Learning-Guided Higher-Order Automated Reasoning for Isabelle/HOL

Jan Jakubův[1], Cezary Kaliszyk[2], and Martin Suda[1]

[1] Czech Technical University in Prague, Czech Republic
[2] University of Melbourne, Australia

Abstract. We present higher-order versions of ENIGMA and Deepire, the two prominent learning-guided reasoning systems extending saturation-based automatic theorem provers E and Vampire, respectively. We evaluate these on a large Isabelle/HOL corpus, which emulates their use in Sledgehammer, and observe consistent improvement. Stress-testing the Sledgehammer export revealed several bugs, both in the provers and in Sledgehammer itself, which were all fixed for the final evaluation.

1 Introduction

Interactive theorem provers (ITPs) such as Isabelle/HOL [16] combine expressive logics, large formal libraries, and trusted proof checking. The very richness of these libraries, however, means that many proof obligations benefit from stronger external automation than is practical to provide inside the ITP alone. Automation and in particular Hammer systems [4] have therefore become a central component of formalization workflow: hammers select relevant facts from the imported libraries, translate the resulting problem to formats accepted by external automatic theorem provers (ATPs), and reconstruct successful proofs back in the logic of the prover. In practice, Sledgehammer [5], i.e., the hammer in Isabelle/HOL, is one of the main interfaces through which advances in automated reasoning become available to Isabelle users. Two complementary learning problems arise in this context. The first is premise selection: large formal libraries contain many potentially relevant facts, but only a small subset is usually needed for a given conjecture. The second is guidance inside the ATP once an exported problem has been fixed. Learning has proved highly effective at each of these stages in first-order reasoning [11,23,8]. In particular, Sledgehammer already benefits from learned relevance filtering [15,14], while ENIGMA [10]—the learning-guided version of E—and Deepire [21]—the learning-guided version of Vampire—have shown that learned clause guidance can substantially improve saturation-based proof search.

Recent higher-order (HO) support in E [20,27], called λE, and Vampire [1,2] opens the possibility of extending such learning guidance to HO ATP problems originating from Isabelle. This is what we report on in this paper. We tried to stay as close as possible to the Isabelle source language (in particular including polymorphism for the guided Vampire) while at the same time evaluating the impact of guidance on the performance of AI/ATP automation.

Table 1: Isabelle/Sledgehammer encodings used for the export.

Target	Type encoding	λ translation
TF0	`mono_native`	`liftingN`
TH0	`mono_native_higher`	`keep_lamsN`
TH1	`poly_native_higher`	`keep_lamsN`

Contributions. This paper firstly introduces (1) λENIGMA[3] and Deepire–2.0–HO[4]—extensions of ENIGMA and Deepire to the HO setting, (2) improves the Isabelle/Sledgehammer export pipeline—including fixing several errors and further issues with Sledgehammer polymorphism, and (3) provides a preliminary evaluation on the resulting corpus—suggesting future research directions. Additional resources are provided in Appendix A.

2 Isabelle/HOL Export and Benchmark Dataset

To create our benchmark, we replayed Isabelle/HOL developments using the Mirabelle testing infrastructure. This follows *Judgement Day* [5]: at intermediate proof states (i.e., proof obligations arising during the interactive verification), we collected the corresponding ATP tasks using the Sledgehammer export.

All exports were produced from the last stable Isabelle-2025-2 release, augmented with a few backported fixes from the Isabelle development, as discussed below. We invoked `isabelle mirabelle` over the HOL session and let Mirabelle call Sledgehammer requesting individual slices of 512 facts, without preprocessing and keeping the generated problems. We exported three formats, namely TF0 [26], TH0 [25], and TH1 [12] (see Table 1). The TPTP family TF0 is monomorphically typed first-order logic (with equality), TH0 is monomorphically typed higher-order logic, and TH1 adds rank-1 polymorphism. The TF0 export is included for completeness and comparability with earlier Sledgehammer-based experiments [7], whereas TH0 and TH1 are the main HO targets.

Table 1 shows the Sledgehammer encoding settings for each format. The type encoding controls how polymorphism is handled (`mono_native` for first-order monomorphic, `poly_native_higher` for polymorphic HO) and the λ translation controls how lambda terms are exported. Current HO Vampire does not support the full polymorphic TH1 feature set. In particular we had to switch off `if`-then-`else`, `let`, and the Hilbert choice operator ε for all encodings. This gives us the closest encoding to the native higher-order Isabelle language that the provers can currently accept. Sledgehammer offers several relevance filters [3]. We used MeSh throughout: it combines syntactic similarity with machine-learned predictions, producing less noisy premises in preliminary experiments.

