from proofs of false to proofs of a new assumption: if we know that $h0 \to h1$ and that $h0$ is true, we have $h1$, we do not need to inline the implication. This also simplifies the proof term at each step.

It would be interesting to see if there is a way to better understand which implication are really needed by `metis`: this powerful tactic is complete and thus should be able to find directly the proof for some series of implications: if we have $h1 \to h2$, $h2 \to h3$, $h3 \to h4$, it is likely that it can directly find that $h1 \to h4$ (probably as fast). In the linear part, the compression of Sledgehammer will correct this, but in the terms, there are no simple ways to compress the implications. For example proving that $(\exists x.p\ x\ 0) \to (\forall y.p\ y\ 0 \to p\ y\ 1) \to (\forall y.p\ y\ 1 \to q\ y) \to (\exists x.qx)$ (the three hypotheses are named `a`, `b` and `c`), gives the following proof: on the right you can see the Satallax proof and on the left the corresponding veriT proof. The term `f1` is not needed in the proof, but Sledgehammer is not able to compress it.

```
proof -
  have f1: "not p x 1 \/ not not p x 1"
    by metis
  obtain x :: 'a where "p x 0" by (metis a)
  hence "p x 1" by (metis c)
  thus "EX y. q y" using f1 by (metis b)
qed
```

```
proof -
  obtain x :: 'a where "p x 0"
    using a by metis
  hence "p x 1" by (metis c)
  hence "q x" by (metis b)
  thus "EX. y. q y" by metis
qed
```

After this transformation, Sledgehammer is able to deal with the proof. We will now see what are the rules used by Satallax.

5.4 The rules

We were not able to find any list nor definitions of the rules that can be applied, but thankfully, as the proof can also be exported to a Coq script, we were able to watch the Coq tactic definition in the files `stt.v` and `sttab.v` (in the directory `coq` of the Satallax source file). The formula on the right is the formula used for the step. Goals of the form $P \to \bot$ are directly transformed into $\bot$ where P is added to the assumptions. We do not indicate the unchanged hypotheses. Most rules (see annex A on page 16 for the list) are not very interesting, especially as `metis` does not need the fact to reconstruct the proof (except some instantiation, but as Sledgehammer preplays the proof, it finds that `simp` is better suited for this case).

6 Conclusion

This report presents the integration of three new provers to Sledgehammer and their corresponding proof reconstruction. Adding the reconstruction for a higher-order prover should improve the usability of Sledgehammer for higher-order goals. First-order prover support is still interesting, because, even if the ones we added is not part of the best provers, each one has its strengths and weaknesses. On the ATP side, we have seen some bugs that can be solved and some problems (like the Ecall that does not produce a proof). We hope that an output standard will emerge in SMT world, allowing us easier set up for them in Isabelle/HOL.

Future work More testing by using Böhme and Nipkow's Judgement Day [42] would be interesting to see how much the reconstruction improves the efficiency compared to a `metis` call with the facts. Proof support have been announced for CVC4 and there is work on a new higher-order prover `HolyHammer`. It is important to notice that the format does slightly change when the version changes. More generally it would be interesting to see if the exchange between automatic theorem prover and the Isabelle lemma can help to develop the provers such that they are more efficient: integrating automated provers is interesting for people who want to prove something, but the former could also be improved to be more efficient for goals generated by humans and not only for TPTP problems.

References

[1] Michael Gordon, Robin Milner, and Christopher Wadsworth. *Edinburgh LCF: a mechanized logic of computation, volume 78 of Lecture Notes in Computer Science*. 1979.

[2] Georges Gonthier. "The Four Colour Theorem: Engineering of a Formal Proof". English. In: *Computer Mathematics*. Ed. by Deepak Kapur. Vol. 5081. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2008, pp. 333–333. ISBN: 978-3-540-87826-1. DOI: `10.1007/978-3-540-87827-8_28`.

[3] Karthikeyan Bhargavan, Antoine Delignat-Lavaud, Cédric Fournet, Alfredo Pironti, and Pierre-Yves Strub. "Triple Handshakes and Cookie Cutters: Breaking and Fixing Authentication over TLS". In: *IEEE Symposium on Security & Privacy (Oakland)*. 2014. URL: `pubs/triple-handshakes-and-cookie-cutters-sp14.pdf`.

