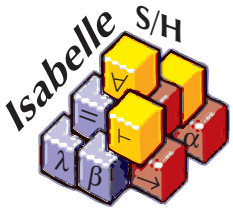

Hammering Away

A User's Guide to Sledgehammer for Isabelle/HOL

Jasmin Christian Blanchette
Institut für Informatik, Technische Universität München

with contributions from

Lawrence C. Paulson
Computer Laboratory, University of Cambridge

May 25, 2015

Contents

1 Introduction

Sledgehammer is a tool that applies automatic theorem provers (ATPs) and satisfiability-modulo-theories (SMT) solvers on the current goal.[1] The supported ATPs are AgsyHOL [13], Alt-Ergo [4], E [15], E-SInE [10], E-ToFoF [17], iProver [11], iProver-Eq [12], LEO-II [3], Satallax [7], SNARK [16], SPASS [19], Vampire [14], Waldmeister [9], and Zipperposition [8]. The ATPs are run either locally or remotely via the SystemOnTPTP web service

[1]The distinction between ATPs and SMT solvers is convenient but mostly historical. The two communities are converging, with more and more ATPs supporting typical SMT features such as arithmetic and sorts, and a few SMT solvers parsing ATP syntaxes. There is also a strong technological connection between instantiation-based ATPs (such as iProver and iProver-Eq) and SMT solvers.

[18]. The supported SMT solvers are CVC3 [2], CVC4 [1], veriT [6], and Z3 [20]. These are always run locally.

The problem passed to the external provers (or solvers) consists of your current goal together with a heuristic selection of hundreds of facts (theorems) from the current theory context, filtered by relevance.

The result of a successful proof search is some source text that usually (but not always) reconstructs the proof within Isabelle. For ATPs, the reconstructed proof typically relies on the general-purpose *metis* proof method, which integrates the Metis ATP in Isabelle/HOL with explicit inferences going through the kernel. Thus its results are correct by construction.

For Isabelle/jEdit users, Sledgehammer provides an automatic mode that can be enabled via the "Auto Sledgehammer" option under "Plugins > Plugin Options > Isabelle > General." In this mode, a reduced version of Sledgehammer is run on every newly entered theorem for a few seconds.

To run Sledgehammer, you must make sure that the theory *Sledgehammer* is imported—this is rarely a problem in practice since it is part of *Main*. Examples of Sledgehammer use can be found in Isabelle's `src/HOL/Metis_Examples` directory. Comments and bug reports concerning Sledgehammer or this manual should be directed to the author at `blanchette@in.tum.de`.

2 Installation

Sledgehammer is part of Isabelle, so you do not need to install it. However, it relies on third-party automatic provers (ATPs and SMT solvers).

Among the ATPs, AgsyHOL, Alt-Ergo, E, LEO-II, Satallax, SPASS, Vampire, and Zipperposition can be run locally; in addition, AgsyHOL, E, E-SInE, E-ToFoF, iProver, iProver-Eq, LEO-II, Satallax, SNARK, Vampire, and Waldmeister are available remotely via SystemOnTPTP [18]. The SMT solvers CVC3, CVC4, veriT, and Z3 can be run locally.

There are three main ways to install automatic provers on your machine:

- If you installed an official Isabelle package, it should already include properly setup executables for CVC4, E, SPASS, and Z3, ready to use.[2]

[2]Vampire's license prevents us from doing the same for this otherwise remarkable tool.

- Alternatively, you can download the Isabelle-aware CVC3, CVC4, E, SPASS, and Z3 binary packages from `http://isabelle.in.tum.de/components/`. Extract the archives, then add a line to your `$ISABELLE_HOME_USER/etc/components`[3] file with the absolute path to CVC3, CVC4, E, SPASS, or Z3. For example, if the `components` file does not exist yet and you extracted SPASS to `/usr/local/spass-3.8ds`, create it with the single line

  ```
  /usr/local/spass-3.8ds
  ```

  in it.

- If you prefer to build AgsyHOL, Alt-Ergo, E, LEO-II, Satallax, or SPASS manually, or found a Vampire executable somewhere (e.g., `http://www.vprover.org/`), set the environment variable `AGSYHOL_HOME`, `E_HOME`, `LEO2_HOME`, `SATALLAX_HOME`, `SPASS_HOME`, or `VAMPIRE_HOME` to the directory that contains the `agsyHOL`, `eprover` (and/or `eproof` or `eproof_ram`), `leo`, `satallax`, `SPASS`, or `vampire` executable; for Alt-Ergo, set the environment variable `WHY3_HOME` to the directory that contains the `why3` executable. Sledgehammer has been tested with AgsyHOL 1.0, Alt-Ergo 0.95.2, E 1.6 to 1.8, LEO-II 1.3.4, Satallax 2.2 to 2.7, SPASS 3.8ds, and Vampire 0.6 to 3.0.[4]Since the ATPs' output formats are neither documented nor stable, other versions might not work well with Sledgehammer. Ideally, you should also set `E_VERSION`, `LEO2_VERSION`, `SATALLAX_VERSION`, `SPASS_VERSION`, or `VAMPIRE_VERSION` to the prover's version number (e.g., "3.0").

  Similarly, if you want to install CVC3, CVC4, veriT, or Z3, set the environment variable `CVC3_SOLVER`, `CVC4_SOLVER`, `VERIT_SOLVER`, or `Z3_SOLVER` to the complete path of the executable, *including the file name*. Sledgehammer has been tested with CVC3 2.2 and 2.4.1, CVC4 1.5-prerelease, veriT smtcomp2014, and Z3 4.3.2. Since Z3's output format is somewhat unstable, other versions of the solver might not work well with Sledgehammer. Ideally, also set `CVC3_VERSION`, `CVC4_VERSION`, `VERIT_VERSION`, or `Z3_VERSION` to the solver's version number (e.g., "4.4.0").

To check whether E, SPASS, Vampire, and/or Z3 are successfully installed, try out the example in § 3. If the remote versions of any of these provers is

[3]The variable `$ISABELLE_HOME_USER` is set by Isabelle at startup. Its value can be retrieved by executing `isabelle getenv ISABELLE_HOME_USER` on the command line.

[4]Following the rewrite of Vampire, the counter for version numbers was reset to 0; hence the (new) Vampire versions 0.6, 1.0, 1.8, 2.6, and 3.0 are more recent than 9.0 or 11.5.

used (identified by the prefix "*remote_*"), or if the local versions fail to solve the easy goal presented there, something must be wrong with the installation.

