## 1. ATPs usage

In this section we provide basic guidance on how to use ATPs from a practical point of view, based on both the existing academic information, and our experience and available evidence[1] within the ETP. When the project started, contributors had varying degrees of ATPs' knowledge and utilisation skills, with several of us having to start using them from the ground up. With hindsight, several of the project computations could have been approached in better ways, their difficulty diminished due to better ATP expertise. Accordingly, we provide here a collection of facts and hints about ATP usage, primarily with the algebraist working with (unsorted) equational theories in mind. This is the brief guide we would have liked to have at our disposal at the beginning of this project. We also make a conscious effort at documentation, as the information we provide here is scattered around the web and the literature, or contained in non-persistent sources, or contradictory at times (as when the most reachable source informs about an outdated version), or even not published anywhere else.

We restrict our attention to Vampire and Prover9-Mace4, as those have been the main tools used for exploration of implications and anti-implications along the ETP.

### 1.1. **Vampire and Prover9-Mace4.**

Vampire is a saturation-based, world-leading ATP, a 78-time winner of the CASC[2] in the period 1999-2025, and the winner in all categories of the 2025 contest (meaning it was the ATP solving the most problems in each category). Vampire is a complex and sophisticated software system that has an active community of developers[3] and has experienced a steady flow of important developments in the last 10 years, up to the newly (Sept'25) released version 5.0. At present, in addition to classical first-order reasoning, Vampire is able to perform reasoning in multi-sorted logic with theories, polymorphic logic, higher-order logic, linear (integer, real, or mixed) arithmetic, induction, and program analysis and verification. It can also be used for database querying. Lastly, Vampire includes a SAT-based finite model builder. For the most of the ETP, we worked with version 4.9 of Vampire; both versions differ significantly, although both excelled at CASC-30.

In contrast, Prover9-Mace4 was mainly developed by the late William McCune[4] up to 2009[5], and is virtually abandoned software[6]. Its main purpose is the study of equational logic with

[1]The timings presented here cannot be taken as benchmarking. Different experiments were run in different computers, with different software and OS (Ubuntu, Windows 10...), with a variable number of parallel processes (from both inside and outside the experiment), and using the `nice` command in Linux.

[2]The CADE ATP System Competition (CASC, `https://tptp.org/CASC/`) is an annual contest, the world championship of automated theorem proving. It is divided into divisions according to problem characteristics, with some further divided into problem categories, such as the FOF division for first-order theorems, subdivided in the FEQ (FOF with equality) and FNE (FOF with no equality) categories.

[3]Currently organized around GitHub, `https://github.com/vprover/vampire`.

[4]With assistance from some collaborators, notably Bob Veroff.

[5]The latest official version is LADR-2009-11A.

[6]In fact, the Makefile in the Linux install needs a fix in order to compile with newer versions of gcc. A fixed version of this file can be found at `[ETPLINK]`.

operations. At CASC, it is still considered the lowest benchmark against which the rest of ATPs have to be measured[7]. There have been some unofficial Prover9-Mace4 updates, notably the 2017 version developed by Veroff to expand the hints functionality[8] (explained in Section 1), and the ProverX extension developed by Ivo Robert[9], that provides a web GUI with a scripting language and some extra functionality like interaction with GAP. Interestingly, Prover9-Mace4 also provides an official GUI which features a previous version[10] with different behaviour. In the ETP we used both the 2009 and the GUI versions of Prover9-Mace4. Prover9 is a saturation-based ATP, while Mace4 is a DPLL-based but not SAT-based finite model builder[11].

Although older and discontinued, and despite the existence of the all-round and powerful Vampire, which is a marvellous instrument itself, we think Prover9-Mace4 is still a valuable tool to have at disposal, for several reasons:

• Prover9-Mace4 is ideal to get started in the usage of ATPs: it is simple, well documented, and includes a user-friendly GUI which allows to see and configure all options in a nutshell. In contrast, being able to configure Vampire for a specific problem has a steeper learning curve.
• In general, given a batch of problems on which one wants to work, it is useful to have several ATPs when they complement each other on the batch -i.e., when they can solve different parts of the batch, so that altogether they can prove more theorems than any of them alone. This is often the case with Vampire and Prover9: see e.g. [?, Figure 3] (study from 2024, using Vampire 4.5.1 and 4.8), where it is shown that from a batch of around 770 TPTP problems[12] solved with Vampire and Prover9 together (with some restrictions), around 60% is solved by both, 20% is solved only by Vampire, and the remaining 20% is solved only by Prover9.[13]
• Prover9 is still very useful with equational logic and algebraic problems. Within the ETP we likewise identified several problems that were easier to solve with Prover9 than with Vampire, and even some solved by Prover9 but not by Vampire (with the configurations we tried). The most salient example is the proof of E102744082 implying the injectivity of the right multiplication map, used in the Higman-Neumann side project (see Section ??), which was originally found by Prover9 in several hours with parameters chosen to produce a big search space; an optimized choice of Prover9 options lowers the runtime to 0.2s. Upon contact with the Vampire development team, after several attempts they were able to provide

[7]`https://tptp.org/CASC/30/SystemDescriptions.html#Prover9---1109a`.

[8]Version LADR-2017-11A (see [?]). A GitHub clone of this version can be found at `https://github.com/ai4reason/Prover9`. A changelog with Veroff's updates since 2009 is found at `https://github.com/ai4reason/Prover9/blob/master/Changelog`.

[9]`https://www.proverx.dm.fct.unl.pt`, [?].

[10]LADR-dec-2007.

[11]Counterintuitively, finite model builders based on a SAT solver are called MACE-style, because the earliest versions of the Mace program pioneered this approach; but Mace4 itself uses term equations in what is called the SEM style.

[12]The Thousand Problems for Theorem Provers (TPTP, `https://tptp.org/TPTP/`) is a referential and freely available collection of problems for ATPs.