[4] William Mccune. "Solution of the Robbins Problem". English. In: *Journal of Automated Reasoning* 19.3 (1997), pp. 263–276. ISSN: 0168-7433. DOI: `10.1023/A:1005843212881`.

[5] Renate A. Schmidt, Stephan Schulz, and Boris Konev, eds. *Proceedings of the 2nd Workshop on Practical Aspects of Automated Reasoning, PAAR-2010, Edinburgh, Scotland, UK, July 14, 2010*. Vol. 9. EPiC Series. EasyChair, 2012.

[6] Tobias Nipkow, Markus Wenzel, and Lawrence C. Paulson. *Isabelle/HOL: A Proof Assistant for Higher-order Logic*. Berlin, Heidelberg: Springer-Verlag, 2002. ISBN: 3-540-43376-7.

[7] Makarius Wenzel. "Isabelle/jEdit – A Prover IDE within the PIDE Framework". English. In: *Intelligent Computer Mathematics*. Ed. by Johan Jeuring, JohnA. Campbell, Jacques Carette, Gabriel Dos Reis, Petr Sojka, Makarius Wenzel, and Volker Sorge. Vol. 7362. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2012, pp. 468–471. ISBN: 978-3-642-31373-8. DOI: `10.1007/978-3-642-31374-5_38`.

[8] David C.J. Matthews and Makarius Wenzel. “Efficient Parallel Programming in Poly/ML and Isabelle/ML”. In: *Proceedings of the 5th ACM SIGPLAN Workshop on Declarative Aspects of Multicore Programming*. DAMP ’10. Madrid, Spain: ACM, 2010, pp. 53–62. ISBN: 978-1-60558-859-9. DOI: 10.1145/1708046.1708058. URL: http://doi.acm.org/10.1145/1708046.1708058.

[9] Mike Gordon. “Proof, Language, and Interaction”. In: ed. by Gordon Plotkin, Colin Stirling, and Mads Tofte. Cambridge, MA, USA: MIT Press, 2000. Chap. From LCF to HOL: A Short History, pp. 169–185. ISBN: 0-262-16188-5. URL: http://dl.acm.org/citation.cfm?id=345868.345890.

[10] Lawrence C. Paulson. “Set theory for verification: I. From foundations to functions”. English. In: *Journal of Automated Reasoning* 11.3 (1993), pp. 353–389. ISSN: 0168-7433. DOI: 10.1007/BF00881873.

[11] Olaf Müller, Tobias Nipkow, David von Oheimb, and Oscar Slotosch. “HOLCF = HOL + LCF”. In: *Journal of Functional Programming* 9 (02 Mar. 1999), pp. 191–223. ISSN: 1469-7653. DOI: 10.1017/S095679689900341X.

[12] Jeremy Avigad, Kevin Donnelly, David Gray, and Paul Raff. “A Formally Verified Proof of the Prime Number Theorem”. In: *ACM Trans. Comput. Logic* 9.1 (Dec. 2007). ISSN: 1529-3785. DOI: 10.1145/1297658.1297660. URL: http://doi.acm.org/10.1145/1297658.1297660.

[13] Gernot Heiser, Kevin Elphinstone, Ihor Kuz, Gerwin Klein, and Stefan M. Petters. “Towards Trustworthy Computing Systems: Taking Microkernels to the Next Level”. In: *SIGOPS Oper. Syst. Rev.* 41.4 (July 2007), pp. 3–11. ISSN: 0163-5980. DOI: 10.1145/1278901.1278904. URL: http://doi.acm.org/10.1145/1278901.1278904.

[14] Gerwin Klein and Tobias Nipkow. “A Machine-checked Model for a Java-like Language, Virtual Machine, and Compiler”. In: *ACM Trans. Program. Lang. Syst.* 28.4 (July 2006), pp. 619–695. ISSN: 0164-0925. DOI: 10.1145/1146809.1146811.

[15] Makarius Wenzel. “Isabelle/Isar – a generic framework for human-readable proof documents”. In: *UNIVERSITY OF BIALYSTOK*. 2007.