Remote prover invocation requires Perl with the World Wide Web Library (`libwww-perl`) installed. If you must use a proxy server to access the Internet, set the `http_proxy` environment variable to the proxy, either in the environment in which Isabelle is launched or in your `$ISABELLE_HOME_USER/etc/settings` file. Here are a few examples:

```
http_proxy=http://proxy.example.org
http_proxy=http://proxy.example.org:8080
http_proxy=http://joeblow:pAsSwRd@proxy.example.org
```

3 First Steps

To illustrate Sledgehammer in context, let us start a theory file and attempt to prove a simple lemma:

theory *Scratch*
imports *Main*
begin

lemma "$[a] = [b] \Longrightarrow a = b$"
sledgehammer

Instead of issuing the **sledgehammer** command, you can also use the Sledgehammer panel in Isabelle/jEdit. Sledgehammer produces the following output after a few seconds:

Sledgehammer: "cvc4"
Try this: **by** *(metis last_ConsL) (64 ms).*

Sledgehammer: "z3"
Try this: **by** *(metis list.inject) (20 ms).*

Sledgehammer: "remote_vampire"
Try this: **by** *(metis hd.simps) (14 ms).*

Sledgehammer: "spass"
Try this: **by** *(metis list.inject) (17 ms).*

Sledgehammer ran CVC4, SPASS, Vampire, and Z3 in parallel. Depending on which provers are installed and how many processor cores are available, some of the provers might be missing or present with a *remote_* prefix.

For each successful prover, Sledgehammer gives a one-line *metis* or *smt2* method call. Rough timings are shown in parentheses, indicating how fast the call is. You can click the proof to insert it into the theory text.

In addition, you can ask Sledgehammer for an Isar text proof by enabling the *isar_proofs* option (§ 7.4):

sledgehammer [*isar_proofs*]

When Isar proof construction is successful, it can yield proofs that are more readable and also faster than the *metis* or *smt2* one-line proofs. This feature is experimental and is only available for ATPs.

4 Hints

This section presents a few hints that should help you get the most out of Sledgehammer. Frequently asked questions are answered in § 5.

4.1 *Presimplify the goal*

For best results, first simplify your problem by calling *auto* or at least *safe* followed by *simp_all*. The SMT solvers provide arithmetic decision procedures, but the ATPs typically do not (or if they do, Sledgehammer does not use it yet). Apart from Waldmeister, they are not particularly good at heavy rewriting, but because they regard equations as undirected, they often prove theorems that require the reverse orientation of a *simp* rule. Higher-order problems can be tackled, but the success rate is better for first-order problems. Hence, you may get better results if you first simplify the problem to remove higher-order features.

4.2 *Make sure E, SPASS, Vampire, and Z3 are locally installed*

Locally installed provers are faster and more reliable than those running on servers. See § 2 for details on how to install them.

4.3 *Familiarize yourself with the main options*

Sledgehammer's options are fully documented in § 6. Many of the options are very specialized, but serious users of the tool should at least familiarize themselves with the following options:

- ***provers*** (§ 7.1) specifies the automatic provers (ATPs and SMT solvers) that should be run whenever Sledgehammer is invoked (e.g., "*provers = e spass remote_vampire*"). For convenience, you can omit "*provers =*" and simply write the prover names as a space-separated list (e.g., "*e spass remote_vampire*").
- ***max_facts*** (§ 7.2) specifies the maximum number of facts that should be passed to the provers. By default, the value is prover-dependent but varies between about 50 and 1000. If the provers time out, you can try lowering this value to, say, 25 or 50 and see if that helps.
- ***isar_proofs*** (§ 7.4) specifies that Isar proofs should be generated, in addition to one-line *metis* or *smt2* proofs. The length of the Isar proofs can be controlled by setting *compress* (§ 7.4).
- ***timeout*** (§ 7.6) controls the provers' time limit. It is set to 30 seconds, but since Sledgehammer runs asynchronously you should not hesitate to raise this limit to 60 or 120 seconds if you are the kind of user who can think clearly while ATPs are active.

Options can be set globally using **sledgehammer_params** (§ 6). The command also prints the list of all available options with their current value. Fact selection can be influenced by specifying "(*add*: *my_facts*)" after the **sledgehammer** call to ensure that certain facts are included, or simply "(*my_facts*)" to force Sledgehammer to run only with *my_facts* (and any facts chained into the goal).

5 Frequently Asked Questions

This sections answers frequently (and infrequently) asked questions about Sledgehammer. It is a good idea to skim over it now even if you do not have any questions at this stage. And if you have any further questions not listed here, send them to the author at `blanchette@in.tum.de`.

5.1 *Which facts are passed to the automatic provers?*

Sledgehammer heuristically selects a few hundred relevant lemmas from the currently loaded libraries. The component that performs this selection is called *relevance filter*.

- The traditional relevance filter, called *MePo* (Meng–Paulson), assigns a score to every available fact (lemma, theorem, definition, or axiom) based upon how many constants that fact shares with the conjecture. This process iterates to include facts relevant to those just accepted. The constants are weighted to give unusual ones greater significance. MePo 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 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.
- An alternative to MePo is *MaSh* (Machine Learner for Sledgehammer). It applies machine learning to the problem of finding relevant facts.
- The *MeSh* filter combines MePo and MaSh.

The default is either MePo or MeSh, depending on whether the environment variable `MASH` is set and what class of provers the target prover belongs to (§ 7.2).

The number of facts included in a problem varies from prover to prover, since some provers get overwhelmed more easily than others. You can show the number of facts given using the *verbose* option (§ 7.4) and the actual facts using *debug* (§ 7.4).

Sledgehammer is good at finding short proofs combining a handful of existing lemmas. If you are looking for longer proofs, you must typically restrict the number of facts, by setting the *max_facts* option (§ 7.2) to, say, 25 or 50.

You can also influence which facts are actually selected in a number of ways. If you simply want to ensure that a fact is included, you can specify it using the "(*add*: *my_facts*)" syntax. For example:

sledgehammer (*add*: *hd.simps tl.simps*)

The specified facts then replace the least relevant facts that would otherwise be included; the other selected facts remain the same. If you want to direct the selection in a particular direction, you can specify the facts via **using**:

using *hd.simps tl.simps*
sledgehammer

The facts are then more likely to be selected than otherwise, and if they are selected at iteration j they also influence which facts are selected at iterations $j + 1$, $j + 2$, etc. To give them even more weight, try

using *hd.simps tl.simps*
apply −
sledgehammer

5.2 *Why does Metis fail to reconstruct the proof?*

There are many reasons. If Metis runs seemingly forever, that is a sign that the proof is too difficult for it. Metis's search is complete for first-order logic with equality, so if the proof was found by an ATP such as E, SPASS, or Vampire, Metis should eventually find it, but that's little consolation.