[13]In contrast, in said study the problems solved by the E prover happened to be a subset of those proved by Vampire, except for one.

a Vampire configuration (inspired by the Prover9 optimization) which produces a proof in 3s.
• Mace4 non-SAT algorithm makes it faster in general at finding models than generic SAT-based finite model builders. In particular for the ETP problems, Mace4 is faster than Vampire's finite model builder in both tasks of finding a model for some size and of exhausting a size without any models. For example, for E677 models, Mace4 is able to exhaust size 8 in 1.5s and to find a model of size 9 in 16s (see details in Section 1.5), while Vampire's finite model builder is unable to perform any of these tasks in 10 minutes.
• Prover9 includes some features that Vampire doesn't (to the best of our knowledge), such as the use of hints, semantic guidance, and user-specified weight limits (see below for explanations).

1.1.1. *Documentation.* Vampire does not yet have a user manual[14]. Presently, the main sources for learning about the internal workings of Vampire are the articles [?] (written for version 2.6) and the recent [?], which overviews the updates to Vampire's system since the previous reference. The various slide files from a tutorial[15] delivered at the ASE 2017 conference are also instructive (including information on the theoretical foundations, finite model building, reasoning in theories, etc.). Unfortunately, there is an absence of documentation detailing the more than 200 options offered to users by Vampire[16]; as far as we know, the best way to proceed is to make Vampire show all its options together with some basic explanations (with the command `--show_options on`) and save them to a file[17]. Alternatively, when the option name is known, its explanation can be recovered with the command `-explain <option>`.

Several Vampire subsystems have names of their own, and familiarity with them aids in understanding Vampire's options:

- *Spider*: Given a set of problems, prepares and optimizes a portfolio for it.
- *SInE*: From a large set of axioms, tries to select a smaller subset of the axioms most relevant to the goal.
- *AVATAR*: Allows theory reasoning within superposition by allowing generated ground clauses to be passed to SMT solvers.
- *ALASCA*: A superposition calculus for reasoning in arithmetic (integer, real, and mixed).
- *FOOL*: A logic suited for program analysis that contains a fixed Boolean sort, if-then-else expressions, and rich let-in expressions.

Prover9 is accompanied with an online manual[18] in HTML which is well written and details almost every aspect of the program. It includes a section on Mace4 with limited detail, which in particular does not describe its internal workings nor its advanced search options; for that, one has to consult the original Mace4 manual ([?]). Other documents containing

[14]A detailed tutorial is currently in development at `https://vprover.github.io/vampireGuide`.
[15]`https://github.com/vprover/ase17tutorial`.
[16]But note that not all of them are useful for equational theories.
[17]This file is about 44 pages long for version 4.9, and about 46 for version 5.0.
[18]`https://www.cs.unm.edu/~mccune/prover9/manual/2009-11A`.

important information about Prover9-Mace4 are the LADR changelog[19], the GUI help, and the output from the command `-help` for the various LADR programs.

1.1.2. *Basic usage.* Vampire[20] is run from a Linux terminal; for Windows usage one can run an Ubuntu environment through the Windows Subsystem for Linux (WSL). Similarly, Prover9-Mace4[21] runs from a Linux terminal (on Windows, through WSL), although its GUI is versioned for Linux, Windows and macOS (modulo installation problems that may appear due to obsolescence). All options can be given to Vampire as commands directly in the shell; in contrast, some options can only be given to Prover9-Mace4 through the input file.

When running an ATP or finite model builder to solve a nontrivial problem, the two essential pieces of advice to take into consideration are: 1) Produce several runs modifying the search parameters, and 2) provide enough computational resources for each search: memory[22], number of processing cores[23], number of user instructions executed[24], and specially, time. Along the ETP, some proofs or models which couldn't be found in under 1 second with some configuration, could be found in under (say) 8 seconds without changing the configuration, and there were implications that initally could only be proved by runs lasting several minutes, or even hours. Once a proof is found by some configuration, typically a short and quick proof can be found by tweaking that configuration.

In Vampire, options are specified with `--option <value>`, and many options are abbreviated with `-option_abbreviature`. The options restricting the resources are `--time_limit` (`-t`), `--memory_limit` (`-m`), `--instructions_limit` (`-i`), and `--cores`.[25]

The configuration of the search parameters is a difficult art that occupies a substantial part of this tutorial. As it is well said in [?], with empirical support from [?],

> Although expert users may sometimes have an idea of which options could be well suited to tackle a problem, the prover's behavior tends to be so chaotic that even expert hunches often fail.

1.1.3. *Flow control.* Vampire is equipped with a user-friendly and powerful standard mode, that in many situations avoids the need of configuring specific search options for the given

[19]`https://www.cs.unm.edu/~mccune/prover9/download/Changelog.txt`, a copy is found at `[ETPLINK]`.

[20]Last version downloadable from `https://github.com/vprover/vampire/releases`.

[21]To install, download the LADR package at `https://www.cs.unm.edu/~mccune/prover9/download`.

[22]If a memory limit is not specified, Vampire will try to use as much RAM as possible, but by default Prover9-Mace4 sets memory limits which are rather low for current standards, so we advise raising them before starting a long computation.

[23]Vampire can make use of as many cores as the user specifies. To take advantage of multicore processors with Prover9-Mace4, one can run several terminal or GUI instances. Moreover, each GUI window allows to run Prover9 and Mace4 simultaneously on the same problem, and they run on different cores.

[24]According to [?], Vampire 5.0 can prove all true implications from the ETP (which are more than 8 million), and can prove 99.97% of them using less than 500 instructions per proof.

