LEO-II version 1.5

Christoph Benzmüller[1] and Nik Sultana[2]

[1] Freie Universität Berlin, Germany
[2] Cambridge University, UK

Abstract

Leo-II cooperates with other theorem-provers to prove theorems in classical higher-order logic. It returns hybrid proofs, which contain inferences made by Leo-II as well as the backend provers with which it cooperates. This article describes recent improvements made to Leo-II.

1 Introduction

Leo-II [9] is an automatic theorem-prover for classical higher-order logic, more precisely for Church's type theory with Choice, under Henkin semantics [1, 2]. Its cooperation with backend provers is one of its distinguishing characteristics. These provers are regularly invoked by Leo-II for help with finding a refutation. In this article we outline the current system and describe recent improvements. Further details on Leo-II's hybrid proofs are reported in [16].

2 System overview

Leo-II's calculus [16] is a higher-order adaptation of RUE (Resolution by Unification and Equality) [4]. RUE is an approach for extending a resolution calculus to interpret equality, and which allows equality literals to be processed by both resolution and unification. Furthermore, Leo-II's calculus relies on a 'Boolean aware' (or, more generally, 'theory aware') extensional preunification engine (extensional preunification is discussed in [5]). In recent versions, Leo-II's unification algorithm also interprets logical constants — for example, the algorithm in version 1.5 treats disjunction as a commutative function.

Leo-II accepts problems encoded in the CNF (clausal first-order form) and FOF (first-order form) languages from the TPTP [18], but its principal input language is THF0, core typed higher-order form [19].

The logical organisation of the prover is illustrated in Figure 1, and corresponds to the modular organisation of the code. It is structured into four layers, as the figure shows:

Operating mode. The prover can be operated in two ways: (i) Leo-II can be used as a proof assistant when run in *interactive mode*. It provides a command interface through which the user can inspect and manipulate the prover's state, making calls to the calculus' rules as needed. This mode is very valuable for exploring logical problems and for debugging the prover's automatic mode. (ii) The prover is usually run in *automatic mode*: this comprises a set of strategy schedules, and a main loop which drives applications of the calculus' rules.

Prover interface. Both modes use a common infrastructure: they parse a problem and load it into the prover's state, then further manipulate the state by executing *commands*. A command might involve carrying out an inference, inspecting the state, switching flags, calling external provers, etc. Each command makes calls to lower levels of the prover.

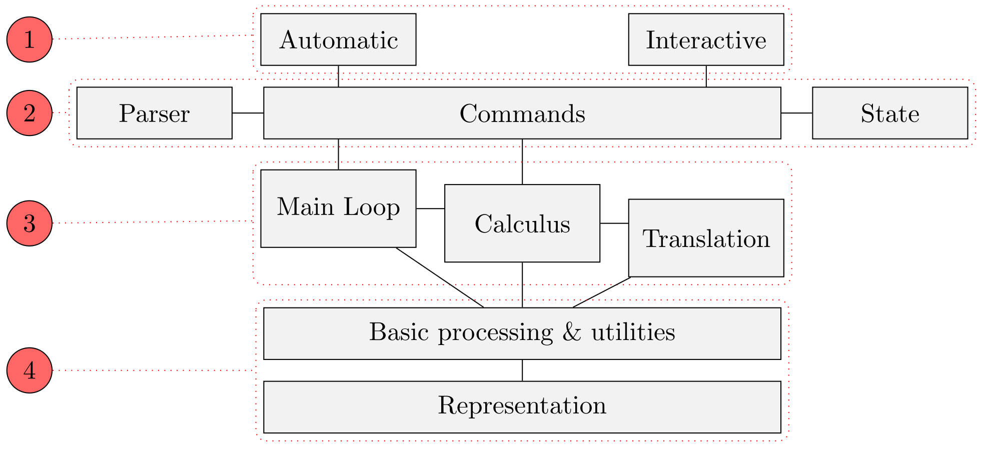

Figure 1: Leo-II's architecture

Logic. The main component in this level consists of the calculus: a collection of functions which accept and return clauses. This level also contains Leo-II's main loop, and an interface to external ATPs (which also translates problems to other formats).