[16] Geoff Sutcliffe. “The TPTP World – Infrastructure for Automated Reasoning”. English. In: *Logic for Programming, Artificial Intelligence, and Reasoning*. Ed. by EdmundM. Clarke and Andrei Voronkov. Vol. 6355. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2010, pp. 1–12. ISBN: 978-3-642-17510-7. DOI: 10.1007/978-3-642-17511-4_1.

[17] Geoff Sutcliffe. “The TPTP Problem Library and Associated Infrastructure”. English. In: *Journal of Automated Reasoning* 43.4 (2009), pp. 337–362. ISSN: 0168-7433. DOI: 10.1007/s10817-009-9143-8.

[18] Stephan Schulz. “System Description: E 1.8”. English. In: *Logic for Programming, Artificial Intelligence, and Reasoning*. Ed. by Ken McMillan, Aart Middeldorp, and Andrei Voronkov. Vol. 8312. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2013, pp. 735–743. ISBN: 978-3-642-45220-8. DOI: 10.1007/978-3-642-45221-5_49.

[19] Christoph Weidenbach, Dilyana Dimova, Arnaud Fietzke, Rohit Kumar, Martin Suda, and Patrick Wischnewski. “SPASS Version 3.5”. English. In: *Automated Deduction – CADE-22*. Ed. by RenateA. Schmidt. Vol. 5663. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2009, pp. 140–145. ISBN: 978-3-642-02958-5. DOI: 10.1007/978-3-642-02959-2_10.

[20] Laura Kovács and Andrei Voronkov. “First-Order Theorem Proving and Vampire”. English. In: *Computer Aided Verification*. Ed. by Natasha Sharygina and Helmut Veith. Vol. 8044. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2013, pp. 1–35. ISBN: 978-3-642-39798-1. DOI: 10.1007/978-3-642-39799-8_1.

[21] Stephan Schulz. “E - a Brainiac Theorem Prover”. In: *AI Commun.* 15.2,3 (Aug. 2002), pp. 111–126. ISSN: 0921-7126. URL: http://dl.acm.org/citation.cfm?id=1218615.1218621.

[22] Clark Barrett, Christopher L. Conway, Morgan Deters, Liana Hadarean, Dejan Jovanovic, Tim King, Andrew Reynolds, and Cesare Tinelli. “CVC4”. English. In: *Computer Aided Verification*. Ed. by Ganesh Gopalakrishnan and Shaz Qadeer. Vol. 6806. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2011, pp. 171–177. ISBN: 978-3-642-22109-5. DOI: 10.1007/978-3-642-22110-1_14.

[23] Bruno Dutertre. “Yices 2.2”. English. In: *Computer Aided Verification*. Ed. by Armin Biere and Roderick Bloem. Vol. 8559. Lecture Notes in Computer Science. Springer International Publishing, 2014, pp. 737–744. ISBN: 978-3-319-08866-2. DOI: 10.1007/978-3-319-08867-949.

[24] Leonardo de Moura and Nikolaj Bjørner. “Z3: An Efficient SMT Solver”. English. In: *Tools and Algorithms for the Construction and Analysis of Systems*. Ed. by C.R. Ramakrishnan and Jakob Rehof. Vol. 4963. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2008, pp. 337–340. ISBN: 978-3-540-78799-0. DOI: 10.1007/978-3-540-78800-3_24.

[25] Clark Barrett, Aaron Stump, and Cesare Tinelli. “The smt-lib standard: Version 2.0”. In: *Proceedings of the 8th International Workshop on Satisfiability Modulo Theories (Edinburgh, England)*. Vol. 13. 2010, p. 14.

[26] Frédéric Besson, Pascal Fontaine, and Laurent Théry. "A Flexible Proof Format for SMT: a Proposal". English. In: *First International Workshop on Proof eXchange for Theorem Proving - PxTP 2011*. Ed. by Pascal Fontaine and Aaron Stump. Wroclaw, Pologne, Aug. 2011. URL: http://hal.inria.fr/hal-00642544.