[25]The number of cores is 1 by default. Set it to 0 to use all available ones.

problem. This mode is run with the command `--mode casc`.[26] Hence, to run a first try on a problem (specifying resources restrictions), one would command

```
vampire -t <seconds> -m <MB> -i <Mi> --cores <number> --mode casc problem.p
```

To understand what this mode means we need some ATP definitions: Technically, an *option* is a pair parameter-value. A collection of options is called a *strategy* for the ATP. A sequence of strategies together with assigned time limits is a *schedule*. And given a batch of problems, a *portfolio* for that batch is a schedule whose strategies are known to be good at collectively solving its problems. Vampire's casc mode (or mode portfolio schedule casc) therefore invokes the schedule that Vampire includes for solving the batch of problems from the last CASC competition, produced by putting problems in classes, each class with its sequence of strategies. The time given to each strategy is also important: counterintuitively, giving less time to the casc mode may end up producing a faster proof[27].

Note that the casc mode may be overfitted towards the problems appearing in the competition. To alleviate this problem we can use the casc mode while forcing some options to specified values, by adding the command

```
--forced_options <opt1>=<val1>:<opt2>=<val2>:...:<optN>=<valN>.
```

The tool Vampire uses to determine a portfolio from a batch of problems is called *Spider*. It has been one of the main weapons of Vampire and the most secretive until recently. It is described in [?].

On the other hand, Prover9 itself does not have the capability of preparing portfolios or running several schedules in a row. Its flow control is rather limited: it tries to adjust its behaviour to the input problem[28], features an experimental mode to limit its search space (see 1.2.2) and, in addition, lets the user implement some rudimentary strategies through actions (explained in Section 1.2.8). Also, a separate program included in the LADR package, *FOF-Prover9*, includes a preprocessing step that attempts to reduce the problem to independent subproblems[29]; if the problem reduction succeeds, each subproblem is automatically given to the ordinary search procedure (possibly reducing the overall time significantly). The output then informs about the proof status of each subproblem (see Figure 1.1.3).

Lastly, an outside system can be coupled to Prover9 in order to provide an automated schedule configuration. This is what *Grackle*[30] does, and may do on the fly (in what is called *automated strategy invention*). In [?] it is shown that Prover9 with a *Grackle*-determined

[26]Deprecated from now on in favor of `--mode portfolio --schedule casc`.

[27]There are proofs than can be found when the time limit is set to less than 5% of the time Vampire needs to find a proof without the time limit [?]. This is due to at least two factors: the fact that the right strategy will be tried faster since less time is spent with each strategy; and the behaviour of Vampire's default saturation algorithm, the limited resource strategy algorithm (see Section 1.2.2).

[28]More concretely, it adjusts its inference rules and strategy according to syntactic properties of the input clauses, such as the presence of equality and non-Horn clauses.

[29]By a miniscope (antiprenexing) transformation.

[30]`https://github.com/ai4reason/grackle`.

```
============================================ FOF-Prover9 ===========================================
[...]
Attempting problem reduction; original problem has <nnf_size,cnf_max> = <210,16384>.

Problem reduction (0.01 sec) gives 32 independent subproblems:
( <38,8> <38,8> <37,10> <39,10> <40,8> <40,8> <39,10> <41,10> <39,10> <39,10> <38,12>  <40,12> <39,10>
<39,10> <38,12> <40,12>  <38,8> <40,8> <39,10> <39,10> <38,8> <40,8>  <39,10> <39,10> <37,10> <39,10>
<38,12> <38,12> <39,10> <41,10> <40,12> <40,12> ).
[...]
============================================ end of multisearch ====================================
All 32 subproblems have been proved, so we are done.
```

FIGURE 1. Relevant details of the output from a successful *FOF-Prover9* run.

portfolio outperformed Vampire[31] in casc mode on an AIM benchmark[32]: concretely, out of 200 problems, Vampire 4.5.1 proved 25%, Vampire 4.8 proved 40%, and Prover9+*Grackle* proved 46% (including all those proved by version 4.5.1). An strategy invention procedure based on previous proof history has been tried in Vampire with promising results ([?]), but it does not appear to be implemented in Vampire as of version 5.0.

1.1.4. *Input.* Prover9-Mace4 (LADR[33]) syntax is close to standard writing in usual algebra, using infix notation for binary operations and relations, with familiar symbols reserved for them (`+,*,\,/,@,<,>`), hence being accessible to the mathematician's eye. The symbols reserved for first-order logic are `$T,$F` for truth values, `=,!=` for equalities, `-,|,&,->,<-,<->` for connectives (negation, disjunction, conjuction, implication, reverse implication, equivalence), and `all,exists` for quantifiers. Each formula must end in a dot (`.`). Axioms are enclosed within the `formulas(assumptions).  end_of_list.` environment, while goal(s)[34] are enclosed within the `formulas(goals).  end_of_list.` environment. Prover9-Mace4 formulas can have free variables, which are assumed to be universally quantified at the outermost level of the whole formula. This property is susceptible to lead to some mistakes:

(1) Since formulas can have free variables, constants and variables must be distinguished by their names. The default rule is that variables range from $u$ to $z$ while the rest of lowercase letters are constants. Therefore writing universal axioms with other lowercase letters (e.g. $a * b = b * a$) would lead to a mistake (in the example, a property imposed over two fixed constants, instead of over all pairs of elements). If more variables are needed, PROLOG style can be activated with `set(prolog_style_variables).`[35]. In PROLOG style all uppercase letters are variables and all lowercase ones are constants.

[31]And also the E prover.

[32]AIM stands for the Abelian Inner Map conjecture ([?]), one of the top open conjectures in quasigroup theory. Working with Prover9, a large number of AIM-related proofs were obtained (`https://www.cs.unm.edu/~veroff/AIM_REDONE/`), from which a database with 3468 equations was produced as an ATP benchmark [?]. The database is available at `https://github.com/ai4reason/aimleap/tree/master/aimleapprobs`.

[33]LADR stands for Library for Automated Deduction Research.