In some rare cases, *metis* fails fairly quickly, and you get the error message "One-line proof reconstruction failed." This indicates that Sledgehammer determined that the goal is provable, but the proof is, for technical reasons, beyond *metis*'s power. You can then try again with the *strict* option (§ 7.3).

If the goal is actually unprovable and you did not specify an unsound encoding using *type_enc* (§ 7.3), this is a bug, and you are strongly encouraged to report this to the author at `blanchette@in.tum.de`.

5.3 *How can I tell whether a suggested proof is sound?*

Earlier versions of Sledgehammer often suggested unsound proofs—either proofs of nontheorems or simply proofs that rely on type-unsound inferences. This is a thing of the past, unless you explicitly specify an unsound encoding using *type_enc* (§ 7.3). Officially, the only form of "unsoundness" that lurks in the sound encodings is related to missing characteristic theorems of datatypes. For example,

lemma "$\exists xs.\ xs \neq []$"
sledgehammer ()

suggests an argumentless *metis* call that fails. However, the conjecture does actually hold, and the *metis* call can be repaired by adding *list.distinct*. We

hope to address this problem in a future version of Isabelle. In the meantime, you can avoid it by passing the *strict* option (§ 7.3).

5.4 *What are the full_types, no_types, and mono_tags arguments to Metis?*

The *metis* (*full_types*) proof method and its cousin *metis* (*mono_tags*) are fully-typed versions of Metis. It is somewhat slower than *metis*, but the proof search is fully typed, and it also includes more powerful rules such as the axiom "$x = \mathsf{True} \vee x = \mathsf{False}$" for reasoning in higher-order places (e.g., in set comprehensions). The method is automatically tried as a fallback when *metis* fails, and it is sometimes generated by Sledgehammer instead of *metis* if the proof obviously requires type information or if *metis* failed when Sledgehammer preplayed the proof. (By default, Sledgehammer tries to run *metis* with various sets of option for up to 1 second each time to ensure that the generated one-line proofs actually work and to display timing information. This can be configured using the *preplay_timeout* and *dont_preplay* options (§ 7.6).) At the other end of the soundness spectrum, *metis* (*no_types*) uses no type information at all during the proof search, which is more efficient but often fails. Calls to *metis* (*no_types*) are occasionally generated by Sledgehammer. See the *type_enc* option (§ 7.3) for details.

Incidentally, if you ever see warnings such as

> *Metis: Falling back on "metis (full_types)".*

for a successful *metis* proof, you can advantageously pass the *full_types* option to *metis* directly.

5.5 *And what are the lifting and hide_lams arguments to Metis?*

Orthogonally to the encoding of types, it is important to choose an appropriate translation of λ-abstractions. Metis supports three translation schemes, in decreasing order of power: Curry combinators (the default), λ-lifting, and a "hiding" scheme that disables all reasoning under λ-abstractions. The more powerful schemes also give the automatic provers more rope to hang themselves. See the *lam_trans* option (§ 7.3) for details.

5.6 *Are generated proofs minimal?*

Automatic provers frequently use many more facts than are necessary. Sledgehammer includes a minimization tool that takes a set of facts returned by a given prover and repeatedly calls a prover or proof method with subsets of those facts to find a minimal set. Reducing the number of facts typically helps reconstruction, while also removing superfluous clutter from the proof scripts.

In earlier versions of Sledgehammer, generated proofs were systematically accompanied by a suggestion to invoke the minimization tool. This step is now performed by default but can be disabled using the *minimize* option (§ 7.1).

5.7 *A strange error occurred—what should I do?*

Sledgehammer tries to give informative error messages. Please report any strange error to the author at `blanchette@in.tum.de`. This applies doubly if you get the message

> *The prover derived "False" using "foo", "bar", and "baz". This could be due to inconsistent axioms (including "***sorry***"s) or to a bug in Sledgehammer. If the problem persists, please contact the Isabelle developers.*

5.8 *Auto can solve it—why not Sledgehammer?*

Problems can be easy for *auto* and difficult for automatic provers, but the reverse is also true, so do not be discouraged if your first attempts fail. Because the system refers to all theorems known to Isabelle, it is particularly suitable when your goal has a short proof but requires lemmas that you do not know about.

5.9 *Why are there so many options?*

Sledgehammer's philosophy should work out of the box, without user guidance. Many of the options are meant to be used mostly by the Sledgehammer developers for experiments. Of course, feel free to try them out if you are so inclined.

6 Command Syntax

6.1 Sledgehammer

Sledgehammer can be invoked at any point when there is an open goal by entering the **sledgehammer** command in the theory file. Its general syntax is as follows:

sledgehammer ⟨*subcommand*⟩? ⟨*options*⟩? ⟨*facts_override*⟩? ⟨*num*⟩?

In the general syntax, the ⟨*subcommand*⟩ may be any of the following:

- ***run* (the default):** Runs Sledgehammer on subgoal number ⟨*num*⟩ (1 by default), with the given options and facts.
- ***messages*:** Redisplays recent messages issued by Sledgehammer. This allows you to examine results that might have been lost due to Sledgehammer's asynchronous nature. The ⟨*num*⟩ argument specifies a limit on the number of messages to display (10 by default).
- ***supported_provers*:** Prints the list of automatic provers supported by Sledgehammer. See § 2 and § 7.1 for more information on how to install automatic provers.
- ***running_provers*:** Prints information about currently running automatic provers, including elapsed runtime and remaining time until timeout.
- ***kill_all*:** Terminates all running threads (automatic provers and machine learners).
- ***refresh_tptp*:** Refreshes the list of remote ATPs available at SystemOnTPTP [18].

In addition, the following subcommands provide finer control over machine learning with MaSh:

- ***unlearn*:** Resets MaSh, erasing any persistent state.
- ***learn_isar*:** Invokes MaSh on the current theory to process all the available facts, learning from their Isabelle/Isar proofs. This happens automatically at Sledgehammer invocations if the *learn* option (§ 7.2) is enabled.