Basis. The lowest level of Leo-II defines the representation of terms and types, and associated operations (e.g. substitution, unification, matching, etc).

3 Improvements

The TPTP problem set [18] is the canonical benchmark by which theorem provers are evaluated. The improvements described in this section are often accompanied by TPTP problem names whose solution is affected by the improvement. These problems consist of THF problems (more precisely, THF0 problems) drawn from TPTP 5.4.0. We have used E version 1.6 as the backend ATP. Our tests were run on a 2GHz AMD Opteron with 4GB RAM, and given 60-second timeout. Leo-II was compiled with OCaml 3.11.2.

3.1 ATP interface

Leo-II cooperates with other provers in order to maximise its potential. Recall that Leo-II proves a theorem by refuting a set of clauses. It gradually accumulates a set of clauses, some of which are first-order; a refutation in these first-order clauses will refute the overall problem. Instead of attempting to refute first-order clauses itself, Leo-II invokes external first-order ATPs to do this, since they are likely to do a far better job than Leo-II on such problems. This leaves Leo-II to focus on higher-order reasoning.

We improved Leo-II's translation to FOL in recognition of this benefit. Version 1.5 includes a better translation into FOF, and added an experimental translation into TFF [20] (a TPTP syntax for sorted first-order logic). We also improved the system interface with backend ATPs, and experimented with additional backends.

3.1.1 Translation into FOL.

It benefits Leo-II to use a translation into FOL which returns the strongest set of clauses, as long as that translation is sound.

Alongside the old translations which were previously implemented in Leo-II, version 1.5 features a new translation module which was written from scratch. This module contains an intermediate language to which problems are first translated, before being transformed further and printed into a specific target syntax. HOL-to-FOL translations consist of a pipeline of functions which bring HOL formulas into this intermediate language, applying analyses and transformations along the way.

Leo-II's old and new FOF encodings can be used through the command-line arguments `--translation fully-typed` and `--translation fof_full` respectively. In version 1.5, the translation `fof_full` is now set as default. The old translation had some undesirable qualities which harmed the performance of the FOL ATPs with which Leo-II cooperated:

1. For some examples, the old translation did omit certain necessary information in its output to the ATP. This information is of two kinds: the first relates to proxy terms, and the second relates to λ-terms.

   Here is a trivial example: when Leo-II is asked to prove

   ```
   thf(goal, conjecture, ((=) = (=))).
   ```

   and use the old translation, it would send a single clause to the ATP (after transforming the negated conjecture in its input processing into `~($true)`):

   ```
   fof(7,axiom,((~ leoLit(leoTi(true,o))))).
   ```

   In both the old and new translations used by Leo-II, `leoTi` is used to assign types to terms — here it is saying that the term `true` is of type `o` (i.e., propositions), where '`o`' itself is a term in the language. That is, the translation encodes types as first-order terms. The constant `leoTi` is used to lift propositional terms (i.e., those typed '`o`') into formulas. Unfortunately, Leo-II did not include an axiom to give semantics to `true`; such as

   ```
   fof(true, axiom, leoLit(leoTi(true,o))).
   ```

   Had such an axiom been included, the FOL ATP would have been able to find a refutation.

   Given the same THF problem, the new translation sends the following output to the ATP:

   ```
   fof(7, axiom, ~($true)).
   ```

   That is, it notices that instead of using `leoLit` to encode "true", it can simply use the FOF constant with that denotation. When it becomes necessary to use a proxy term such as `true`, then it includes an axiom giving its semantics. For instance, while attempting to prove

   ```
   thf(conj_0,conjecture,(
       ? [F: $o > $o] :
       ! [P: $o > $o,Q: $o] :
         ( ~ ( P @ ( => @ ( F @ $true ) @ Q ) )
         | ~ ( F @ ( => @ Q @ ( F @ $false ) ) )
         | ( F @ Q ) ) )).
   ```

Leo-II's output to the ATP will include the axiom

```
fof(prox_true1, plain, ($true <=> leoLit(leoTi(true, o)))).
```

Note that Leo-II's current behaviour is not perfect either: Leo-II should be able to spot trivial refutations (as the first one above), and avoid invoking the FOL ATP and instead use its own refutation mechanism only.