Running this export at scale exposed several issues in Isabelle's Sledgehammer infrastructure. First, a name clash introduced during η-expansion in the TPTP generator could produce type-invalid problems; this affected many exported tasks in HOL-Analysis. Second, the HOL-Nominal theory `Class1` suffered

[3] Available at `https://github.com/ai4reason/eprover/tree/enigma`.
[4] Available at `https://github.com/vprover/vampire/tree/mtpa-gnn-hol`.

from clashes between names and conames. Both problems have since been fixed in the Isabelle development version. In addition, we found smaller monomorphisation issues that are being addressed. These fixes improve the performance of automation not just for our provers but for all users of Sledgehammer.

3 Higher-Order ENIGMA and Deepire

Both E and Vampire are based on a *given-clause saturation loop* [17], in which clauses are iteratively selected and used to derive new clauses via inference rules until saturation or refutation is reached. Selection of the next *given clause* is the most critical choice point in the proof search. Accordingly, both ENIGMA and Deepire use machine learning to guide decisions at this choice point.

Higher-order (HO) extensions of both E and Vampire must represent HO terms and types. In TPTP, term application is expressed using the operator @. While in the first-order (FO) case applications are typically written as $\mathsf{f}(\mathsf{a},\mathsf{b})$, the HO syntax allows $(\mathsf{f}\ @\ \mathsf{a}\ @\ \mathsf{b})$, with partial applications such as $(\mathsf{f}\ @\ \mathsf{a})$ also permitted. Additionally, λ-abstractions of the shape $(\lambda \mathsf{x} : \tau.\mathsf{T})$ need to be represented, where x is a bound variable of type τ and T is the body. HO types are encoded as *arrow types*, e.g., $(\mathsf{int} \to (\mathsf{int} \to \mathsf{int}) \to \mathsf{bool})$, where each component is either atomic or itself an arrow type. The polymorphic TH1 further supports type variables, parametric type constructors, and (universal) type quantification, e.g., $(\forall \alpha.\ \mathsf{list}(\alpha) \to (\alpha \to \mathsf{int}) \to \mathsf{int})$ expresses a polymorphic list-to-int signature.

HO extensions of both ENIGMA and Deepire operate on these representations. System-specific details are covered in the rest of this section.

3.1 λENIGMA: Higher-Order ENIGMA for λE

E Prover [18,19] has been extended with HO support since version 3.0 [20,27]. The HO E, also known as λE, currently supports monomorphically typed HO formulas, that is, the TH0 fragment of TPTP. There is currently no support for TH1-style polymorphism. Extending E to λE was a non-trivial task, concerning mainly (1) representation of HO terms, (2) HO inference rules, and (3) HO unification. However, the core structure of the solver is preserved, with the *saturation loop* and *clause selection* playing the key roles in E's functionality. Hence, the key challenge for λENIGMA is the representation of HO terms.

ENIGMA [10,6] is a given clause guidance system for E that can learn from previous successful proof searches, extract information about useful and useless clauses, and use this knowledge to guide future proof searches on similar conjectures. The core of ENIGMA functionality is a clause feature extractor which feeds a gradient boosted decision tree framework. Any successful proof search of E can be used by ENIGMA to extract training examples. For this purpose, clauses that are used by the proof are classified as positive, and others as negative. The clauses are then represented by numeric feature vectors and a decision tree model is trained. The ENIGMA model is then tied to E's saturation loop, influencing the prover to select clauses classified as positive.

In λE, both FO and HO terms are represented by a uniform recursive data structure storing a top-level symbol code (integer) and a list of subterms. For applications such as (FXY), where the head F is a compound term, λE introduces an internal symbol @, encoding the term as @(F, X, Y) with the head as the first subterm followed by its arguments. λ-abstractions (λx.T) are represented using a special binary symbol as λ(x, T).

ENIGMA extracts *vertical* and *horizontal* features by traversing syntax trees of clause terms. Vertical features correspond to top-down paths of bounded length in the syntax tree; e.g., from f(a, g(b, X)), it extracts (f, a), (f, g, b), and (f, g, ⋆), where variables are replaced by a placeholder ⋆. Horizontal features provide a flattened view capturing a head symbol and the heads of its arguments, e.g., f(a, g) and g(b, ⋆). Features are collected from each clause, hashed into vector indices, and represented as occurrence-count vectors.

Since λE uses the same term structure as E, λENIGMA extends this process to HO terms. For λ(x, T), the variable name is omitted and features record the binder position. For applications such as @(F, X, Y), arguments X and Y are treated as children of F rather than siblings. Types are incorporated by embedding symbol types into features: a monomorphic arrow type like (i → (i → i) → o) is encoded as the string i-(i-i)-o and appended to symbol names using : as separator before hashing. For example, a typed λENIGMA vertical feature might look like (f:i-i, a:i), and a horizontal one like f:i-(i-i)-i(a:i, g:i-i). Finally, name-independent features [9] replace symbol names with their arities. Hence, in name-independent λENIGMA, all atomic types (except the built-in types i and o for individuals and propositions) are mapped to a common placeholder.