[27] Aaron Stump and Duckki Oe. "Towards an SMT Proof Format". In: *Proceedings of the Joint Workshops of the 6th International Workshop on Satisfiability Modulo Theories and 1st International Workshop on Bit-Precise Reasoning*. SMT '08/BPR '08. Princeton, New Jersey: ACM, 2008, pp. 27–32. ISBN: 978-1-60558-440-9. DOI: 10.1145/1512464.1512470. URL: http://doi.acm.org/10.1145/1512464.1512470.

[28] Daniel Kühlwein, Jasmin Christian Blanchette, Cezary Kaliszyk, and Josef Urban. "MaSh: Machine Learning for Sledgehammer". English. In: *Interactive Theorem Proving*. Ed. by Sandrine Blazy, Christine Paulin-Mohring, and David Pichardie. Vol. 7998. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2013, pp. 35–50. ISBN: 978-3-642-39633-5. DOI: 10.1007/978-3-642-39634-2_6.

[29] Nik Sultana, Jasmin Christian Blanchette, and Lawrence C. Paulson. "LEO-II and Satallax on the Sledgehammer test bench". In: *Journal of Applied Logic* 11.1 (2013), pp. 91–102. ISSN: 1570-8683. DOI: http://dx.doi.org/10.1016/j.jal.2012.12.002. URL: http://www.sciencedirect.com/science/article/pii/S1570868312000766.

[30] Joe Hurd. "First-Order Proof Tactics in Higher-Order Logic Theorem Provers". In: pp. 56–68. URL: http://www.gilith.com/research/papers.

[31] Jasmin Christian Blanchette. "Redirecting Proofs by Contradiction". In: *PxTP 2013*. Ed. by Jasmin Christian Blanchette and Josef Urban. Vol. 14. EPiC Series. EasyChair, 2013, pp. 11–26. DOI: 10.1.1.299.2517. URL: http://www.easychair.org/publications/?page=1248043783.

[32] Sascha Böhme. "Proof Reconstruction for Z3 in Isabelle/HOL". In: *7th International Workshop on Satisfiability Modulo Theories (SMT '09)*. 2009.

[33] Cezary Kaliszyk and Josef Urban. "PRocH: Proof Reconstruction for HOL Light". English. In: *Automated Deduction – CADE-24*. Ed. by MariaPaola Bonacina. Vol. 7898. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2013, pp. 267–274. ISBN: 978-3-642-38573-5. DOI: 10.1007/978-3-642-38574-2_18.

[34] Josef Urban, Piotr Rudnicki, and Geoff Sutcliffe. "ATP and Presentation Service for Mizar Formalizations". English. In: *Journal of Automated Reasoning* 50.2 (2013), pp. 229–241. ISSN: 0168-7433. DOI: 10.1007/s10817-012-9269-y.

[35] Simon Foster and Georg Struth. "Integrating an Automated Theorem Prover into Agda". English. In: *NASA Formal Methods*. Ed. by Mihaela Bobaru, Klaus Havelund, GerardJ. Holzmann, and Rajeev Joshi. Vol. 6617. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2011, pp. 116–130. ISBN: 978-3-642-20397-8. DOI: 10.1007/978-3-642-20398-5.

[36] Geoff Sutcliffe. "The 6th IJCAR Automated Theorem Proving System Competition –CASC-J6". In: *AI Commun.* 26.2 (Apr. 2013), pp. 211–223. ISSN: 0921-7126. URL: http://dl.acm.org/citation.cfm?id=2594582.2594586.

[37] Nik Sultana and Christoph Benzmüller. "Understanding LEO-II's Proofs". In: *The 9th International Workshop on the Implementation of Logics (IWIL-2012, affiliated with LPAR-2012)*. Ed. by Eugenia Ternovska, Konstantin Korovin, and Stephan Schulz. Merida, Venezuela, 2012.

[38] Thomas Bouton, Diego Caminha B. de Oliveira, David Déharbe, and Pascal Fontaine. "veriT: An Open, Trustable and Efficient SMT-Solver". English. In: *Automated Deduction – CADE-22*. Ed. by RenateA. Schmidt. Vol. 5663. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2009, pp. 151–156. ISBN: 978-3-642-02958-5. DOI: 10.1007/978-3-642-02959-2_12.