The second kind of information relates to the reduction of λ-terms into first-order form: the previous translation simply created fresh constants for λ-terms, and did not characterise these constants further. For example, while trying to prove the THF problem mentioned earlier, the output of the previous translation includes axioms such as

```
fof(44,axiom,(
    ~ leoLit(leoTi(leoAt(leoTi(sK2_SY3,leoFt(leoFt(o,o),o)),
                         leoTi(abstrSX0SX0,leoFt(o,o))),o)) )).
```

The encoded type `leoFt(o,o)` indicates that the constant `abstrSX0SX0` is of type $o \to o$. The name `abstrSX0SX0` is derived from serialising the term $\lambda \mathrm{SX}_0 .\, \mathrm{SX}_0$, but no further definition of this constant is given by the translation. The new translation λ-lifts such terms fully, yielding the pair of axioms

```
fof(ll1,axiom,(
    ! [SX0] :
      ( leoLit(leoTi(leoAt(leoTi(ll1,leoFt(o,o)),
                           leoTi(SX0,o)),o))
    <=> leoLit(leoTi(SX0,o)) ) )).

fof(44,axiom,(
    ~ leoLit(leoTi(leoAt(leoTi(csK2_SY3,leoFt(leoFt(o,o),o)),
                         leoTi(ll1,leoFt(o,o))),o)) )).
```

2. The old translation was verbose, and its use potentially resulted in fairly large first-order formulas due to the encoding of type information. This verbosity causes additional overhead to the ATPs, and this contributes to ATPs missing their timeout to find a refutation. Arguably, the new translation is more verbose, since it tends to include more information. To address this problem we are experimenting with lighter encoding of type information. We have closely followed Claessen et al [11] to implement their monotonicity analysis by producing a SAT encoding, which we send to MiniSat using an interface adapted from Satallax [10, 3]. This translation can be used by giving Leo-II the argument `--translation fof_experiment`.

Problems which become provable in LEO-II using the new `fof_full` translation include NUM636^1.p and LCL631^1.p.

3.1.2 Backend ATPs.

Leo-II is mainly used in combination with E [15], and Leo-II version 1.5 features small improvements in how it interacts with E. Support for SPASS [21] was added during past experiments [8]. In version 1.5 we improved Leo-II's ATP interface and added support for various other backend ATPs, including remote provers on SystemOnTPTP [18].

3.2 Support for Axiom of Choice

The default semantics for THF0 is Henkin semantics with choice. Until version 1.5, Leo-II did not support reasoning with choice, unless naïve Skolemization was used—that is, first-order Skolemization without employing further restrictions (as investigated by Miller [12]). This enables limited reasoning with choice, and succeeds in some example cases, but it fails in many others [6, Section 3.2].

In order to extend Leo-II to support the axiom (scheme) of choice (AC), instances of AC could be automatically added to the input problem. An example is the following instance of AC for type $(\iota \to o) \to \iota$ (where o is the type of propositions as before, and ι is the type of individuals):

$$\exists E_{(\iota\to o)\to\iota} \forall P_{(\iota\to o)}.\ \exists X_\iota (P\ X) \Rightarrow P\ (E\ P) \tag{1}$$

However, such kinds of impredicative axioms should generally be avoided in automated proof search since they allow for simulation of the cut rule in any Henkin-complete THF prover [7].

Our approach involves adding two new rules to Leo-II: detectChoiceFn and choice. The first rule detects and removes instances of AC, such as (1) above, and keeps a register of choice functions CFs. CFs always contains at least one choice function symbol for each choice type. The second rule gives the semantics to choice functions. Taken together, these rules allow AC-valid reasoning without the risk of cut-simulation.