- ***learn_prover*:** Invokes MaSh on the current theory to process all the available facts, learning from proofs generated by automatic provers. The prover to use and its timeout can be set using the *prover* (§ 7.1) and *timeout* (§ 7.6) options. It is recommended to perform learning using an efficient first-order ATP (such as E, SPASS, and Vampire) as opposed to a higher-order ATP or an SMT solver.
- ***relearn_isar*:** Same as *unlearn* followed by *learn_isar*.
- ***relearn_prover*:** Same as *unlearn* followed by *learn_prover*.
- ***running_learners*:** Prints information about currently running machine learners, including elapsed runtime and remaining time until timeout.

Sledgehammer's behavior can be influenced by various ⟨*options*⟩, which can be specified in brackets after the **sledgehammer** command. The ⟨*options*⟩ are a list of key–value pairs of the form "$[k_1 = v_1, \ldots, k_n = v_n]$". For Boolean options, "= *true*" is optional. For example:

sledgehammer [*isar_proofs*, *timeout* = 120]

Default values can be set using **sledgehammer_params**:

sledgehammer_params ⟨*options*⟩

The supported options are described in § 7.

The ⟨*facts_override*⟩ argument lets you alter the set of facts that go through the relevance filter. It may be of the form "(⟨*facts*⟩)", where ⟨*facts*⟩ is a space-separated list of Isabelle facts (theorems, local assumptions, etc.), in which case the relevance filter is bypassed and the given facts are used. It may also be of the form "(*add*: ⟨*facts*$_1$⟩)", "(*del*: ⟨*facts*$_2$⟩)", or "(*add*: ⟨*facts*$_1$⟩ *del*: ⟨*facts*$_2$⟩)", where the relevance filter is instructed to proceed as usual except that it should consider ⟨*facts*$_1$⟩ highly-relevant and ⟨*facts*$_2$⟩ fully irrelevant.

If you use Isabelle/jEdit, Sledgehammer also provides an automatic mode that can be enabled via the "Auto Sledgehammer" option under "Plugins > Plugin Options > Isabelle > General." For automatic runs, only the first prover set using *provers* (§ 7.1) is considered (typically E), *slice* (§ 7.1) is disabled, *minimize* (§ 7.1) is disabled, fewer facts are passed to the prover, *fact_filter* (§ 7.2) is set to *mepo*, *strict* (§ 7.3) is enabled, *verbose* (§ 7.4) and *debug* (§ 7.4) are disabled, *preplay_timeout* (§ 7.6) is set to 0, and *timeout* (§ 7.6) is superseded by the "Auto Time Limit" option in jEdit. Sledgehammer's output is also more concise.

6.2 Metis

The *metis* proof method has the syntax

metis (⟨*options*⟩)$^?$ ⟨*facts*⟩$^?$

where ⟨*facts*⟩ is a list of arbitrary facts and ⟨*options*⟩ is a comma-separated list consisting of at most one λ translation scheme specification with the same semantics as Sledgehammer's *lam_trans* option (§ 7.3) and at most one type encoding specification with the same semantics as Sledgehammer's *type_enc* option (§ 7.3). The supported λ translation schemes are *hide_lams*, *lifting*, and *combs* (the default). All the untyped type encodings listed in § 7.3 are supported. For convenience, the following aliases are provided:

- ***full_types*:** Alias for *poly_guards_query*.
- ***partial_types*:** Alias for *poly_args*.
- ***no_types*:** Alias for *erased*.

7 Option Reference

Sledgehammer's options are categorized as follows: mode of operation (§ 7.1), problem encoding (§ 7.3), relevance filter (§ 7.2), output format (§ 7.4), regression testing (§ 7.5), and timeouts (§ 7.6).

The descriptions below refer to the following syntactic quantities:

- ⟨***string***⟩: A string.
- ⟨***bool***⟩: *true* or *false*.
- ⟨***smart_bool***⟩: *true*, *false*, or *smart*.
- ⟨***int***⟩: An integer.
- ⟨***float***⟩: A floating-point number (e.g., 2.5 or 60) expressing a number of seconds.
- ⟨***float_pair***⟩: A pair of floating-point numbers (e.g., 0.6 0.95).
- ⟨***smart_int***⟩: An integer or *smart*.

Default values are indicated in curly brackets ({}). Boolean options have a negative counterpart (e.g., *blocking* vs. *non_blocking*). When setting Boolean options or their negative counterparts, "= *true*" may be omitted.

7.1 Mode of Operation

[*provers* =] ⟨*string*⟩

Specifies the automatic provers to use as a space-separated list (e.g., "*e spass remote_vampire*"). Provers can be run locally or remotely; see § 2 for installation instructions.

The following local provers are supported:

- ***agsyhol*:** AgsyHOL is an automatic higher-order prover developed by Fredrik Lindblad [13], with support for the TPTP typed higher-order syntax (THF0). To use AgsyHOL, set the environment variable `AGSYHOL_HOME` to the directory that contains the `agsyHOL` executable. Sledgehammer has been tested with version 1.0.
- ***alt_ergo*:** Alt-Ergo is a polymorphic ATP developed by Bobot et al. [4]. It supports the TPTP polymorphic typed first-order format (TFF1) via Why3 [5]. To use Alt-Ergo, set the environment variable `WHY3_HOME` to the directory that contains the `why3` executable. Sledgehammer requires Alt-Ergo 0.95.2 and Why3 0.83.
- ***cvc3*:** CVC3 is an SMT solver developed by Clark Barrett, Cesare Tinelli, and their colleagues [2]. To use CVC3, set the environment variable `CVC3_SOLVER` to the complete path of the executable, including the file name, or install the prebuilt CVC3 package from `http://isabelle.in.tum.de/components/`. Sledgehammer has been tested with versions 2.2 and 2.4.1.
- ***cvc4*:** CVC4 [1] is the successor to CVC3. To use CVC4, set the environment variable `CVC4_SOLVER` to the complete path of the executable, including the file name, or install the prebuilt CVC4 package from `http://isabelle.in.tum.de/components/`. Sledgehammer has been tested with version 1.5-prerelease.
- ***e*:** E is a first-order resolution prover developed by Stephan Schulz [15]. To use E, set the environment variable `E_HOME` to the directory that contains the `eproof` executable and `E_VERSION` to the version number (e.g., "1.8"), or install the prebuilt E package from `http://isabelle.in.tum.de/components/`. Sledgehammer has been tested with versions 1.6 to 1.8.
- ***e_males*:** E-MaLeS is a metaprover developed by Daniel Kühlwein that implements strategy scheduling on top of E. To

use E-MaLeS, set the environment variable `E_MALES_HOME` to the directory that contains the `emales.py` script. Sledgehammer has been tested with version 1.1.