[39] Joseph Boudou and Bruno Woltzenlogel Paleo. "Compression of Propositional Resolution Proofs by Lowering Subproofs". English. In: *Automated Reasoning with Analytic Tableaux and Related Methods*. Ed. by Didier Galmiche and Dominique Larchey-Wendling. Vol. 8123. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2013, pp. 59–73. ISBN: 978-3-642-40536-5. DOI: 10.1007/978-3-642-40537-2_7.

[40] ChadE. Brown. "Satallax: An Automatic Higher-Order Prover". English. In: *Automated Reasoning*. Ed. by Bernhard Gramlich, Dale Miller, and Uli Sattler. Vol. 7364. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2012, pp. 111–117. ISBN: 978-3-642-31364-6. DOI: 10.1007/978-3-642-31365-3_11.

[41] Olivier Gasquet, Andreas Herzig, Bilal Said, and François Schwarzentruber. *Kripke's Worlds - An Introduction to Modal Logics via Tableaux*. Studies in Universal Logic. Birkhäuser, 2014, pp. I–XV, 1–198. ISBN: 978-3-7643-8503-3.

[42] Sascha Böhme and Tobias Nipkow. "Sledgehammer: Judgement Day". English. In: *Automated Reasoning*. Ed. by Jürgen Giesl and Reiner Hähnle. Vol. 6173. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2010, pp. 107–121. ISBN: 978-3-642-14202-4. DOI: 10.1007/978-3-642-14203-1_9.

A Satallax rules

To shorten the notation we will use: $\dfrac{\neg P \vdash \bot \qquad Q \vdash \bot}{(P \to Q) \vdash \bot}$ $(P \to Q) \to (\neg P \to \bot) \to (Q \to \bot) \to \bot$ to mean that to prove $\dfrac{P \to Q}{\bot}$, we must show that $\dfrac{\neg P}{\bot}$ and $\dfrac{Q}{\bot}$.

First there are some proofs theorems, that Satallax knows and introduce as a dependency. It is mostly first-order rewriting.

A.1 Satallax' theorems

eq_ind $P\ x \to (\forall y, x = y \to P\ y)$
eq_trans2 $x = y \to z = y \to x = z$
eq_trans3 $y = x \to y = z \to x = z$
eq_neg_neg_id $\neg\neg x = x$
eq_true $\top = \neg\bot$
eq_and_nor $x \wedge y = \neg(\neg x \vee \neg y)$
eq_or_nand $x \vee y = \neg(\neg x \wedge \neg y)$
eq_or_imp $x \vee y = \neg x \to y$
eq_and_imp $x \wedge y = \neg(x \to \neg y)$
eq_imp_or $x \to y = (\neg x y)$
eq_iff $x \iff t = (x = t)$
eq_sym $(s = t) = (t = s)$
eq_forall_nexists $(\forall x.f\ x) = (\neg\exists x.\neg f\ x)$
eq_exists_nforall $(\exists x.f\ x) = (\neg\forall x.\neg f\ x)$
eq_leib1 $(\forall p.p\ s \implies p\ t) = (s = t)$
eq_leib2 $(\forall p.\neg p\ s \implies \neg p\ t) = (s = t)$
eq_leib3 $(\forall p.p\ x\ x \implies p\ s\ t) = (s = t)$
eq_leib4 $(\forall p.\neg p\ x\ x \implies \neg p\ s\ t) = (s = t)$
eq_eta $(\lambda f\ x.\ f\ x) = \lambda f.\ f$
SinhE $(\forall x, P) \to P$ (yes, P does not depend on x)
eq_forall_Seps $(\forall x.p\ x) = p\,(\text{Sepsilon}\ \ (\lambda x.\neg p\ x))$
eq_SPi_Seps $(\lambda p.p) = (\lambda p.p\,(\text{Sepsilon}\ \ (\lambda x.\neg p\ x)))$
eq_exists_Seps $(\exists x.\ p\ x) = p\ \ (\text{Sepsilon}\ p)$

Those theorems are used by `tab_known`. There are a lot of other rules, all starting with `tab`, since Satallax is using the tableau method.