[34]If there are multiple goals, Prover9 tries to find a proof for each one, while Mace4 looks for structures falsifying all the goals simultaneously. If there is no goal, Prover9 will try to find a contradiction among the axioms, while Mace4 will try to produce a model of them.

[35]This flag affects all formulas regardless of its location within the input.

(2) If universal quantifiers are not explicitly included in a formula, quantifiers will be added at the outermost level regardless of its parenthesization. For example, suppose we want to formulate that the right multiplication map is injective if and only if it is surjective. If we had both map properties already written separately and without quantifiers, we may think of copy-pasting and joining them as

```
(y*x = z*x -> y = z) <-> (exists w w*u = v).
```

But this actually means

```
all x all y all z all u all v (y*x = z*x -> y = z) <-> exists w w*u = v.
```

instead of the desired[36]

```
all x all y all z (y*x = z*x -> y = z) <-> all u all v exists w w*u = v.
```

On the other side, Vampire uses[37] the TPTP language[38], the current standard for ATPs[39]. It follows many PROLOG conventions, with different sublanguages for different logics (first order, higher order, typed, etc.). Variables start with an uppercase letter, user-defined operations and relations start with a lowercase letter and are prefixed. The symbols reserved for first-order logic are `$false,$true` for truth values, `~, |, &, =>, <=, <=>, <~>` for connectives (negation, disjunction, conjunction, implication, reverse implication, equivalence, XOR), and `!,?` for quantifiers (for all, exists). Quantified formulas are written in the form `Quantifier [variables] : formula`. Axioms (known results) are written in the form `fof(name,axiom,formula).`, the conjecture (goal[40]) as `fof(name,conjecture,formula).`. Every variable in a formula must be bound by a preceding quantification with adequate scope; negation has higher precedence than quantification, which has higher precedence than the binary connectives. This property is susceptible to lead to a mistake analogue to 2) above:

(3) Wrongly placed parenthesis can disrupt the quantification scope. Suppose for example that we want to formulate that the right multiplication map is injective if and only if it is surjective. If we had both map properties already written separately with their quantifiers, we may think of copy-pasting and joining them[41] as:

```
(![U,V,W]: m(V,U) = m(W,U) => V = W) <=> (![Y,Z]: ?[X]: m(X,Y) = Z)
```

But since quantification has higher precedence than binary connectives, and no opening parenthesis actually restricts the first quantifier scope, that formula actually means

```
![U,V,W]: m(V,U) = m(W,U) => V = W <=> ![Y,Z]: ?[X]: m(X,Y) = Z
```

with the first "for all" quantifying the whole formula. So we need to place the parentheses creating the right scope for the first quantifier:

```
![U,V,W]: (m(V,U) = m(W,U) => V = W) <=> ![Y,Z]: ?[X]: m(X,Y)= Z
```

[36]These formulas are not equivalent, since the model [[0,0],[0,0]] satisfies the desired property but not the unquantified one.

[37]It can also use the SMT-LIB language, `https://smt-lib.org/language.shtml`, which we won't describe here.

[38]https://tptp.org/UserDocs/TPTPLanguage/TPTPLanguage.shtml

[39]For Prover9-Mace4, the LADR package includes the programs TPTP_to_LADR and LADR_to_TPTP to translate formulas between both languages, with some restrictions.

[40]Vampire allows at most one goal in its input file. If there is no goal, it will try to find a contradiction among the axioms.

[41]after a `fof(rmis,axiom,` and closing with `).`.

We have in fact committed mistakes 2) and 3) over the course of the ETP, giving rise to false proofs of E677 $\models_{\mathrm{fin}}$ E255, our only open implication.

Frequently, it is useful not to run an ATP alone, but to run it in a larger environment permitting multiple calls with different input files, strategies, etc. so that we can provide (automated) semiguidance or (user-guided) interactive guidance. Integrating the ATP into a computer algebra system further allows to leverage the latter's mathematical capabilities, such as preparing input files with operations from sophisticated algebraic structures. For example, in the ETP we have integrated Prover9-Mace4 with Python and SAGE, and (among several other uses) we have invoked GAP's small groups library through SAGE in order to search, with Mace4, for translation-invariant countermodels (see Section **??**) with specific groups.

An important factor to take into consideration when generating an input file for proving a conjecture is, for both Vampire and Prover9, that owing to the way the saturation algorithm operates (see Section 1.2.1), the original order in which formulas are presented is actually important: it is perfectly possible to have a collection of axioms which produces a proof in some order but not in another. Vampire includes the option `--normalize` to prevent this effect, but its operation is not described. For Prover9, one can use a larger environment to automate the permutations of the input file and make several tries with some time restrictions. In contrast, the order of the formulas does not affect Mace4's response.

1.1.5. *Output.* Prover9 may stop either because some imposed limit is hit or because it finds a proof[42]. Vampire can also stop if it detects the existence of a countermodel to the goal[43]; in both cases, Vampire deems the result a success. After a proof in equational logic is found, both Vampire and Prover9 output a proof scheme[44] with comments[45] that a human can read (with some effort)[46], once she understands the inference rules used in a superposition/saturation ATP (see Section 1.2.1). Mace4 may stop either because some imposed limit is hit, the requested sizes have been exhausted without finding any models, or the required number of models has been found.

For interpreting the output and helping a guidance system decide its next step, the best option, in our opinion, is not to search the output file for some specific clue, but the use of

[42]Or proofs, in case more than one goal has been included.

[43]This can happen in two ways: Vampire's schedule may invoke its finite model builder to seek for an explicit countermodel, or some strategy may find a saturated set of clauses not including the empty one, meaning that a refutation cannot be achieved, so that the negation of the goal together with the axioms is satisfiable.

[44]Due to the way they operate, it is usual that ATPs generate many more formulas in their output than strictly needed for the proof a posteriori.

[45]The proof is a directed acyclic graph with False at the root, where each node represents an intermediate clause in the proof, which is labelled by its parents and the inference rule used to derive it from them.

