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

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

Gothenburg CHAIR Workshop, 28th May, 2026

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

Outline

Automated Reasoning for Interactive Theorem Provers

- Interactive theorem provers (ITPs) such as Isabelle/HOL combine expressive logics with large formal libraries.

Automated Reasoning for Interactive Theorem Provers

- Interactive theorem provers (ITPs) such as Isabelle/HOL combine expressive logics with large formal libraries.
- Hammer systems (e.g. Sledgehammer) bridge ITPs and external automatic theorem provers (ATPs):
  - Select relevant facts from the library (premise selection).
  - Export the proof obligation to an ATP format.
  - Reconstruct successful ATP proofs inside the ITP.

Automated Reasoning for Interactive Theorem Provers

- Interactive theorem provers (ITPs) such as Isabelle/HOL combine expressive logics with large formal libraries.
- Hammer systems (e.g. Sledgehammer) bridge ITPs and external automatic theorem provers (ATPs):
  - Select relevant facts from the library (premise selection).
  - Export the proof obligation to an ATP format.
  - Reconstruct successful ATP proofs inside the ITP.
- Two complementary learning problems:
  - **Premise selection**: which facts to include?
  - **Clause guidance**: which clauses to explore inside the ATP?

Automated Reasoning for Interactive Theorem Provers

- Interactive theorem provers (ITPs) such as Isabelle/HOL combine expressive logics with large formal libraries.
- Hammer systems (e.g. Sledgehammer) bridge ITPs and external automatic theorem provers (ATPs):
  - Select relevant facts from the library (premise selection).
  - Export the proof obligation to an ATP format.
  - Reconstruct successful ATP proofs inside the ITP.
- Two complementary learning problems:
  - **Premise selection**: which facts to include?
  - **Clause guidance**: which clauses to explore inside the ATP?
- Both problems have been successfully tackled in the first-order (FO) setting.

Learning-Guided Clause Selection

- ENIGMA: learning-guided clause selection for E.
  - Gradient-boosted decision trees classify clauses as useful/useless.
  - Positive = clauses appearing in a proof; negative = others.

Learning-Guided Clause Selection

- ENIGMA: learning-guided clause selection for E.
  - Gradient-boosted decision trees classify clauses as useful/useless.
  - Positive = clauses appearing in a proof; negative = others.
- Deepire 2.0: neural clause selection guidance for Vampire.
  - Graph Neural Network (GNN) embeds the problem.
  - Recursive Neural Networks (RvNN) score derived clauses.

Learning-Guided Clause Selection

- ENIGMA: learning-guided clause selection for E.
  - Gradient-boosted decision trees classify clauses as useful/useless.
  - Positive = clauses appearing in a proof; negative = others.
- Deepire 2.0: neural clause selection guidance for Vampire.
  - Graph Neural Network (GNN) embeds the problem.
  - Recursive Neural Networks (RvNN) score derived clauses.
- Both are trained on large benchmarks and run inside the ATP's saturation loop.

Learning-Guided Clause Selection

- ENIGMA: learning-guided clause selection for E.
  - Gradient-boosted decision trees classify clauses as useful/useless.
  - Positive = clauses appearing in a proof; negative = others.
- Deepire 2.0: neural clause selection guidance for Vampire.
  - Graph Neural Network (GNN) embeds the problem.
  - Recursive Neural Networks (RvNN) score derived clauses.
- Both are trained on large benchmarks and run inside the ATP's saturation loop.
- This work: extend both to the higher-order (HO) setting using Isabelle/HOL exports.

Outline

Exporting Isabelle/HOL Proof Obligations

- We use Mirabelle to replay Isabelle/HOL developments and collect intermediate proof obligations.

Exporting Isabelle/HOL Proof Obligations

- We use Mirabelle to replay Isabelle/HOL developments and collect intermediate proof obligations.
- Sledgehammer exports each obligation in TPTP format:
  - TF0: monomorphic first-order. $[\mathsf{fun}_0 \to \mathsf{list}_0 \to \mathsf{list}_0]$
  - TH0: monomorphic higher-order. $[(i \to i) \to \mathsf{list}_i \to \mathsf{list}_i]$
  - TH1: *polymorphic* higher-order. $[\forall\alpha.(\alpha \to \alpha) \to \mathsf{list}\ \alpha \to \mathsf{list}\ \alpha]$

Exporting Isabelle/HOL Proof Obligations

- We use Mirabelle to replay Isabelle/HOL developments and collect intermediate proof obligations.
- Sledgehammer exports each obligation in TPTP format:
  - TF0: monomorphic first-order. $[\mathsf{fun}_0 \to \mathsf{list}_0 \to \mathsf{list}_0]$
  - TH0: monomorphic higher-order. $[(i \to i) \to \mathsf{list}_i \to \mathsf{list}_i]$
  - TH1: *polymorphic* higher-order. $[\forall\alpha.(\alpha \to \alpha) \to \mathsf{list}\ \alpha \to \mathsf{list}\ \alpha]$
- Based on the stable **Isabelle-2025-2** release.