In more detail, rule detectChoiceFn removes choice-axiom clauses from the search space and registers the corresponding choice function symbols f in CFs.

$$\frac{[PX]^{\mathtt{ff}} \vee [P(f_{(\alpha\to o)\to\alpha} P)]^{\mathtt{tt}}}{\mathsf{CFs} \longleftarrow \mathsf{CFs} \cup \{f_{(\alpha\to o)\to\alpha}\}}\ \mathsf{detectChoiceFn}$$

In the notation used above, α is a metavariable ranging over types. $P_{\alpha\to o}$ is a set variable. Literals are enclosed in square brackets, and $\mathtt{ff}$ and $\mathtt{tt}$ indicate negative and position polarity respectively. The rule abuses standard notation: the rule does not describe a logical inference, since the conclusion of the rule indicates a side-effect which extends the set CFs of choice functions.

Rule choice investigates whether a term $\epsilon_{(\alpha\to o)\to\alpha}\mathbf{B}_{\alpha\to o}$ (where $\epsilon \in \mathsf{CFs}$ is a registered choice function or a free variable) is contained as a subterm of a literal $[\mathbf{A}]^p$ in a clause C. In this case it adds the instantiation of AC at type $(\alpha \to o) \to \alpha$, and with term $\mathbf{B}_{\alpha\to o}$ to the search space. Side-conditions guard against unsound reasoning, such as the 'uncapturing' of free variables in $\mathbf{B}$:

$$\frac{C := \mathbf{C}' \vee [\mathbf{A}[E_{(\alpha\to o)\to\alpha}\mathbf{B}]]^p \qquad \begin{array}{c}\epsilon \in \mathsf{CFs},\ E = \epsilon\ \mathit{or}\ E \in \mathit{freeVars}(C),\\ \mathit{freeVars}(\mathbf{B}) \subseteq \mathit{freeVars}(C), Y\ \mathit{fresh}\end{array}}{[\mathbf{B}\ Y]^{\mathtt{ff}} \vee [\mathbf{B}\ (\epsilon_{(\alpha\to o)\to\alpha}\mathbf{B})]^{\mathtt{tt}}}\ \mathsf{choice}$$

Rules detectChoiceFn and choice are obviously sound: detectChoiceFn simply removes clauses from the search space, and for any choice function f, the rule choice only introduces new instances of the corresponding choice axiom.

There is a correspondence with the handling of choice in Satallax. Satallax too considers only selective instantiations of AC in order to avoid cut-simulation. For instance, when (1) is assumed, the terms $\mathbf{T}$ which Satallax considers to be eligible instantiations for variable P are those occurring in formulas of the following forms in a tableau branch (and where ϵ is a choice function): $(\epsilon\ \mathbf{T})\ \mathbf{S_1}\ \ldots\ \mathbf{S_n}$ or $\neg((\epsilon\ \mathbf{T})\ \mathbf{S_1}\ \ldots\ \mathbf{S_n})$, or the disequations $(\epsilon\ \mathbf{T})\ \mathbf{S_1}\ \ldots\ \mathbf{S_n} \neq \mathbf{S}$ or $\mathbf{S} \neq (\epsilon\ \mathbf{T})\ \mathbf{S_1}\ \ldots\ \mathbf{S_n}$. It is easy to see that our rule choice, which is less restrictive, subsumes these cases. We also experimented with Satallax's approach in LEO-II but this led to worse results. Our choice rule is more closely related to that of Mints [13]. Use of the choice rules can be disabled using the `-nuc` command-line switch. A completeness proof for LEO-II's improved handling of choice remains future work.

Problems which become provable in LEO-II using our improved support for choice include SYO517^1.p, SYO534^1.p–SYO537^1.p, and SYO555^1.p.