3.2 Deepire–2.0–HO: Higher-Order Deepire 2.0

Deepire 2.0 [22] is an extension of the ATP Vampire [1,13] with neural clause selection guidance as described by Suda [23,24]. In this work, we adapt Deepire 2.0, originally designed to work with only (multi-sorted) FO problems, to handle (polymorphic) HO problems. Vampire's HOL support currently resides in a dedicated `hol` branch (see Appendix A for additional resources). A short paper [2] describes the functionality implemented therein. While a reimplementation of the HOL capabilities for the main development branch is currently underway [1], we adopt the `hol` branch as our basis and extend it with neural clause selection guidance along the lines of Deepire 2.0.

Deepire–2.0–HO works with the same overall neural-network architecture for discriminating clauses as its FO predecessor [23,24]. The architecture consists of 1) a Graph Neural Network (GNN) applied once—after preprocessing, but before actual saturation starts—to the input problem's Clause Normal Form (CNF). The task of the GNN is to compute *embeddings* (vectorial representations) of the problem's signature symbols and initial clauses. 2) Two Recursive Neural Networks (RvNN), one unfolding along the syntactic structure of the terms and literals forming the derived clauses (and using the symbol embedding at the leaves) and one unfolding along the derivation history of the clauses (and using the initial clause embeddings at the leaves). 3) A fully connected network

combining the corresponding outputs of the two RvNNs together with several simple, hand-crafted features into a single score of a given clause.[5]

To handle polymorphic HO problems, we adapted the GNN part, where we added a new kind of nodes for type constructors and promoted sorts to full-fledged expressions while allowing for sort variables. We made sure universal syntactic concepts, which should have common meaning across all problems, are easy to recognize by the GNN, by labelling the corresponding graph nodes with distinct features. These concepts include the arrow type constructor, propositional connectives and FO quantifiers (which may appear explicitly in Vampire's HO CNF), the apply and lambda constructs, and the de Bruijn indices.[6]

Because HO terms are only built of applications and constants (i.e., they cannot explicitly feature arbitrary-arity function symbols as in FOL), we could simplify the term-structure RvNN and hardcode a fixed number of arguments to the recursive case.[7] Analogously to the FO version, we use a single embedding to represent all term variables (and a different single embedding to represent the type variables). Finally, we added two HO-specific hand-crafted features to the input of the final fully connected network, namely the number of applied variables and number of lambdas occurring in a clause.

4 Evaluation and Future Work

The full export described in Sect. 2 yielded 246 277 problems (in each of the discussed dialects, cf. Table 1). For the experiments[8], we sampled 120 000 problems for training and held out 5000 for testing (aligned across dialects).

While ENIGMA has previously been successfully evaluated on FOF/TF0 problems from Isabelle [7], this paper presents the first results in the HO setting. We trained λENIGMA on 29 000 problems and reserved 1000 for development. After extracting the training data, we built models with different hyperparameters and used the development set to select the best one based on ATP performance rather than ML metrics. The selected model was then evaluated on the training problems to generate new training examples, and this process was repeated 10 times. Each iteration takes approximately 12 hours, using a time limit of 10 s per prover run.

We ran Deepire–2.0–HO on TH0 and TH1 and, for comparison, Deepire 2.0 on TF0. We used a limit of 16 billion CPU instructions (roughly $\sim$6 s) per problem (see Appendix C). For each of the three dialects, a strong base strategy was obtained by local search through Vampire's configuration space. We then ran 11 iterations of the clause selection guidance improvement loop [23], which took roughly 4.5 days on the server in each case.

[5] See Appendix A and B of [24] for all relevant details.

[6] See Sect. 4 of [2] for an explanation of the applicative encoding of HO syntax into polymorphic FOL used by Vampire.

[7] We use 2 type arguments and 2 term arguments, which are an exact match for Vampire's apply construct, and crop or pad to exactly four for anything else.

[8] On dual-socket AMD EPYC 7513 (32 cores, 2.6 GHz); 500 GB RAM; Ubuntu LTS.

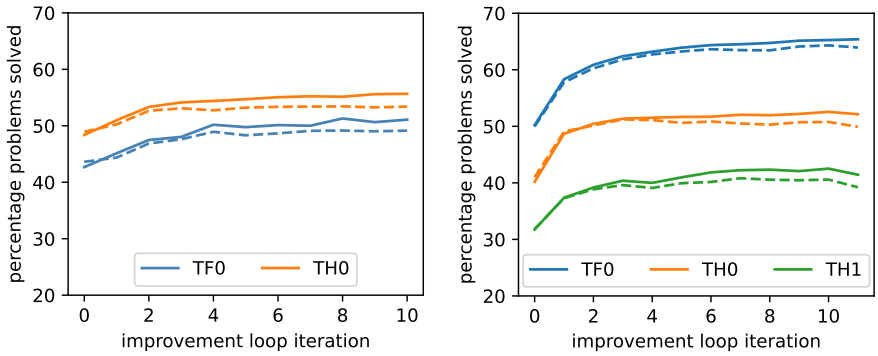

Fig. 1: λENIGMA (left) and Deepire–2.0–HO (right) performance on Isabelle's exports TF0, TH0, and TH1. Solid/dashed lines show the train/test performance, resp.