A.2 Rules

tab_conflict $A = B, A\neg B \vdash \bot$ or $A = B, B\neg A \vdash \bot$
tab_false $\bot \vdash \bot$
tab_refl $A \neq A \vdash \bot$
tab_imp $\dfrac{\neg X \to \bot \vdash \bot \qquad Y \to \bot \vdash \bot}{X \to Y \vdash \bot}$ $(X \to Y) \to (\neg X \to \bot) \to (Y \to \bot) \to \bot$
tab_or $\dfrac{X \to \bot \vdash \bot \qquad Y \to \bot \vdash \bot}{X \vee Y \vdash \bot}$ $(X \vee Y) \to (X \to \bot) \to (Y \to \bot) \to \bot$
tab_nand $\dfrac{\neg X \to \bot \vdash \bot \qquad \neg Y \to \bot \vdash \bot}{\neg X \wedge Y \vdash \bot}$ $(\neg X \wedge Y) \to (\neg X \to \bot) \to (\neg Y \to \bot) \to \bot$
tab_negimp $\dfrac{X \to \neg Y \to \bot \vdash \bot}{\neg X \to Y \vdash \bot}$ $(\neg X \to Y) \to (X \to \neg Y \to \bot) \to \bot$
tab_and $\dfrac{X \to Y \to \bot \vdash \bot}{X \wedge Y \vdash \bot}$ $(X \wedge Y) \to (X \to Y \to \bot) \to \bot$
tab_nor $\dfrac{\neg X \to \neg Y \to \bot \vdash \bot}{\neg X \vee Y \vdash \bot}$ $(\neg X \vee Y) \to (\neg X \to \neg Y \to \bot) \to \bot$

tab_all $\dfrac{\forall y : X, p\ y \to \bot \vdash \bot}{\forall x : X, p\ x \vdash \bot}$ $(\forall x : X, p\ x) \to (\forall y : X, p\ y \to \bot) \to \bot$

tab_negall $\dfrac{\forall y : X, \neg p\ y \to \bot \vdash \bot}{\neg\forall x : X, p\ x \vdash \bot}$ $(\neg\forall x : X, p\ x) \to (\forall y : X, \neg p\ y \to \bot) \to \bot$

tab_ex $\dfrac{\forall y : X, p\ y \to \bot \vdash \bot}{\exists x : X, p\ x \vdash \bot}$ $(\exists x : X, p\ x) \to (\forall y : X, p\ y \to \bot) \to \bot$

tab_negex $\dfrac{\forall y : X, \neg p\ y \to \bot \vdash \bot}{\neg\exists x : X, p\ x \vdash \bot}$ $(\neg\exists x : X, p\ x) \to (\forall y : X, \neg p\ y \to \bot) \to \bot$

tab_mat $\dfrac{s \neq t \to \bot \vdash \bot}{p\ s \to \neg q\ t \to p \neq q \to \bot \vdash \bot}$ $(p\ s \to \neg q\ t \to p \neq q \to \bot) \to (s \neq t \to \bot) \to \bot$

tab_dec $\dfrac{p \neq q \to \bot \vdash \bot \qquad s \neq t \to \bot \vdash \bot}{p\ s \neq q\ t \vdash \bot}$ $(p\ s \neq q\ t) \to (p \neq q \to \bot) \to (s \neq t \to \bot) \to \bot$ otherwise we would have $p\ s \neq\ p\ s \vdash\ \bot$

tab_con $\dfrac{Q \vdash \bot \qquad R \vdash \bot}{P \vdash \bot}$ $(P) \to (Q) \to (R) \to \bot$ where $P = s = t \to u \neq v$, $Q = s \neq u \to t \neq u \to \bot$ and $R = s \neq v \to t \neq v \to \bot$

tab_trans $\dfrac{s = u \to \bot \vdash \bot}{s = t \to t = u \vdash \bot}$ $(s = t \to t = u) \to (s = u \to \bot) \to \bot$

tab_be $\dfrac{X \to \neg Y \to \bot \vdash \bot \qquad \neg X \to Y \to \bot \vdash \bot}{X \neq Y \vdash \bot}$ $(X \neq Y) \to (X \to \neg Y \to \bot) \to (\neg X \to Y \to \bot) \to \bot$