[46]In addition, the LADR package includes a program, *prooftrans*, that allows to expand the proof and also to export it in IVy (the proof checker) or XML formats, while in Vampire, the command `-p tptp` translates the proof to TPTP format

exit values[47]. Since they are stored in ephemeral sources, we include here Prover9's (Table 1.1.5) and Mace4's exit codes (Table 1.1.5).

| Exit code | Trigger | Reason |
|---|---|---|
| 0 | MAX_PROOFS | The specified number of proofs (`max_proofs`) was found. |
| 1 | FATAL | A fatal error occurred (syntax error or Prover9's bug). |
| 2 | SOS_EMPTY | Sos list exhausted. |
| 3 | MAX_MEGS | The memory limit (`max_megs`) was exceeded. |
| 4 | MAX_SECONDS | The time limit (`max_seconds`) was exceeded. |
| 5 | MAX_GIVEN | The `max_given` parameter was exceeded. |
| 6 | MAX_KEPT | The `max_kept` parameter was exceeded. |
| 7 | ACTION | A Prover9 action terminated the search. |
| 101 | SIGINT | Prover9 received an interrupt signal. |
| 102 | SIGSEGV | Prover9 crashed, most probably due to a bug. |

FIGURE 2. Prover9's exit codes together with their reason for termination.

| Exit code | Trigger | Reason |
|---|---|---|
| 0 | MAX_MODELS | The specified number of models (`max_models`) was found. |
| 1 | FATAL | A fatal error occurred (syntax error, (`max_megs`) exceeded, or Mace4's bug). |
| 2 | EXHAUSTED | There are no models within the given constraints. |
| 3 | SOME | Some models were found, but terminated without the number specified. |
| 4 | SOME_TIME | Terminated by time limit with some models, but not the number specified. |
| 5 | MAX_SECONDS | The time limit (`max_seconds`) was exceeded before any models were found. |

FIGURE 3. Mace4's exit codes together with their reason for termination.

Since they are not officially documented, we include here Vampire's return values[48] (Table 1.1.5).

| Exit code and Trigger | Reason |
|---|---|
| 0 (success) | Either found refutation or established satisfiability. |
| VAMP_RESULT_STATUS_INTERRUPTED (SIGINT, SIGHUP) | Vampire was interrupted. |
| VAMP_RESULT_STATUS_OTHER_SIGNAL | Other signal. |
| VAMP_RESULT_STATUS_UNHANDLED_EXCEPTION (error) | Unhandled exception or user error. |
| 134 or some other nonzero value (time limit) | Vampire was terminated by the timer. |

FIGURE 4. Vampire's return values together with their reason for termination.

In the case of Vampire, it may be more informative to look inside the output file. Vampire uses the SZS ontology, so one should search for the SZS status (the word next to `SZS status`), which in most cases will be one of:

• `Theorem`: The input contains a conjecture, after negation of which the input is unsatisfiable. Hence the conjecture is derived from the axioms as a theorem.
• `CounterSatisfiable`: The input contains a conjecture, after negation of which the input is satisfiable. Hence there is a countermodel satisfying the axioms but not the conjecture.

[47]As an illustration, during a stage of the ETP, success in Prover9 was initially identified by detecting the output message "Exiting with 1 proof", but this approach proved unreliable when Prover9 was occasionally run with more than one goal and reported "Exiting with $N$ proofs" for some $N > 1$.

[48]Extracted from `https://github.com/vprover/vampire/blob/master/vampire.cpp#L65`.

- `ContradictoryAxioms`: The input contains a conjecture, but the axioms alone are unsatisfiable.
- `Satisfiable`: The input does not contain a conjecture and is satisfiable.
- `Unsatisfiable`: The input does not contain a conjecture and is unsatisfiable.
- `FiniteModel`: A finite model was found.
- `Timeout`: Proof or model not found in the allowed time.

We need to consider that, typically, Vampire will run not a strategy, but a schedule, and thus the successful strategy (if any) will be the last one in the output, preceded by several unsuccessful ones. The SZS status will appear at the start of the last strategy[49].

At the beginning of each strategy run, Vampire's output includes a line with a code determining the strategy (an example is shown in Figure 1.1.5). Said strategy can be later rerun by using the command `--decode <strategy code>`.

```
lrs+4_2:3_slsqr=1,8:to=lpo:sil=2000:tgt=full:plsq=on:fde=none:gsssp=full:
plsqr=4,31:sp=const_min:urr=on:fd=preordered:foolp=on:gs=on:flr=on:slsq=on:
random_seed=330991339:gsem=randomized:s2pl=no:i=536:nm=0:fsd=on_3
```

Figure 5. Code sample from a Vampire's strategy.

## 1.2. **Operational principles.**

### 1.2.1. *Saturation algorithm.*

Effective modification of the strategy of an ATP requires a clear understanding of its basic underlying mechanisms. Herein we explain the fundamentals of a saturation algorithm (which both Vampire and Prover9 follow in general terms, with some modifications[50]).

(1) The first step is **preprocessing** the input formulas, which can be subdivided into simplification and clausification.

- **Simplification** is the process of removing (e.g. redundant formulas), simplifying, subdividing, or rewriting some formulas, or selecting or highlighting some others (for example, assigning them a high weight).
- **Clausification**: Saturation algorithms work with clauses, hence their input formulas must be put into clause form. To achieve this the usual procedure consists of sequentially doing an NNF[51] conversion, Skolemization[52], moving universal quantifiers to the top, and a CNF[53] conversion.

[49]Along the output we may also see `Refutation found` before the proof and `Termination reason: Refutation` after the proof; also `Termination reason: Satisfiable` in the relevant occasions.

[50]Refer to [?] for more detailed information on Vampire's saturation algorithm.

[51]A formula is in negative normal form if it only uses negation, conjunction, disjunction and quantification, and all negation operations are applied to atomic formulas.