The performance of λENIGMA during the evaluation–training loop is shown in Fig. 1 (left). Iteration 0 (the leftmost point on the x-axis) corresponds to the non-learning baseline strategy—the best strategy without guidance—for the corresponding fragment (see Appendix B). Subsequent iterations show the effect of learning guidance. We observed that λE itself performs better on TH0 than on TF0, and on TH0 it even outperforms Vampire. Moreover, λENIGMA improved upon the baseline on both fragments in a similar manner. On the testing data, λENIGMA reached 9.12 % improvement over the baseline on TF0 (from 48.9 % to 53.4 %), and 12.66 % improvement over the baseline on TH0 (from 43.6 % to 49.1 %). Further improvements are expected on larger benchmarks.

The performance of Deepire–2.0–HO is shown in Fig. 1 (right), with iteration 0 again representing the non-learning Vampire baseline. We can see that for Vampire, TF0 is the strongest dialect, reaching ∼65 % test coverage with neural guidance. On the higher-order fragments, the relative improvement brought about by the neural guidance of Deepire–2.0–HO is of comparable magnitude: +27 % on TH0 (from 40.2 % to 51.2 %) and +28 % on TH1 (from 31.8 % to 40.8 %). However, this does not make up for the lower respective starting points. We also see that Vampire/Deepire–2.0–HO currently does not capitalize on the "most native" polymorphic encoding TH1. Finally, the test performance dwindles sooner for the HO encodings than for TF0, which could also explain why even the train performance does not grow there so much.[9] The fact that Deepire–2.0–HO (on both TH0 and TH1) currently fares worse than Deepire 2.0 (on TF0) is a bit disappointing. It is not immediately clear what the main cause could be[10] and we plan to focus on this question in future work. One candidate ex-

[9] Good generalization (which we indirectly observe through the test performance) is a prerequisite for subsequent iterations claiming new train problems to learn from.

[10] We are looking at different versions of the prover, implementing substantially different calculi, running different strategies on differently encoded original problems.

Table 2: Greedy cover of all λENIGMA and Deepire–2.0–HO strategies (best per fragment per system).

prover	*fragment*	*solves*	*addon*		*total*	
Deepire	TF0	3229	+3229	−	= 3229	= 64.58%
λENIGMA	TH0	2670	+145	+4.49%	= 3374	= 67.48%
Deepire	TH1	2041	+66	+1.96%	= 3440	= 68.80%
Deepire	TH0	2561	+24	+0.70%	= 3464	= 69.28%
λENIGMA	TF0	2440	+16	+0.46%	= 3480	= 69.60%

planation is that the HO fragments simply contain harder problems on average, leading to faster saturation of the performance gains from learning. Cross-dialect transfer—using guidance trained on TH0 to bootstrap TH1 training, or running a jointly trained model across dialects—is one avenue we intend to explore.

Another important direction is to scale training to the full 246 277-problem corpus. The current experiments used only 120 000 problems; training on all available data is expected to further improve performance. The bottleneck is training time: each λENIGMA iteration already requires roughly 12 h, and GPU-accelerated training would be needed to make full-corpus iterations practical. For Deepire–2.0–HO, the neural network is already the most expensive component, and GPU-based training is likewise a natural next step.

To assess the complementarity of the five strategies, we constructed a greedy cover of the combined portfolio (Table 2). Starting from the best single strategy (Deepire on TF0, solving 64.58 %), each step adds the strategy that solves the most problems not yet covered. Adding λENIGMA on TH0 yields the largest gain (+4.49 %), bringing the total to 67.48 %. All five strategies together reach 69.60 %, demonstrating that the HO strategies already provide a valuable complementary contribution on top of the FO baseline. A more detailed breakdown by prover and by fragment is provided in Appendix D.

To summarize, these are the first results for learning-guided HO automated reasoning on a large Isabelle/HOL benchmark, and they are encouraging. Both λENIGMA and Deepire–2.0–HO consistently improve over the corresponding non-learning baselines on the monomorphic fragments TF0 and TH0, and their five-strategy portfolio collectively solves 69.60 % of the test problems, substantially above any individual strategy. The challenge of the polymorphic TH1 fragment, together with the identified scaling opportunities, means that there is ample room for improvement as the area of learning-guided higher-order reasoning continues to mature.

Acknowledgments

This work was supported by the Renaissance Philanthropy grant DEEPER.