- ***e_par*:** E-Par is an experimental metaprover developed by Josef Urban that implements strategy scheduling on top of E. To use E-Par, set the environment variable `E_HOME` to the directory that contains the `runepar.pl` script and the `eprover` and `epclextract` executables, or use the prebuilt E package from `http://isabelle.in.tum.de/components/`. Be aware that E-Par is experimental software. It has been known to generate zombie processes. Use at your own risks.
- ***iprover*:** iProver is a pure instantiation-based prover developed by Konstantin Korovin [11]. To use iProver, set the environment variable `IPROVER_HOME` to the directory that contains the `iprover` and `vclausify_rel` executables. Sledgehammer has been tested with version 0.99.
- ***iprover_eq*:** iProver-Eq is an instantiation-based prover with native support for equality developed by Konstantin Korovin and Christoph Sticksel [12]. To use iProver-Eq, set the environment variable `IPROVER_EQ_HOME` to the directory that contains the `iprover-eq` and `vclausify_rel` executables. Sledgehammer has been tested with version 0.8.
- ***leo2*:** LEO-II is an automatic higher-order prover developed by Christoph Benzmüller et al. [3], with support for the TPTP typed higher-order syntax (THF0). To use LEO-II, set the environment variable `LEO2_HOME` to the directory that contains the `leo` executable. Sledgehammer requires version 1.3.4 or above.
- ***satallax*:** Satallax is an automatic higher-order prover developed by Chad Brown et al. [7], with support for the TPTP typed higher-order syntax (THF0). To use Satallax, set the environment variable `SATALLAX_HOME` to the directory that contains the `satallax` executable. Sledgehammer requires version 2.2 or above.
- ***spass*:** SPASS is a first-order resolution prover developed by Christoph Weidenbach et al. [19]. To use SPASS, set the environment variable `SPASS_HOME` to the directory that contains the `SPASS` executable and `SPASS_VERSION` to the version number (e.g., "3.8ds"), or install the prebuilt SPASS package from `http://isabelle.in.tum.de/components/`. Sledgehammer requires version 3.8ds or above.

- ***vampire*:** Vampire is a first-order resolution prover developed by Andrei Voronkov and his colleagues [14]. To use Vampire, set the environment variable `VAMPIRE_HOME` to the directory that contains the `vampire` executable and `VAMPIRE_VERSION` to the version number (e.g., "3.0"). Sledgehammer has been tested with versions 0.6 to 3.0. Versions strictly above 1.8 support the TPTP typed first-order format (TFF0).
- ***verit*:** veriT [6] is an SMT solver developed by David DÃľharbe, Pascal Fontaine, and their colleagues. It is specifically designed to produce detailed proofs for reconstruction in proof assistants. To use veriT, set the environment variable `VERIT_SOLVER` to the complete path of the executable, including the file name. Sledgehammer has been tested with version smtcomp2014.
- ***z3*:** Z3 is an SMT solver developed at Microsoft Research [20]. To use Z3, set the environment variable `Z3_SOLVER` to the complete path of the executable, including the file name. Sledgehammer has been tested with a pre-release version of 4.4.0.
- ***z3_tptp*:** This version of Z3 pretends to be an ATP, exploiting Z3's support for the TPTP untyped and typed first-order formats (FOF and TFF0). It is included for experimental purposes. It requires version 4.3.1 of Z3 or above. To use it, set the environment variable `Z3_TPTP_HOME` to the directory that contains the `z3_tptp` executable.
- ***zipperposition*:** Zipperposition [8] is an experimental first-order resolution prover developed by Simon Cruane. To use Zipperposition, set the environment variable `ZIPPERPOSITION_HOME` to the directory that contains the `zipperposition` executable.

Moreover, the following remote provers are supported:

- ***remote_agsyhol*:** The remote version of AgsyHOL runs on Geoff Sutcliffe's Miami servers [18].
- ***remote_e*:** The remote version of E runs on Geoff Sutcliffe's Miami servers [18].
- ***remote_e_sine*:** E-SInE is a metaprover developed by Kryštof Hoder [10] based on E. It runs on Geoff Sutcliffe's Miami servers.
- ***remote_e_tofof*:** E-ToFoF is a metaprover developed by Geoff Sutcliffe [17] based on E running on his Miami servers. This ATP supports the TPTP typed first-order format (TFF0). The remote version of E-ToFoF runs on Geoff Sutcliffe's Miami servers.

- ***remote_iprover*:** The remote version of iProver runs on Geoff Sutcliffe's Miami servers [18].
- ***remote_iprover_eq*:** The remote version of iProver-Eq runs on Geoff Sutcliffe's Miami servers [18].
- ***remote_leo2*:** The remote version of LEO-II runs on Geoff Sutcliffe's Miami servers [18].
- ***remote_pirate*:** Pirate is a highly experimental first-order resolution prover developed by Daniel Wand. The remote version of Pirate run on a private server he generously set up.
- ***remote_satallax*:** The remote version of Satallax runs on Geoff Sutcliffe's Miami servers [18].
- ***remote_snark*:** SNARK is a first-order resolution prover developed by Stickel et al. [16]. It supports the TPTP typed first-order format (TFF0). The remote version of SNARK runs on Geoff Sutcliffe's Miami servers.
- ***remote_vampire*:** The remote version of Vampire runs on Geoff Sutcliffe's Miami servers.
- ***remote_waldmeister*:** Waldmeister is a unit equality prover developed by Hillenbrand et al. [9]. It can be used to prove universally quantified equations using unconditional equations, corresponding to the TPTP CNF UEQ division. The remote version of Waldmeister runs on Geoff Sutcliffe's Miami servers.