3.3 Detection of defined equality

Primitive equality in HOL refers to the use of the interpreted constant '='. Equality can also be *defined* in HOL—for example, as

$$\lambda X_\alpha \lambda Y_\alpha \forall P_{\alpha\to o}.\ P\ X \Rightarrow P\ Y$$

or

$$\lambda X_\alpha \lambda Y_\alpha \forall Q_{\alpha\to\alpha\to o}.\ \forall Z_\alpha (Q\ Z\ Z) \Rightarrow Q\ X\ Y$$

The former is known as Leibniz equality and the latter we call Andrews equality (cf. [1], Exercise X5303). Both Leibniz and Andrews equality support cut-simulation due to their impredicative nature [7], and should thus be avoided in proof automation. In fact, using primitive, rather than defined, equality may save many *primitive substitution* steps in proofs. Such steps involve instantiations of set variables, and this generally involves blind guessing. Examples of the benefit of using primitive, rather than defined, equality have been given in the literature [6, Sections 5.1 and 5.2]. In order to address this issue we added the following two rules to LEO-II's calculus; they instantiate the variable P with primitive equality:

$$\frac{\mathbf{C} \lor [P\ \mathbf{A}]^{\mathrm{ff}} \lor [P\ \mathbf{B}]^{\mathrm{tt}}}{\mathbf{C}\{\lambda X.\ \mathbf{A} = X/P\} \lor [\mathbf{A} = \mathbf{B}]^{\mathrm{tt}}}\ \mathsf{LeibEQ} \qquad \frac{\mathbf{C} \lor [P\ \mathbf{A}\ \mathbf{A}]^{\mathrm{ff}}}{\mathbf{C}\{\lambda X \lambda Y.\ X = Y/P\}}\ \mathsf{AndrEQ}$$

Soundness of LeibEQ and AndrEQ is obvious, since both rules simply realise specific instances of primitive substitution. For improved configurability, either rule can be individually disabled from the command-line by using the switches `-nrleq` and `-nraeq` respectively. If LeibEQ is used in combination with the new FOF translations (see Section 3.1) several TPTP problems whose previous SZS [17] status was 'Unknown' can now be solved by LEO-II. Examples include SYO246^5.p, SYO244^5.p, NUM817^5.p, NUM816^5.p and NUM814^5.p. There are also many problems that can now be solved with primitive substitution (blind guessing) disabled when LeibEQ and AndrEQ are available. Examples include SEV081^5.p, SEV158^5.p, SEV992^1.p, and SYO276^5.p. Overall, these two new rules lead to significantly better coverage using the lighter primitive-substitution search modes `-ps 0` or `-ps 1`.

3.4 Strategy scheduling

Strategy schedules were added to LEO-II in version 1.2 and the catalogue of schedules has slowly increased in the versions that followed. In version 1.5 we recoded the strategy-scheduling feature to facilitate the encoding of new strategies, to improve code reuse with other parts of LEO-II, and to have greater flexibility when encoding strategies. The new setup affords greater flexibility: for example, the new setup can schedule varying number of strategies (depending on the problem being processed) and each schedule could be of varying duration. This has opened up many opportunities for experimentation and tuning.

We are also interested in computing strategies on-the-fly based on problem characteristics, and version 1.5 carries out some small initial checks (e.g. size of the problem, and whether it contains instances of AC), and schedules strategies based on that limited analysis. Optimising this further remains as future work.

3.5 Other improvements

Numerous other additions were made to Leo-II. Previously, Leo-II was entirely focused on refutation: that is, until version 1.5, in terms of the SZS classification, Leo-II would judge a problem to be a Theorem (if a refutation exists), Unsatisfiable (if the problem's axioms themselves can be refuted), or diverge (by extending the preunification depth and reattempting a refutation). It can now classify Satisfiable problems and detect CounterSatisfiable problems, thus improving both Leo-II's precision and termination behaviour. The added support for choice was very relevant for achieving this. Leo-II decides that a problem is Satisfiable when the problem consists of a collection of axioms (lacking a conjecture) and Leo-II succeeds in saturating a set of clauses (without finding a refutation, otherwise the problem would be classified as Unsatisfiable).