References

1. Bártek, F., Bhayat, A., Coutelier, R., Hajdú, M., Hetzenberger, M., Hozzová, P., Kovács, L., Rath, J., Rawson, M., Reger, G., Suda, M., Schoisswohl, J., Voronkov, A.: The Vampire diary. In: Piskac, R., Rakamaric, Z. (eds.) Computer Aided Verification - 37th International Conference, CAV 2025, Zagreb, Croatia, July 23-25, 2025, Proceedings, Part III. pp. 57–71. LNCS, Springer (2025). https://doi.org/10.1007/978-3-031-98682-6_4
2. Bhayat, A., Suda, M.: A higher-order Vampire (short paper). In: Benzmüller, C., Heule, M.J.H., Schmidt, R.A. (eds.) Automated Reasoning - 12th International Joint Conference, IJCAR 2024, Nancy, France, July 3-6, 2024, Proceedings, Part I. pp. 75–85. LNCS, Springer (2024). https://doi.org/10.1007/978-3-031-63498-7_5
3. Blanchette, J.C., Greenaway, D., Kaliszyk, C., Kühlwein, D., Urban, J.: A learning-based fact selector for isabelle/hol. J. Autom. Reason. **57**(3), 219–244 (2016). https://doi.org/10.1007/S10817-016-9362-8
4. Blanchette, J.C., Kaliszyk, C., Paulson, L.C., Urban, J.: Hammering towards QED. J. Formaliz. Reason. **9**(1), 101–148 (2016). https://doi.org/10.6092/ISSN.1972-5787/4593
5. Böhme, S., Nipkow, T.: Sledgehammer: Judgement day. In: Giesl, J., Hähnle, R. (eds.) Automated Reasoning, 5th International Joint Conference, IJCAR 2010, Edinburgh, UK, July 16-19, 2010. Proceedings. pp. 107–121. Lecture Notes in Computer Science, Springer (2010). https://doi.org/10.1007/978-3-642-14203-1_9
6. Goertzel, Z.A., Chvalovský, K., Jakubův, J., Olsák, M., Urban, J.: Fast and slow enigmas and parental guidance. In: Konev, B., Reger, G. (eds.) Frontiers of Combining Systems - 13th International Symposium, FroCoS 2021, Birmingham, UK, September 8-10, 2021, Proceedings. pp. 173–191. Lecture Notes in Computer Science, Springer (2021). https://doi.org/10.1007/978-3-030-86205-3_10
7. Goertzel, Z.A., Jakubův, J., Kaliszyk, C., Olsák, M., Piepenbrock, J., Urban, J.: The isabelle ENIGMA. In: Andronick, J., de Moura, L. (eds.) 13th International Conference on Interactive Theorem Proving, ITP 2022, Haifa, Israel, August 7-10, 2022. pp. 16:1–16:21. LIPIcs, Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2022). https://doi.org/10.4230/LIPICS.ITP.2022.16
8. Jakubův, J., Chvalovský, K., Goertzel, Z.A., Kaliszyk, C., Olsák, M., Piotrowski, B., Schulz, S., Suda, M., Urban, J.: Mizar 60 for mizar 50. In: Naumowicz, A., Thiemann, R. (eds.) 14th International Conference on Interactive Theorem Proving, ITP 2023, Białystok, Poland, July 31 - August 4, 2023. pp. 19:1–19:22. LIPIcs, Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2023). https://doi.org/10.4230/LIPICS.ITP.2023.19
9. Jakubův, J., Chvalovský, K., Olsák, M., Piotrowski, B., Suda, M., Urban, J.: ENIGMA anonymous: Symbol-independent inference guiding machine (system description). In: Peltier, N., Sofronie-Stokkermans, V. (eds.) Automated Reasoning - 10th International Joint Conference, IJCAR 2020, Paris, France, July 1-4, 2020, Proceedings, Part II. pp. 448–463. Lecture Notes in Computer Science, Springer (2020). https://doi.org/10.1007/978-3-030-51054-1_29
10. Jakubův, J., Urban, J.: ENIGMA: efficient learning-based inference guiding machine. In: Geuvers, H., England, M., Hasan, O., Rabe, F., Teschke, O. (eds.) Intelligent Computer Mathematics - 10th International Conference, CICM 2017, Edinburgh, UK, July 17-21, 2017, Proceedings. pp. 292–302. Lecture Notes in Computer Science, Springer (2017). https://doi.org/10.1007/978-3-319-62075-6_20