By default, Sledgehammer runs a subset of CVC4, E, E-SInE, SPASS, Vampire, veriT, and Z3 in parallel, either locally or remotely—depending on the number of processor cores available and on which provers are actually installed. It is generally a good idea to run several provers in parallel.

prover $= \langle$***string***$\rangle$

Alias for *provers*.

blocking $\left[= \langle\textit{bool}\rangle\right]$ **{*false*}** **(neg.: *non_blocking*)**

Specifies whether the **sledgehammer** command should operate synchronously. The asynchronous (non-blocking) mode lets the user start proving the putative theorem manually while Sledgehammer looks for a proof, but it can also be more confusing. Irrespective of the value of this option, Sledgehammer is always run synchronously if *debug* (§ 7.4) is enabled.

slice [= ⟨***bool***⟩] **{*true*}** **(neg.: *dont_slice*)**

Specifies whether the time allocated to a prover should be sliced into several segments, each of which has its own set of possibly prover-dependent options. For SPASS and Vampire, the first slice tries the fast but incomplete set-of-support (SOS) strategy, whereas the second slice runs without it. For E, up to three slices are tried, with different weighted search strategies and number of facts. For SMT solvers, several slices are tried with the same options each time but fewer and fewer facts. According to benchmarks with a timeout of 30 seconds, slicing is a valuable optimization, and you should probably leave it enabled unless you are conducting experiments.

See also *verbose* (§ 7.4).

minimize [= ⟨***bool***⟩] **{*true*}** **(neg.: *dont_minimize*)**

Specifies whether the minimization tool should be invoked automatically after proof search.

See also *preplay_timeout* (§ 7.6) and *dont_preplay* (§ 7.6).

spy [= ⟨***bool***⟩] **{*false*}** **(neg.: *dont_spy*)**

Specifies whether Sledgehammer should record statistics in `$ISABELLE_HOME_USER/spy_sledgehammer`. These statistics can be useful to the developers of Sledgehammer. If you are willing to have your interactions recorded in the name of science, please enable this feature and send the statistics file every now and then to the author of this manual (`blanchette@in.tum.de`). To change the default value of this option globally, set the environment variable `SLEDGEHAMMER_SPY` to *yes*.

See also *debug* (§ 7.4).

overlord [= ⟨***bool***⟩] **{*false*}** **(neg.: *no_overlord*)**

Specifies whether Sledgehammer should put its temporary files in `$ISABELLE_HOME_USER`, which is useful for debugging Sledgehammer but also unsafe if several instances of the tool are run simultaneously. The files are identified by the prefixes `prob_` and `mash_`; you may safely remove them after Sledgehammer has run.

Warning: This option is not thread-safe. Use at your own risks.

See also *debug* (§ 7.4).

7.2 Relevance Filter

fact_filter = ⟨***string***⟩ **{*smart*}**

Specifies the relevance filter to use. The following filters are available:

- ***mepo*:** The traditional memoryless MePo relevance filter.
- ***mash*:** The MaSh machine learner. Three learning algorithms are provided:
  - ***nb*** is an implementation of naive Bayes.
  - ***knn*** is an implementation of k-nearest neighbors.
  - ***nb_knn*** (also called ***yes*** and ***sml***) is a combination of naive Bayes and k-nearest neighbors.

  In addition, the special value *none* is used to disable machine learning by default (cf. *smart* below).

  The default algorithm is *nb_knn*. The algorithm can be selected by setting `MASH`—either in the environment in which Isabelle is launched, in your `$ISABELLE_HOME_USER/etc/settings` file, or via the "MaSh" option under "Plugins > Plugin Options > Isabelle > General" in Isabelle/jEdit. Persistent data for both algorithms is stored in the directory `$ISABELLE_HOME_USER/mash`.
- ***mesh*:** The MeSh filter, which combines the rankings from MePo and MaSh.
- ***smart*:** A combination of MePo, MaSh, and MeSh. If the learning algorithm is set to be *none*, *smart* behaves like MePo.

max_facts = ⟨***smart_int***⟩ **{*smart*}**

Specifies the maximum number of facts that may be returned by the relevance filter. If the option is set to *smart* (the default), it effectively takes a value that was empirically found to be appropriate for the prover. Typical values lie between 50 and 1000.

fact_thresholds = ⟨***float_pair***⟩ **{0.45 0.85}**

Specifies the thresholds above which facts are considered relevant by the relevance filter. The first threshold is used for the first iteration of the relevance filter and the second threshold is used for the last iteration (if it is reached). The effective threshold is quadratically interpolated for the other iterations. Each threshold ranges from 0 to 1, where 0 means that all theorems are relevant and 1 only theorems that refer to previously seen constants.

learn $[= \langle$***bool***$\rangle]$ **{*true*}** **(neg.: *dont_learn*)**

Specifies whether MaSh should be run automatically by Sledgehammer to learn the available theories (and hence provide more accurate results). Learning takes place only if MaSh is enabled.

max_new_mono_instances $= \langle$***int***$\rangle$ **{*smart*}**

Specifies the maximum number of monomorphic instances to generate beyond *max_facts*. The higher this limit is, the more monomorphic instances are potentially generated. Whether monomorphization takes place depends on the type encoding used. If the option is set to *smart* (the default), it takes a value that was empirically found to be appropriate for the prover. For most provers, this value is 100.

See also *type_enc* (§ 7.3).

max_mono_iters $= \langle$***int***$\rangle$ **{*smart*}**

Specifies the maximum number of iterations for the monomorphization fixpoint construction. The higher this limit is, the more monomorphic instances are potentially generated. Whether monomorphization takes place depends on the type encoding used. If the option is set to *smart* (the default), it takes a value that was empirically found to be appropriate for the prover. For most provers, this value is 3.

See also *type_enc* (§ 7.3).

7.3 Problem Encoding

lam_trans $= \langle$***string***$\rangle$ **{*smart*}**

Specifies the λ translation scheme to use in ATP problems. The supported translation schemes are listed below:

- ***hide_lams*****:** Hide the λ-abstractions by replacing them by unspecified fresh constants, effectively disabling all reasoning under λ-abstractions.
- ***lifting*****:** Introduce a new supercombinator c for each cluster of n λ-abstractions, defined using an equation $\mathsf{c}\ x_1\ \ldots\ x_n = t$ (λ-lifting).
- ***combs*****:** Rewrite lambdas to the Curry combinators (I, K, S, B, C). Combinators enable the ATPs to synthesize λ-terms but tend to yield bulkier formulas than λ-lifting: The translation is quadratic

in the worst case, and the equational definitions of the combinators are very prolific in the context of resolution.

- ***combs_and_lifting*:** Introduce a new supercombinator `c` for each cluster of λ-abstractions and characterize it both using a lifted equation $\mathsf{c}\ x_1\ \ldots\ x_n = t$ and via Curry combinators.
- ***combs_or_lifting*:** For each cluster of λ-abstractions, heuristically choose between λ-lifting and Curry combinators.
- ***keep_lams*:** Keep the λ-abstractions in the generated problems. This is available only with provers that support the THF0 syntax.
- ***smart*:** The actual translation scheme used depends on the ATP and should be the most efficient scheme for that ATP.