Leo-II's unification algorithm has been redone, and can be set (from the command-line) to disregard Boolean and functional extensionality. This has strengthened Leo-II's behaviour in non-extensional problems, since disabling the extensional behaviour shrinks the search space.

Numerous other improvements and fixes have been made: these range from system features (such as the parser, status reporting, avoiding redundant computations, etc) to deeper areas in the calculus and main loop (including factorisation, subsumption, and clause selection).

4 Future work

We have started experimenting with using term orderings to influence literal selection. We also plan to revise Leo-II's internals to make full use of the potential benefit they offer. For instance, the shared term graph is currently underutilised.

More work is needed to compute better schedules, paired with better problem analyses. Such analyses can determine the scheduling of specific strategies, which can be better tuned to the problem.

The ATP interface can be improved further to call multiple backend ATPs in parallel. Experiments comparing 30-second invocations of Leo-II on all THF problems, supported by provers E (version 1.6), SPASS (version 3.5) [21] and Vampire (version 2.6) [14] showed us that there were 37, 5 and 20 theorems that were proved exclusively by Leo-II(E), Leo-II(SPASS) and Leo-II(Vampire), respectively. And there were 31, 95 and 98 theorems that Leo-II(E), Leo-II(SPASS) and Leo-II(Vampire) missed, but which one of the others could prove.

Supporting various ATP backends increases the scope for peephole optimisation; we have not yet investigated this. The translation module can be optimised further, and extended to target more formats. Table 1 one shows how the new HOL-to-FOL translation (`fof_full`) and its lighter variant (`fof_experiment`) are superior to Leo-II's preexisting encoding (`fully_typed`). In future work we plan to improve `fof_experiment` further and make it the default translation.

SZS Status	`fully-typed`	`fof_full`	`fof_experiment`
Thm	64.8	64.9	65.3
All	60.9	61	61.3

Table 1: Comparing FOL encodings in LEO-II 1.5 (30s timeout). Table shows the percentage of matches between LEO-II's SZS output and the 'Status' field of problems.

Timeout (s)	**v1.2**		**v1.4.3**		**v1.5**	
	Thm	*All*	*Thm*	*All*	*Thm*	*All*
30	58.4	51.1	62.1	54.4	64.3	61.3
60	58.7	51.3	65	56.9	67.1	62.9

Table 2: Percentage match between different versions of LEO-II and the Status field of TPTP problems. LEO-II version 1.2 was the winner of the CASC competition in 2010, and version 1.4.3 was the last public release. Version 1.5 was run with the `fof_experiment` encoding.

5 Conclusion

Version 1.5 of LEO-II includes various improvements which affect its performance and coverage. To obtain a broader picture, we compared the results of using LEO-II version 1.5 with earlier versions, and the results are shown in Table 2. In this experiment we counted the matches between LEO-II's SZS output and the TPTP problem's SZS status (included in its header).[1] All the net gains are positive, but a more thorough evaluation (on different benchmarks, and considering various parameters) remains as future work. Within a 30s timeout, LEO-II version 1.5 can classify 196 more problems than its predecessor. The main boost ($\frac{125}{196}$ problems) in this version is provided by the detection of non-theorems (i.e. satisfiable or countersatisfiable problems).

Acknowledgement. Work on LEO-II was supported by the UK Engineering and Physical Sciences Research Council (grant number EP/D070511/1), and the German Research Foundation under grant BE 2501/9-1. The second author was supported by a grant from the German Academic Exchange Service (DAAD) during a visit to Freie Universität Berlin, during which the work described in this paper was carried out. We thank Chad Brown for allowing us to use Satallax's MiniSat interface, and thank Mike Gordon, Grant Passmore, Larry Paulson, and the anonymous referees for feedback on this article.

References

[1] P.B. Andrews. *An Introduction to Mathematical Logic and Type Theory: To Truth Through Proof.* Applied Logic Series. Springer, 2002.

[2] P.B. Andrews. Church's type theory. In Edward N. Zalta, editor, *The Stanford Encyclopedia of Philosophy.* Spring 2009 edition, 2009.