[52]The process of replacing existentially quantified variables with new constants or functions symbols.

[53]A formula is in conjuctive normal form if it is a conjunction of clauses.

(2) Now the **saturation loop** starts. At any iteration, the algorithm uses three main objects: the given clause, the usable list, and the set-of-support (sos) list[54].

- The **given clause** is the clause being processed. All inferences in this step must have the given clause as a parent.
- The **usable list** is formed of those clauses available to make inferences with the given clause, as the other parents.
- The **sos list** is formed of those clauses waiting to be selected as the given clause. It is initialized to the (simplified) input clauses.

(3) The **given clause is selected** from the sos list by some previously specified method and added to the usable list.

- Priority queues (age priority, weight priority): light clauses easier to process and more likely to contribute to a derivation of the empty clause, old clauses maintain the unsatisfiability of the set. Age-weight ratio
- True and false.

(4) Some **generating inference rule**[55] with the given clause and other usable clauses as parents **is selected**, and its conclusion is derived. At each iteration, for a given clause, all generating rules are eventually performed with all the usable list[56].

(5) **Forward simplification**: The conclusion is simplified by some clauses, using simplifying inferences.

- In the **Otter algorithm**[57], the conclusion can be simplified by both the usable and the sos list.
- In the **Discount algorithm**[58], only clauses in the usable list can simplify the conclusion[59].

(6) A **retention test** is passed on the (simplified) conclusion. If not retained, go back and generate a new conclusion. Else, move forward into the loop. The retention test is comprised of forward deletion and limits checking.

- In **forward deletion**, a collection of deletion inference rules checks whether the conclusion is redundant with the usable list.
- In **limits checking**, the conclusion is discarded if any of its measures exceeds the predefined limits[60].

(7) **Backward deletion** and **backward simplification**: Simplify or delete usable clauses by the conclusion and move the simplified clauses to the sos list.

(8) Move the **conclusion to the usable list**. Go back and select next generating rule, if any. Else, move the **given clause to the usable list** and start a new loop iteration (with a new given clause).

[54]We have stated the terminology in the language of Prover9. In that of Vampire, the given clause is called the new clause, the usable list is called the set of kept clauses, and the sos list is called the set of unprocessed clauses.

[55]Generating inferences are those that add a new clause without discarding any of its parents, in contraposition to simplifying inferences, in which some parent clauses become redundant after the addition of the conclusion.

[56]The guiding principle is to apply simplifying inferences eagerly and generating ones lazily.

[57]After the Otter ATP, Prover9's direct predecessor.

[58]After the Discount ATP. Used for example by the E prover.

[59]In this context, more generally the list of clauses that actively participate in simplification may be called the active list, the rest of clauses conforming the passive list.

[60]This process compromises the algorithm's refutational completeness.

(9) The **loop ends** whenever the empty clause is derived (refutation), a saturated set of clauses containing the axioms and the negation of the conjecture is built which does not contain the empty clause (satisfiability by saturation), some global limit is hit (time, number of processed given clauses, etc.) or the sos list gets empty (unsuccessful attempt).

The sos list size roughly determines the size of the search space, and the given clause selection strategy greatly affects the path followed through the search space. Prover9 `sos_limit`

Both Prover9 and Vampire produce statistics about the number of active, passive, and discarded clauses; simplifying, deletion, and generating inferences, etc. For Vampire to output them, one has to add the command `-stats full`. This information may prove useful in changing the strategy in the search for a proof, or for a simpler proof.

1.2.2. *Algorithm alternatives.* Vampire offers, through the command `-sa <algorithm>`, five core algorithms: the three saturation-based algorithms Discount (`discount`), Otter (`otter`), and limited resources strategy (LRS, `lrs`, described below); *Z3* after preprocessing the problem[61] (`z3`); and the finite model builder (`fmb`). The default algorithm is LRS[62].

Vampire's LRS algorithm always run with a time limit. It is Otter based, but it discards those clauses which will be unreachable given the time limit, and so it is estimated that will never be processed. It uses the age-weight ratio and implements a dynamical weight limit[63].

The core algorithm and the parameters chosen for a Vampire's strategy can be read from its code. We use the code in Figure 1.1.5 as an example. The first three letters describe the saturation algorithm (`lrs` in the example), next come literal selection function (`2`), and age-weight ratio (`2:3`), then the various settings like term order `to` (`lpo`), function definition elimination `fde` (`none`)... The last number, after the last low dash, is the strategy time limit in deciseconds (`3`).

Prover9, strictly speaking, uses only the Otter algorithm. It features an experimental mode (also called LRS) that automatically adjusts the sos list size via `assign(lrs_ticks,n)` (with $n \geq 0$ to apply the method), `assign(lrs_interval,n)`, and `assign(min_sos_limit,n)`.

1.2.3. *Selection schemes.* Prover9's default selection scheme works in a cycle of 9 steps: Select the oldest available clause, select the four lightest available 'false' clauses, select the four lightest available 'true' clauses

Prover9's semantic guidance

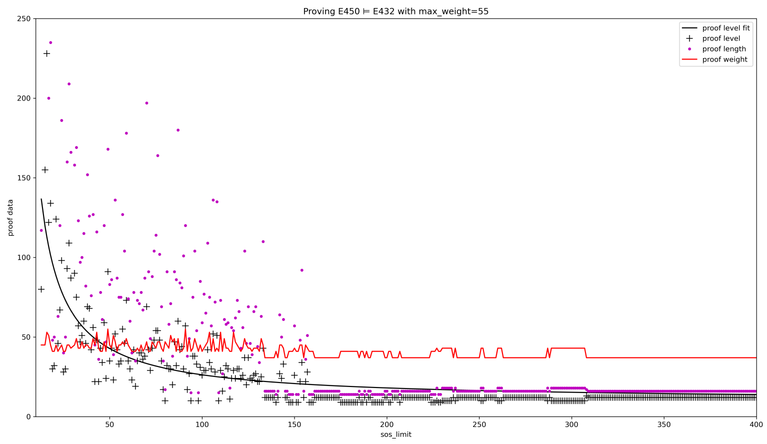
[61]*Z3* is an SMT solver from Microsoft, `https://github.com/Z3Prover`, whose core component is a SAT solver. Hence it needs the problem to be ground.