11. Jakubův, J., Urban, J.: Hammering mizar by learning clause guidance (short paper). In: Harrison, J., O'Leary, J., Tolmach, A. (eds.) 10th International Conference on Interactive Theorem Proving, ITP 2019, Portland, OR, USA, September 9-12, 2019. pp. 34:1–34:8. LIPIcs, Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2019). https://doi.org/10.4230/LIPICS.ITP.2019.34
12. Kaliszyk, C., Sutcliffe, G., Rabe, F.: TH1: the TPTP typed higher-order form with rank-1 polymorphism. In: Fontaine, P., Schulz, S., Urban, J. (eds.) Proceedings of the 5th Workshop on Practical Aspects of Automated Reasoning co-located with International Joint Conference on Automated Reasoning (IJCAR 2016), Coimbra, Portugal, July 2nd, 2016. pp. 41–55. CEUR Workshop Proceedings, CEUR-WS.org (2016), https://ceur-ws.org/Vol-1635/paper-05.pdf
13. Kovács, L., Voronkov, A.: First-order theorem proving and vampire. In: Sharygina, N., Veith, H. (eds.) Computer Aided Verification - 25th International Conference, CAV 2013, Saint Petersburg, Russia, July 13-19, 2013. Proceedings. pp. 1–35. LNCS, Springer (2013). https://doi.org/10.1007/978-3-642-39799-8_1
14. Kühlwein, D., Blanchette, J.C., Kaliszyk, C., Urban, J.: Mash: Machine learning for sledgehammer. In: Blazy, S., Paulin-Mohring, C., Pichardie, D. (eds.) Interactive Theorem Proving - 4th International Conference, ITP 2013, Rennes, France, July 22-26, 2013. Proceedings. pp. 35–50. Lecture Notes in Computer Science, Springer (2013). https://doi.org/10.1007/978-3-642-39634-2_6
15. Meng, J., Paulson, L.C.: Lightweight relevance filtering for machine-generated resolution problems. J. Appl. Log. **7**(1), 41–57 (2009). https://doi.org/10.1016/J.JAL.2007.07.004
16. Paulson, L.C., Nipkow, T., Wenzel, M.: From LCF to isabelle/hol. Formal Aspects Comput. **31**(6), 675–698 (2019). https://doi.org/10.1007/S00165-019-00492-1
17. Riazanov, A., Voronkov, A.: Limited resource strategy in resolution theorem proving. J. Symb. Comput. **36**(1-2), 101–115 (2003). https://doi.org/10.1016/S0747-7171(03)00040-3
18. Schulz, S.: E - a brainiac theorem prover. AI Commun. **15**(2-3), 111–126 (2002)
19. Schulz, S.: System description: E 1.8. In: McMillan, K.L., Middeldorp, A., Voronkov, A. (eds.) Logic for Programming, Artificial Intelligence, and Reasoning - 19th International Conference, LPAR-19, Stellenbosch, South Africa, December 14-19, 2013. Proceedings. pp. 735–743. Lecture Notes in Computer Science, Springer (2013). https://doi.org/10.1007/978-3-642-45221-5_49
20. Schulz, S., Cruanes, S., Vukmirovic, P.: Faster, higher, stronger: E 2.3. In: Fontaine, P. (ed.) Automated Deduction - CADE 27 - 27th International Conference on Automated Deduction, Natal, Brazil, August 27-30, 2019, Proceedings. pp. 495–507. Lecture Notes in Computer Science, Springer (2019). https://doi.org/10.1007/978-3-030-29436-6_29
21. Suda, M.: Vampire with a brain is a good ITP hammer. In: Konev, B., Reger, G. (eds.) Frontiers of Combining Systems - 13th International Symposium, FroCoS 2021, Birmingham, UK, September 8-10, 2021, Proceedings. pp. 192–209. Lecture Notes in Computer Science, Springer (2021). https://doi.org/10.1007/978-3-030-86205-3_11
22. Suda, M.: Deepire II = RL(GNN+2RvNN). In: 10th Conference on Artificial Intelligence and Theorem Proving AITP 2025 – proceedings (2025), https://aitp-conference.org/2025/abstract/AITP_2025_paper_19.pdf
23. Suda, M.: Efficient neural clause-selection reinforcement. In: Barrett, C.W., Waldmann, U. (eds.) Automated Deduction - CADE 30 - 30th International Conference on Automated Deduction, Stuttgart, Germany, July 28-31, 2025,

Proceedings. pp. 403–422. LNCS, Springer (2025). https://doi.org/10.1007/978-3-031-99984-0_22
24. Suda, M.: Efficient neural clause-selection reinforcement. CoRR **abs/2503.07792** (2025). https://doi.org/10.48550/ARXIV.2503.07792
25. Sutcliffe, G., Benzmüller, C.: Automated reasoning in higher-order logic using the TPTP THF infrastructure. J. Formaliz. Reason. **3**(1), 1–27 (2010). https://doi.org/10.6092/ISSN.1972-5787/1710, https://doi.org/10.6092/issn.1972-5787/1710
26. Sutcliffe, G., Schulz, S., Claessen, K., Baumgartner, P.: The TPTP typed first-order form with arithmetic. In: Bjørner, N.S., Voronkov, A. (eds.) Logic for Programming, Artificial Intelligence, and Reasoning - 18th International Conference, LPAR-18, Mérida, Venezuela, March 11-15, 2012. Proceedings. pp. 406–419. Lecture Notes in Computer Science, Springer (2012). https://doi.org/10.1007/978-3-642-28717-6_32, https://doi.org/10.1007/978-3-642-28717-6_32
27. Vukmirovic, P., Blanchette, J., Schulz, S.: Extending a high-performance prover to higher-order logic. In: Sankaranarayanan, S., Sharygina, N. (eds.) Tools and Algorithms for the Construction and Analysis of Systems - 29th International Conference, TACAS 2023, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2022, Paris, France, April 22-27, 2023, Proceedings, Part II. pp. 111–129. Lecture Notes in Computer Science, Springer (2023). https://doi.org/10.1007/978-3-031-30820-8_10