[3] J. Backes and C.E. Brown. Analytic tableaux for higher-order logic with choice. *Journal of Automated Reasoning*, 47(4):451–479, 2011.

[4] C. Benzmüller. Extensional higher-order paramodulation and RUE-resolution. In *Proceedings of CADE-16*, number 1632 in LNCS, pages 399–413. Springer, 1999.

[1]This also means that 'Unknown' problems which LEO-II now classifies as 'Theorem' count against us, but this experiment was only intended to offer a rough idea of progress.

[5] C. Benzmüller. Comparing approaches to resolution based higher-order theorem proving. *Synthese*, 133(1-2):203–235, 2002.

[6] C. Benzmüller and C.E. Brown. A structured set of higher-order problems. In *Proceedings of TPHOLs 2005*, number 3603 in LNCS, pages 66–81. Springer, 2005.

[7] C. Benzmüller, C.E. Brown, and M. Kohlhase. Cut-simulation and impredicativity. *Logical Methods in Computer Science*, 5(1:6):1–21, 2009.

[8] C. Benzmüller, L. Paulson, F. Theiss, and A. Fietzke. Progress report on LEO-II – an automatic theorem prover for higher-order logic. In *TPHOLs 2007 Emerging Trends Proceedings*, pages 33–48. Internal Report 364/07, Department of Computer Science, University Kaiserslautern, Germany, 2007.

[9] C. Benzmüller, F. Theiss, L. Paulson, and A. Fietzke. LEO-II - a cooperative automatic theorem prover for higher-order logic. In *Proceedings of IJCAR 2008*, volume 5195 of *LNCS*, pages 162–170. Springer, 2008.

[10] C.E. Brown. Reducing higher-order theorem proving to a sequence of sat problems. *Journal of Automated Reasoning*, 51(1):57–77, 2013.

[11] K. Claessen, A. Lillieström, and N. Smallbone. Sort it out with monotonicity: translating between many-sorted and unsorted first-order logic. In *Proceedings of CADE 2011*, pages 207–221, Berlin, Heidelberg, 2011. Springer-Verlag.

[12] D.A. Miller. *Proofs in Higher-Order Logic*. PhD thesis, Carnegie Mellon University, 1983.

[13] G. Mints. Cut-elimination for simple type theory with an axiom of choice. *Journal of Symbolic Logic*, pages 479–485, 1999.

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

[15] S. Schulz. E – A Brainiac Theorem Prover. *AI Communications*, 15(2/3):111–126, 2002.

[16] N. Sultana and C. Benzmüller. Understanding LEO-II's proofs. In E. Ternovska, K. Korovin, and S. Schulz, editors, *The 9th International Workshop on the Implementation of Logics (IWIL-2012, affiliated with LPAR-2012)*, Merida, Venezuela, 2012.

[17] G. Sutcliffe. The SZS ontologies for automated reasoning software. In *Proceedings of the LPAR Workshops: Knowledge Exchange: Automated Provers and Proof Assistants, and The 7th International Workshop on the Implementation of Logics*, volume 418, pages 38–49. CEUR Workshop Proceedings, 2008.

[18] G. Sutcliffe. The TPTP problem library and associated infrastructure. *Journal of Automated Reasoning*, 43(4):337–362, 2009.

[19] G. Sutcliffe and C. Benzmüller. Automated Reasoning in Higher-Order Logic using the TPTP THF Infrastructure. *Journal of Formalized Reasoning*, 3(1):1, 2010.

[20] G. Sutcliffe, S. Schulz, K. Claessen, and P. Baumgartner. The TPTP typed first-order form with arithmetic. In *Logic for Programming, Artificial Intelligence, and Reasoning*, volume 7180 of *LNCS*, pages 406–419. Springer, 2012.

[21] C. Weidenbach, D. Dimova, A. Fietzke, R. Kumar, M. Suda, and P. Wischnewski. Spass version 3.5. In *Proceedings of the CADE-22*, volume 5663 of *LNAI*, pages 140–145. Springer, 2009.