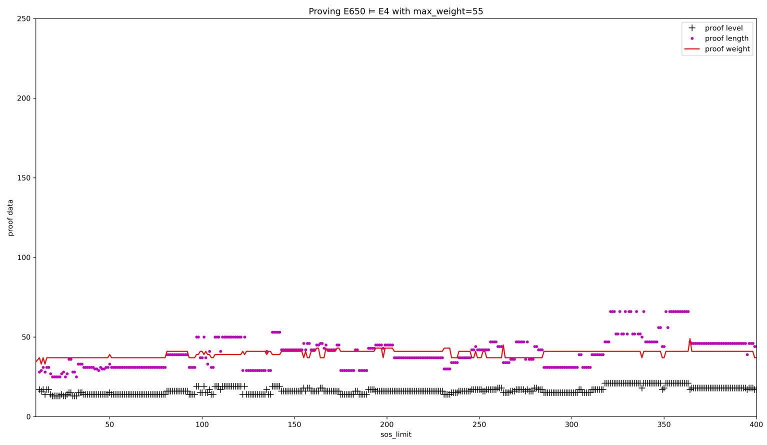
[62]A comparison between the Discount, Otter, and LRS algorithms that justifies LRS as the best default option can be found in [**?**].

[63]The LRS algorithm can miss a proof due to miscalculation of unreachability, specially in backward simplifications, where a simplification of a discarded clause would result in a proof.

### 1.2.4. *The weight limit strategy.* Prover9 `max_weight`

LRS vs weight limit: according to [**?**, Section 8], although a well-fitted weight limit can give optimal solving time, LRS can solve more problems than weight-limit strategies and even be more efficient in some cases[64]





### 1.2.5. *Term orderings.* Term orderings: Prover9 KBO, LPO, RPO. Vampire LPO, KBO and variants for arithmetic problems. Term orderings need symbol precedence, which is configurable in both programs. In addition, Prover9 lets the user determine the KBO weights.

### 1.2.6. *Inference rules.* ATPs allow to change the strategy by modifying some inference rules.

Vampire implements around 80 rules among preprocessing, generating and simplifying inferences, and deletion rules. The `Inferences` section of its options help includes around 40 parameters.

[64]Unlike newest versions, Vampire 2.6 allowed to change the weight limit with parameter `-weight_limit`.

Inferences miniglossary: ground term, demodulator, splitting, flattening, unification, most general unifier (mgu), resolution, subsumption, subsumption resolution (simplifying inference), paramodulation (equality substitution), superposition deletion rules (subsumption, tautology detection, forward demodulation...) hyper-resolution, UR-resolution, binary resolution, unit deletion simplification, resolution

Prover9 `eq_defs`, fold. Vampire's function definition elimination `--function_definition_elimination` (`-fde`)

1.2.7. *Hints.* prooftrans automatically extracts hints

[Veroff]: In contrast to weighting, the hints strategy focuses directly on the identification of key clauses rather than on the general calculation of weights. Any generated clause that subsumes or is subsumed by a user-supplied hint clause is identified as being "interesting". The weight of such a clause is adjusted (either positively or negatively) according to user preferences; the cases of subsuming a hint, being subsumed by a hint, or both are controlled separately. The case in which a generated clause subsumes a hint clause is probably the most interesting and potentially valuable. Such generated clauses can be viewed as key milestones on the way to a proof. (If a fact is deemed significant, then for many applications of automated reasoning, anything that subsumes the fact would be just as significant.) The case in which a clause is subsumed by a hint looks more like weighting (i.e., identifying instanc.es of user-defined patterns) and may be less interesting, although there are problems for which finding instances of key clauses is significant. The third case, identifying clauses that both subsume and are subsumed by a hint, is particularly useful for various types of proof checking. In summary, the hints strategy enhances the weighting strategy, first, by focusing on entire facts (i.e., clauses) rather than terms and subterms, and second, by using subsumption as a criterion for determining the value of a generated clause. Being based on subsumption, the hints strategy adds a semantic, or logical, component to the evaluation of a clause.

See Veroff's paper for examples, cite Kinyon's.

1.2.8. *Actions.*

1.3. **Question answering.** Question answering: Provide question answering when there are existential quantifiers at the outermost level of a conjecture. In Prover9:

In Vampire: `--question\_answering (-qa)` in Vampire, `-qaat` allows to avoid unwanted answers. In the TPTP syntax, questions are written like conjectures, but using the question role instead of the conjecture role. The outermost existentially quantified variables are the ones that the user wants values for, i.e., their sets of instatiations are the answer.

```
fof(xyz,question, ? [X,Y,Z] : p(X,Y,Z)).
```

If the user wants values for only `X` and `Z`:

`fof(xyz,question, ? [X,Z] : ? [Y] : p(X,Y,Z)).`

The SZS status is

## 1.4. **Prover9's GUI particularities and problems.**

- Previous version, faster, more complex
- Allows to run Prover9 and Mace4 simultaneously on the same problem
- Allows to see and change at a glance all options organized by sections. For Prover9: term ordering, limits, search preparation, goals/denials, select given, inference rules, rewriting, weighting, process inferred, input/output, hints, random seed
- Some bugs were fixed in version 2009 from version 2007, particularly when using Mace4 with `set(skolems\_last)` some nonisomorphic models can be missed. See details in the LADR changelog (the GUI version is 0.5 from December 2007). `https://www.cs.unm.edu/~mccune/prover9/download/Changelog.txt`
- Isofilter bugs
- Memory problem
- Additional input tab: weighting rules, KBO weights, function_order and relation_order, actions, interpretations for semantic guidance, and lists of hints.
- Additional input problems: `assign(max\_models,-1)`, `assign(max\_seconds, -1)`
- Needs library MSVCP71.DLL to be put in the Prover9-Mace4 (or perhaps in the Windows system directory) (`https://www.cs.unm.edu/~mccune/mace4/gui/MSVCP71.DLL`).