For SMT solvers, the λ translation scheme is always *lifting*, irrespective of the value of this option.

uncurried_aliases $[= \langle$***smart_bool***$\rangle]$ **{*smart*}**
(neg.: *no_uncurried_aliases*)

Specifies whether fresh function symbols should be generated as aliases for applications of curried functions in ATP problems.

type_enc $= \langle$***string***$\rangle$ **{*smart*}**

Specifies the type encoding to use in ATP problems. Some of the type encodings are unsound, meaning that they can give rise to spurious proofs (unreconstructible using *metis*). The type encodings are listed below, with an indication of their soundness in parentheses. An asterisk (*) indicates that the encoding is slightly incomplete for reconstruction with *metis*, unless the *strict* option (described below) is enabled.

- ***erased*** **(unsound):** No type information is supplied to the ATP, not even to resolve overloading. Types are simply erased.
- ***poly_guards*** **(sound):** Types are encoded using a predicate $\mathsf{g}(\tau, t)$ that guards bound variables. Constants are annotated with their types, supplied as extra arguments, to resolve overloading.
- ***poly_tags*** **(sound):** Each term and subterm is tagged with its type using a function $\mathsf{t}(\tau, t)$.
- ***poly_args*** **(unsound):** Like for *poly_guards* constants are annotated with their types to resolve overloading, but otherwise no type information is encoded. This is the default encoding used by the *metis* proof method.

- ***raw_mono_guards*, *raw_mono_tags* (sound); *raw_mono_args* (unsound):**
  Similar to *poly_guards*, *poly_tags*, and *poly_args*, respectively, but the problem is additionally monomorphized, meaning that type variables are instantiated with heuristically chosen ground types. Monomorphization can simplify reasoning but also leads to larger fact bases, which can slow down the ATPs.
- ***mono_guards*, *mono_tags* (sound); *mono_args* (unsound):**
  Similar to *raw_mono_guards*, *raw_mono_tags*, and *raw_mono_args*, respectively but types are mangled in constant names instead of being supplied as ground term arguments. The binary predicate $\mathtt{g}(\tau, t)$ becomes a unary predicate $\mathtt{g\_}\tau(t)$, and the binary function $\mathtt{t}(\tau, t)$ becomes a unary function $\mathtt{t\_}\tau(t)$.
- ***mono_native* (sound):** Exploits native first-order types if the prover supports the TFF0, TFF1, or THF0 syntax; otherwise, falls back on *mono_guards*. The problem is monomorphized.
- ***mono_native_higher* (sound):** Exploits native higher-order types if the prover supports the THF0 syntax; otherwise, falls back on *mono_native* or *mono_guards*. The problem is monomorphized.
- ***poly_native* (sound):** Exploits native first-order polymorphic types if the prover supports the TFF1 syntax; otherwise, falls back on *mono_native*.
- ***poly_guards*?, *poly_tags*?, *raw_mono_guards*?, *raw_mono_tags*?, *mono_guards*?, *mono_tags*?, *mono_native*? (sound*):**
  The type encodings *poly_guards*, *poly_tags*, *raw_mono_guards*, *raw_mono_tags*, *mono_guards*, *mono_tags*, and *mono_native* are fully typed and sound. For each of these, Sledgehammer also provides a lighter variant identified by a question mark ('?') that detects and erases monotonic types, notably infinite types. (For *mono_native*, the types are not actually erased but rather replaced by a shared uniform type of individuals.) As argument to the *metis* proof method, the question mark is replaced by a "*_query*" suffix.
- ***poly_guards*??, *poly_tags*??, *raw_mono_guards*??, *raw_mono_tags*??, *mono_guards*??, *mono_tags*?? (sound*):**

Even lighter versions of the '?' encodings. As argument to the *metis* proof method, the '??' suffix is replaced by "*_query_query*".

- ***poly_guards@*, *poly_tags@*, *raw_mono_guards@*, *raw_mono_tags@* (sound*):**
  Alternative versions of the '??' encodings. As argument to the *metis* proof method, the '@' suffix is replaced by "*_at*".
- ***poly_args?*, *raw_mono_args?* (unsound):**
  Lighter versions of *poly_args* and *raw_mono_args*.
- ***smart*:** The actual encoding used depends on the ATP and should be the most efficient sound encoding for that ATP.

For SMT solvers, the type encoding is always *mono_native*, irrespective of the value of this option.

See also *max_new_mono_instances* (§ 7.2) and *max_mono_iters* (§ 7.2).

strict [= ⟨***bool***⟩] **{*false*}** **(neg.: *non_strict*)**

Specifies whether Sledgehammer should run in its strict mode. In that mode, sound type encodings marked with an asterisk (*) above are made complete for reconstruction with *metis*, at the cost of some clutter in the generated problems. This option has no effect if *type_enc* is deliberately set to an unsound encoding.

7.4 Output Format

verbose [= ⟨***bool***⟩] **{*false*}** **(neg.: *quiet*)**

Specifies whether the **sledgehammer** command should explain what it does.

debug [= ⟨***bool***⟩] **{*false*}** **(neg.: *no_debug*)**

Specifies whether Sledgehammer should display additional debugging information beyond what *verbose* already displays. Enabling *debug* also enables *verbose* and *blocking* (§ 7.1) behind the scenes.

See also *spy* (§ 7.1) and *overlord* (§ 7.1).

isar_proofs [= ⟨***smart_bool***⟩] **{*smart*}** **(neg.: *no_isar_proofs*)**

Specifies whether Isar proofs should be output in addition to one-line proofs. The construction of Isar proof is still experimental and may sometimes fail; however, when they succeed they are usually faster and more intelligible than one-line proofs. If the option is set to *smart* (the

default), Isar proofs are only generated when no working one-line proof is available.

compress $= \langle$***int***$\rangle$ **{*smart*}**

Specifies the granularity of the generated Isar proofs if *isar_proofs* is explicitly enabled. A value of n indicates that each Isar proof step should correspond to a group of up to n consecutive proof steps in the ATP proof. If the option is set to *smart* (the default), the compression factor is 10 if the *isar_proofs* option is explicitly enabled; otherwise, it is ∞.

dont_compress $[=$ ***true***$]$

Alias for "*compress* = 1".

try0 $[=\langle$***bool***$\rangle]$ **{*true*}** **(neg.: *dont_try0*)**

Specifies whether standard proof methods such as *auto* and *blast* should be tried as alternatives to *metis* in Isar proofs. The collection of methods is roughly the same as for the **try0** command.

smt_proofs $[=\langle$***smart_bool***$\rangle]$ **{*smart*}** **(neg.: *no_smt_proofs*)**

Specifies whether the *smt2* proof method should be tried in addition to Isabelle's other proof methods. If the option is set to *smart* (the default), the *smt2* method is used for one-line proofs but not in Isar proofs.