A Additional Resources

Additional resources for λE and λENIGMA:

- λE with λENIGMA support:
  `https://github.com/ai4reason/eprover/tree/enigma`
- λENIGMA training infrastructure:
  `https://github.com/cbboyan/solverpy` (package `solverpy_learn`)
- Bug reports and fixes discovered in λE on Isabelle export:
  `https://github.com/ai4reason/eprover/tree/enigma/bugs`
  See files `bug00N-*.md` for $\texttt{N} \in \{1, 2, 3, 4\}$.

Additional resource for Vampire and Deepire–2.0–HO:

- Vampire with HO support: developed by Bhayat as a fork of `master` in 2022: `https://github.com/vprover/vampire/tree/hol`
- Vampire with Deepire–2.0–HO support:
  `https://github.com/vprover/vampire/tree/mtpa-gnn-hol`
- Deepire–2.0–HO training infrastructure:
  `https://github.com/quickbeam123/lawa/tree/mtpa-gnn-hol`

B λE and λENIGMA Strategies

The following are the strategies used for λENIGMA experiments on TH0 and TF0 variants of the Isabelle export. The strategies are presented in the form of λE's command line options. These were experimentally evaluated as the best strategies on each fragment from the λE's strategy portfolio. Notably, they differ only in the `--fool-unroll` option.

```
TF0 (new_bool_6 from E 3.2)
-tKBO6 --order-weight-generation=precrank10 --order-precedence-generation=invfreq
--strong-rw-inst --order-constant-weight=1 --no-eq-unfolding --definitional-cnf=4
--presat-simplify --literal-selection-strategy=SelectMaxLComplexAvoidAppVar
--simul-paramod --forward-context-sr --forward-demod-level=1 --condense
--destructive-er-aggressive --strong-destructive-er --satcheck-proc-interval=5000
--satcheck=ConjMinMinFreq --delete-bad-limit=2000000000 --sos-uses-input-types
--neg-ext=all --pos-ext=all --ext-sup-max-depth=0 --lift-lambdas=false
--local-rw=true -H'(
   2*ConjectureRelativeSymbolWeight(PreferGround,0.5,100,100,100,100,1.5,1.5,1),
   6*ConjectureRelativeSymbolWeight(ByDerivationDepth,0.1,100,100,100,100,1.5,1.5,1.5),
   1*FIFOWeight(PreferProcessed),
   1*ConjectureRelativeSymbolWeight(PreferNonGoals,0.5,100,100,100,100,1.5,1.5,1),
   2*Refinedweight(PreferGoals,3,2,2,1.5,2))'

TH0 (new_bool_7 from E 3.2)
-tKBO6 --order-weight-generation=precrank10 --order-precedence-generation=invfreq
--strong-rw-inst --order-constant-weight=1 --no-eq-unfolding --definitional-cnf=4
--presat-simplify --literal-selection-strategy=SelectMaxLComplexAvoidAppVar
--simul-paramod --forward-context-sr --forward-demod-level=1 --condense
```

```
--destructive-er-aggressive --strong-destructive-er --satcheck-proc-interval=5000
--satcheck=ConjMinMinFreq --delete-bad-limit=2000000000 --sos-uses-input-types
--neg-ext=all --pos-ext=all --ext-sup-max-depth=0 --lift-lambdas=false
--local-rw=true --fool-unroll=false -H'(
   2*ConjectureRelativeSymbolWeight(PreferGround,0.5,100,100,100,100,1.5,1.5,1),
   6*ConjectureRelativeSymbolWeight(ByDerivationDepth,0.1,100,100,100,100,1.5,1.5,1.5),
   1*FIFOWeight(PreferProcessed),
   1*ConjectureRelativeSymbolWeight(PreferNonGoals,0.5,100,100,100,100,1.5,1.5,1),
   2*Refinedweight(PreferGoals,3,2,2,1.5,2))'
```