## 1.5. **Finite model building.**

1.5.1. *Operational principles.* Vampire's uses the MACE-style relational method: – reduction to a propositional problem (without equality) – decision by DPLL (Davis-Putnam-Loveland-Logeman)

SAT solvers included in Vampire: minisat, cadical (not in v4.9), Z3 (default is minisat, can be changed with `-sas`). Used in AVATAR for splitting (with the discount, lrs or otter saturation algorithms) and in finite model finding (with the fmb saturation algorithm)

Vampire's finite model building: For v4.9: `--mode casc_sat problem.p`

For v.5.0: `--mode casc --intent sat problem.p` or

`--mode portfolio --schedule casc_sat problem.p`

For the fmb: `-sa fmb -fmbss <starting size> problem.p`

Mace4 uses the SEM-style equational method:

– reduction to a ground problem with equality – decision by case splitting, and rewriting – and possibly negative inference or some flattening

Procedure for selecting best parameters in Mace4: `selection_order`, `selection_measure`, `skolems_last`. Very sensitive to formulas (example $L^n(x) = x$ change with $n$)

Time can be slower for exhausting but faster for finding one model. Example where model found soon but exhaustion is long

Example where set of parameters is better until a point

Mace4 tool support: isofilter and dprofiles, clausefilter, clausetester, interpfilter

1.5.2. *Hints and tips.* When looking for finite countermodels, we can assume finite setting. See case example 1518 anti 47. Originally we had a model of size 232

In the finite setting, adding surjectivity to Mace4 is usually slower

Redundant formulas good for proofs, bad for models

Timing of different ways of writing the same: injectivity, etc. Division operation instead of injectivity seems a bit slower in Vampire.

No use writing particularizations of a formula neither in Prover9 nor in Mace4

1.5.3. *Isofilter.*

(1) without problems up to size 10.
(2) Time to check 135 models of size 15 with 2 operations: 73min. 135 models of size 15 with 3 operations: 73min, making 131 isomorphism checks between pairs of models, involving 100 billion permutations ($10^{11}$) Time to check 1440 models of size 18 with 2 operations: 2647min (1.8 days), making 3006 isomorphism checks between pairs of models, involving more than 3 trillion permutations ($3.5 \cdot 10^{12}$).
(3) discrim, isofilter2

    According to **??**:

    It calculates one invariant, frequency of occurrence of domain element, that is, the number of times a domain element appears in the operation tables. It uses this invariant to help separate non-isomorphic models and to help guide the construction of isomorphic functions between potentially isomorphic models. Needless to say, the discriminating power of one single invariant is limited. Indeed, it fails miserably when the operation table is a Latin square, which is the case for quasigroups. To alleviate this issue, Mace4 has another program, isofilter2, which does not try to construct isomorphic functions between models, but to convert models to their canonical forms based on the same algorithm [23] used by SEMK. Isofilter2 works very well with quasigroups, but the overhead in computing the canonical forms of the models is so high that it becomes slower than isofilter for many algebraic structures such as semigroups.

    List of discriminators used for isofiltering the models of E1518 $\not\models$ E47:
    `x*x=x. (x*y)*z = x*(y*z). x*y=y. x*y=y*x.`

| ALGORITHM | Time | Permutations | Checks |
|---|---|---|---|
| **Isofilter** | 10 days | $3.1 \cdot 10^{13}$ | 27 |
| **Isofilter0** | 10 days | $3.1 \cdot 10^{13}$ | 27 |
| **Isofilter0 with discriminators** | | | |
| **Isofilter2** | 5.14 days | $1.3 \cdot 10^{13}$ | 0 |
| **Isofilter2 with discriminators** | 1 day | $2.6 \cdot 10^{12}$ | 0 |
| **Isofilter with discriminators** | 0.02s | $1.4 \cdot 10^{5}$ | 4 |

FIGURE 6. Statistics for the different isofiltering algorithms, applied to 10 models of size 15 of E1518 $\not\models$ E47.

(4) Check and output: If one of the operations is a Skolem function from an existentially quantified variable, one might wish to omit it for checking and for output

## 1.6. **Vampire and Prover9 strengths and drawback (make a table).**

**Vampire strengths:** textbfFor both: ready to use, higher-order logic, multi-sort and reasoning with theories, specify number of used cores, now a Git project. textbfFor proving: smart use of SAT solvers for subtasks, all-round, saturation, Otter simulation, also looks for satisfiability via small models. textbfFor models: finite model unsatisfiability.

**Vampire drawbacks:** lack of manual, twinkering requires higher learning curve, finite model builder only uses SAT, no search for all models, return values not completely informative, Spider is opaque, possibly overfitted to some specific problems

**Prover9-MACE4 strengths:** textbfFor both: Easy to use, good manual, GUI, infix notation with translation to TPTP, parameter tweaking, multiple goals, support tools. textbfFor proving: hints, semantic guidance, production mode. textbfFor models: model tables, isofilter, Mace4 dedicated and not SAT.

**Prover9-MACE4 drawbacks:** discontinued, no portfolio mode, very limited higher-order capabilities.

(1) Bruno: all positive implications with Prover9

**To add to other sections:**

(1) Thanks: to Vampire's developers and to Michael Kinyon
(2) Add to finite section: periodicity
(3) Change HN section: most difficult proof
(4) References
(5) Add files to GitHub, including corrected Makefile due to newer gcc compiler version