7.5 Regression Testing

expect $= \langle$***string***$\rangle$

Specifies the expected outcome, which must be one of the following:

- ***some*****:** Sledgehammer found a proof.
- ***none*****:** Sledgehammer found no proof.
- ***timeout*****:** Sledgehammer timed out.
- ***unknown*****:** Sledgehammer encountered some problem.

Sledgehammer emits an error (if *blocking* is enabled) or a warning (otherwise) if the actual outcome differs from the expected outcome. This option is useful for regression testing.

See also *blocking* (§ 7.1) and *timeout* (§ 7.6).

7.6 Timeouts

timeout $= \langle$***float***$\rangle$ **{30}**

Specifies the maximum number of seconds that the automatic provers should spend searching for a proof. This excludes problem preparation and is a soft limit.

preplay_timeout $= \langle$***float***$\rangle$ **{1}**

Specifies the maximum number of seconds that *metis* or other proof methods should spend trying to "preplay" the found proof. If this option is set to 0, no preplaying takes place, and no timing information is displayed next to the suggested proof method calls.

See also *minimize* (§ 7.1).

dont_preplay $[=$ ***true***$]$

Alias for "*preplay_timeout* = 0".

References

[1] C. Barrett, C. L. Conway, M. Deters, L. Hadarean, D. Jovanovic, T. King, A. Reynolds, and C. Tinelli. CVC4. In G. Gopalakrishnan and S. Qadeer, editors, *CAV 2011*, volume 6806 of *Lecture Notes in Computer Science*, pages 171–177. Springer, 2011.

[2] C. Barrett and C. Tinelli. CVC3. In W. Damm and H. Hermanns, editors, *CAV*, volume 4590 of *Lecture Notes in Computer Science*, pages 298–302. Springer, 2007.

[3] C. Benzmüller, L. C. Paulson, F. Theiss, and A. Fietzke. LEO-II—a cooperative automatic theorem prover for higher-order logic. In A. Armando, P. Baumgartner, and G. Dowek, editors, *Automated Reasoning: IJCAR 2008*, volume 5195 of *Lecture Notes in Computer Science*, pages 162–170. Springer-Verlag, 2008.

[4] F. Bobot, S. Conchon, E. Contejean, and S. Lescuyer. Implementing polymorphism in SMT solvers. In C. Barrett and L. de Moura, editors, *SMT '08*, ICPS, pages 1–5. ACM, 2008.

[5] F. Bobot, J.-C. Filliâtre, C. Marché, and A. Paskevich. Why3: Shepherd your herd of provers. In K. R. M. Leino and M. Moskal, editors, *Boogie 2011*, pages 53–64, 2011.

[6] T. Bouton, D. C. B. de Oliveira, D. Déharbe, and P. Fontaine. veriT: An open, trustable and efficient SMT-solver. In R. A. Schmidt, editor, *Automated Deduction — CADE-22*, volume 5663 of *Lecture Notes in Computer Science*, pages 151–156. Springer, 2009.

[7] C. E. Brown. Reducing higher-order theorem proving to a sequence of SAT problems. In N. Bjørner and V. Sofronie-Stokkermans, editors, *Automated Deduction — CADE-23*, volume 6803 of *Lecture Notes in Computer Science*, pages 147–161. Springer-Verlag, 2011.

[8] S. Cruanes. Logtk: A Logic ToolKit for automated reasoning, and its implementation. 2014. Presented at the Practical Aspects of Automated Reasoning (PAAR) workshop.

[9] T. Hillenbrand, A. Buch, R. Vogt, and B. Löchner. Waldmeister: High-performance equational deduction. *Journal of Automated Reasoning*, 18(2):265–270, 1997.

[10] K. Hoder and A. Voronkov. Sine qua non for large theory reasoning. In N. Bjørner and V. Sofronie-Stokkermans, editors, *Automated Deduction — CADE-23*, volume 6803 of *Lecture Notes in Computer Science*, pages 299–314. Springer-Verlag, 2011.

[11] K. Korovin. Instantiation-based automated reasoning: From theory to practice. In R. A. Schmidt, editor, *Automated Deduction — CADE-22*, volume 5663 of *LNAI*, pages 163–166. Springer, 2009.

[12] K. Korovin and C. Sticksel. iProver-Eq: An instantiation-based theorem prover with equality. In J. Giesl and R. Hähnle, editors, *Automated Reasoning: IJCAR 2010*, volume 6173 of *Lecture Notes in Computer Science*, pages 196–202. Springer-Verlag, 2010.

[13] F. Lindblad. agsyHOL. `https://github.com/frelindb/agsyHOL`.

[14] A. Riazanov and A. Voronkov. The design and implementation of Vampire. *Journal of AI Communications*, 15(2/3):91–110, 2002.

[15] S. Schulz. E—a brainiac theorem prover. *Journal of AI Communications*, 15(2/3):111–126, 2002.

[16] M. Stickel, R. Waldinger, M. Lowry, T. Pressburger, and I. Underwood. Deductive composition of astronomical software from subroutine libraries. In A. Bundy, editor, *Automated Deduction — CADE-12 International Conference*, LNAI 814, pages 341–355. Springer, 1994.

[17] G. Sutcliffe. ToFoF. http://www.cs.miami.edu/~tptp/ATPSystems/ToFoF/.

[18] G. Sutcliffe. System description: SystemOnTPTP. In D. McAllester, editor, *Automated Deduction — CADE-17 International Conference*, volume 1831 of *Lecture Notes in Artificial Intelligence*, pages 406–410. Springer-Verlag, 2000.

[19] C. Weidenbach, D. Dimova, A. Fietzke, R. Kumar, M. Suda, and P. Wischnewski. SPASS version 3.5. http://www.spass-prover.org/publications/spass.pdf.

[20] Z3: An efficient SMT solver. http://research.microsoft.com/en-us/um/redmond/projects/z3/.