C Deepire 2.0 and Deepire–2.0–HO Strategies

Strategies selected on a smaller subset and with a smaller instruction limit. The list of the obtained strategies follows:

```
TF0 (vampire_rel_mtpa-gnn-2026_10730)
lrs-1010_4:1_anc=none:bce=on:drc=off:eape=off:er=filter:erml=4:fd=preordered:
  fde=unused:fgj=on:gtgl=2:kmz=on:kws=precedence:lcm=predicate:lwlo=on:
  nicw=on:nm=16:nwc=5.0:s2a=on:s2agt=32:s2at=1.5:sd=15:sfv=off:slsq=on:
  slsqr=1,4:spb=intro:to=kbo:urr=on_0

TH0 (vampire_rel_mtpa-gnn-hol_6946)
dis+1002_2:1_aac=none:acc=on:anc=none:au=on:bd=preordered:bsr=unit_only:
  cbe=off:cond=fast:drc=off:e2e=on:er=filter:fd=preordered:fde=unused:
  fe=axiom:fsr=off:gtgl=5:hfsq=on:hfsqc=3:hfsqr=4,1:nwc=5.0:pe=on:plsq=on:
  plsql=on:s2a=on:s2agt=32:s2at=2.0:sfv=off:spb=non_intro_0

TH1 (vampire_rel_mtpa-gnn-hol_6946)
lrs+1002_3:2_acc=on:add=off:au=on:av=off:avsq=on:avsqc=5:avsql=on:avsqr=2,1:
  bce=on:bd=preordered:bet=on:br=off:bsr=unit_only:cbe=off:cha=on:
  cnfonf=off:drc=off:e2e=on:er=known:fde=unused:fe=axiom:hfaw=3:
  hfsq=on:hfsqc=1:hfsqr=4,1:hud=1:kws=inv_frequency:nicw=on:nwc=5.0:
  piset=not:prag=on:s2a=on:s2agt=32:s2at=1.5:s2pl=on:sac=on:sfv=off:
  slsq=on:slsqc=2:slsql=off:slsqr=4,1:sp=unary_frequency_0
```

D Complementarity of λENIGMA and Deepire–2.0–HO

Complementarity of λENIGMA and Deepire–2.0–HO can be expressed by constructing the greedy cover sequence. The greedy cover of a portfolio (set of strategies) is constructed by starting from the best strategy from the portfolio, eliminating the problems it solves, and iterating until all problems are eliminated or strategies are exhausted. First, we consider the portfolio of the best λENIGMA and Deepire–2.0–HO strategies from all eval-train loop iterations for each fragment. The greedy cover is shown in Table 3. Restricting to λENIGMA strategies, we obtain Table 4. Restricting to Deepire–2.0–HO strategies, we obtain Table 5. Finally, we construct separate greedy covers on the HO and FO fragments in Table 6 and Table 7.

Table 3: Greedy cover of the best ENIGMA and Deepire strategies

prover	*fragment*	*solves*	*addon*		*total*	
Deepire	TF0	3229	+3229	−	= 3229	= 64.58%
ENIGMA	TH0	2670	+145	+4.49%	= 3374	= 67.48%
Deepire	TH1	2041	+66	+1.96%	= 3440	= 68.80%
Deepire	TH0	2561	+24	+0.70%	= 3464	= 69.28%
ENIGMA	TF0	2440	+16	+0.46%	= 3480	= 69.60%

Table 4: Greedy cover of the best ENIGMA strategies

prover	*fragment*	*solves*	*addon*		*total*	
ENIGMA	TH0	2670	+2670	−	= 2670	= 53.40%
ENIGMA	TF0	2440	+182	+6.82%	= 2852	= 57.04%

Table 5: Greedy cover of the best Deepire strategies

prover	*fragment*	*solves*	*addon*		*total*	
Deepire	TF0	3229	+3229	−	= 3229	= 64.58%
Deepire	TH1	2041	+130	+4.03%	= 3359	= 67.18%
Deepire	TH0	2561	+51	+1.52%	= 3410	= 68.20%

Table 6: Greedy cover restricted to HO fragments

prover	*fragment*	*solves*	*addon*		*total*	
ENIGMA	TH0	2670	+2670	−	= 2670	= 53.40%
Deepire	TH0	2561	+245	+9.18%	= 2915	= 58.30%
Deepire	TH1	2041	+103	+3.53%	= 3018	= 60.36%

Table 7: Greedy cover restricted to FO fragments

prover	*fragment*	*solves*	*addon*		*total*	
Deepire	TF0	3229	+3229	−	= 3229	= 64.58%
ENIGMA	TF0	2440	+40	+1.24%	= 3269	= 65.38%

We can observe that the Deepire strategy on TF0 is currently the best strategy, which by itself solves 64.58 % of the testing problems. This is complemented by the ENIGMA strategy on TH0, which adds up to 67.48 %, that is, a reasonable improvement of 4.49 % (from 3229 to 3374). Deepire-only strategies solve 3410 problems, which is 68.20 % of the testing problems. λENIGMA strategies add to 69.60 %. The performance on the HO fragment is only 60.36 % using both λENIGMA and Deepire–2.0–HO strategies. While the performance on the FO fragment is 65.38 %, we see that with the help of HO strategies, this is raised

to 69.60 %. Hence the contribution of HO strategies is already quite reasonable. Our future work will focus on improving the performance on the HO fragment.