tab_bq $\dfrac{X \to Y \to \bot \vdash \bot \qquad \neg X \to \neg Y \to \bot \vdash \bot}{X = Y \vdash \bot}$ $(X = Y) \to (X \to Y \to \bot) \to (\neg X \to \neg Y \to \bot) \to \bot$

tab_negiff $\dfrac{X \to Y \vdash \bot \qquad \neg X \to Y \to \bot \vdash \bot}{\neg X \leftrightarrow Y \vdash \bot}$ $(\neg X \leftrightarrow Y) \to (X \to Y) \to (\neg X \to Y \to \bot) \to \bot$

tab_iff $\dfrac{X \to Y \to \bot \vdash \bot \qquad \neg X \to \neg Y \to \bot \vdash \bot}{X \leftrightarrow Y \vdash \bot}$ $(X \leftrightarrow Y) \to (X \to Y \to \bot) \to (\neg X \to \neg Y \to \bot) \to \bot$

tab_fe $\dfrac{\neg\forall x, s\ x = t\ x \to \bot \vdash \bot}{s \neq t \vdash \bot}$ $(s \neq t) \to (\neg\forall x, s\ x = t\ x \to \bot) \to \bot$

tab_fq $\dfrac{\forall x, s\ x = t\ x \to \bot \vdash \bot}{s = t \vdash \bot}$ $(s = t) \to (\forall x, s\ x = t\ x \to \bot) \to \bot$

tab_choice $\dfrac{\forall x, \neg p\ x \vdash \bot \qquad p\ (S\varepsilon\ p) \vdash \bot}{\top \vdash \bot}$ $(\top) \to (\forall x, \neg p\ x) \to (p\ (S\varepsilon\ p)) \to \bot$

tab_cut $\dfrac{s \vdash \bot \qquad \neg s \vdash \bot}{\top \vdash \bot}$ $(\top) \to (s) \to (\neg s) \to \bot$

tab_dn $\dfrac{\neg A \vdash \bot}{A \to \bot \vdash \bot}$

tab_delta delta reduction

tab_known is used to introduce some theorem known by Satallax like eq_ind see previous subsection

B Satallax proof example

```
thf(conj_0,conjecture,(p & (~(r)))).
thf(h0,negated_conjecture,(~((p & (~(r))))),inference(assume_negation,
   [status(cth)],[conj_0])).
thf(h1,assumption,(~(q)),introduced(assumption,[])).
thf(h2,assumption,r,introduced(assumption,[])).
thf(h3,assumption,p,introduced(assumption,[])).
thf(h4,assumption,q,introduced(assumption,[])).
thf(h5,assumption,(~(p)),introduced(assumption,[])).
thf(h6,assumption,(~((~(r)))),introduced(assumption,[])).
thf(a1,axiom,(~(r))).
thf(a3,axiom,(p | q)).
thf(a2,axiom,((~(q)) | r)).
thf(4,plain,$false,inference(tab_conflict,[status(thm),
  assumptions([h5,h3,h1,h0])],[h3,h5])).
thf(5,plain,$false,inference(tab_conflict,[status(thm),assumptions([h6,h3,
```

```
   h1, h0])],[a1,h6])).
thf(3,plain,$false,inference(tab_nand,[status(thm),assumptions([h3,h1,h0]),
tab_nand(discharge,[h5]),tab_nand(discharge,[h6])],[h0,4,5,h5,h6])).
thf(6,plain,$false,inference(tab_conflict,[status(thm),assumptions([h4,h1,
   h0])],[h4,h1])).
thf(2,plain,$false,inference(tab_or,[status(thm),assumptions([h1,h0]),
tab_or(discharge,[h3]),tab_or(discharge,[h4])],[a3,3,6,h3,h4])).
thf(7,plain,$false,inference(tab_conflict,[status(thm),assumptions([h2,
   h0])],[h2,a1])).
thf(1,plain,$false,inference(tab_or,[status(thm),assumptions([h0]),
tab_or(discharge,[h1]),tab_or(discharge,[h2])],[a2,2,7,h1,h2])).
thf(0,theorem,(p & (~(r))),inference(contra,[status(thm),contra(
   discharge,[h0])],[1,h0])).
```