Exporting Isabelle/HOL Proof Obligations

- We use Mirabelle to replay Isabelle/HOL developments and collect intermediate proof obligations.
- Sledgehammer exports each obligation in TPTP format:
  - TF0: monomorphic first-order. $[\mathsf{fun}_0 \to \mathsf{list}_0 \to \mathsf{list}_0]$
  - TH0: monomorphic higher-order. $[(i \to i) \to \mathsf{list}_i \to \mathsf{list}_i]$
  - TH1: *polymorphic* higher-order. $[\forall\alpha.(\alpha \to \alpha) \to \mathsf{list}\ \alpha \to \mathsf{list}\ \alpha]$
- Based on the stable **Isabelle-2025-2** release.
- MeSh premise filter; 512 facts per slice; no ITE/let/choice.

Exporting Isabelle/HOL Proof Obligations

- We use Mirabelle to replay Isabelle/HOL developments and collect intermediate proof obligations.
- Sledgehammer exports each obligation in TPTP format:
  - TF0: monomorphic first-order. $[\mathsf{fun}_0 \to \mathsf{list}_0 \to \mathsf{list}_0]$
  - TH0: monomorphic higher-order. $[(i \to i) \to \mathsf{list}_i \to \mathsf{list}_i]$
  - TH1: *polymorphic* higher-order. $[\forall\alpha.(\alpha \to \alpha) \to \mathsf{list}\ \alpha \to \mathsf{list}\ \alpha]$
- Based on the stable **Isabelle-2025-2** release.
- MeSh premise filter; 512 facts per slice; no ITE/let/choice.
- Total: 246,277 problems per dialect.

Bugs Found and Fixed

Running the export at scale revealed several issues:

- Name clash in η-expansion: TPTP generator produced type-invalid problems; affected many tasks in HOL-Analysis.

Bugs Found and Fixed

Running the export at scale revealed several issues:

- Name clash in η-expansion: TPTP generator produced type-invalid problems; affected many tasks in HOL-Analysis.
- HOL-Nominal `Class1`: clashes between names and conames.

Bugs Found and Fixed

Running the export at scale revealed several issues:

- Name clash in η-expansion: TPTP generator produced type-invalid problems; affected many tasks in HOL-Analysis.
- HOL-Nominal `Class1`: clashes between names and conames.
- Smaller monomorphisation issues in lower-logic encodings.

Bugs Found and Fixed

Running the export at scale revealed several issues:

- Name clash in η-expansion: TPTP generator produced type-invalid problems; affected many tasks in HOL-Analysis.
- HOL-Nominal `Class1`: clashes between names and conames.
- Smaller monomorphisation issues in lower-logic encodings.
- All fixed in the Isabelle development version.
- These fixes benefit *all* provers, not just ours.

Outline

Higher-Order Saturation Provers

Both E and Vampire use a given-clause saturation loop:

1. Select the best *given clause* from the unprocessed/passive set.
2. Apply inference rules to derive new clauses.
3. Repeat until refutation or saturation.

Higher-Order Saturation Provers

Both E and Vampire use a given-clause saturation loop:

1. Select the best *given clause* from the unprocessed/passive set.
2. Apply inference rules to derive new clauses.
3. Repeat until refutation or saturation.

- λE (E $\geq$ 3.0): HO extension supporting monomorphic TH0.
- Vampire hol branch: HO extension supporting polymorphic TH1.

Higher-Order Saturation Provers

Both E and Vampire use a given-clause saturation loop:

1. Select the best *given clause* from the unprocessed/passive set.
2. Apply inference rules to derive new clauses.
3. Repeat until refutation or saturation.

- λE (E $\geq$ 3.0): HO extension supporting monomorphic TH0.
- Vampire hol branch: HO extension supporting polymorphic TH1.
- Clause selection is the key decision point — target for learning.

λENIGMA: HO Features for ENIGMA

- In λE, HO applications (F X Y) are stored as @(F, X, Y) using an internal symbol @.
  λ-abstractions (λx.T) become λ(x, T).

λENIGMA: HO Features for ENIGMA

- In λE, HO applications (F X Y) are stored as @(F, X, Y) using an internal symbol @.
  λ-abstractions (λx.T) become λ(x, T).
- ENIGMA extracts vertical (top-down paths) and horizontal (head + argument heads) features from clause terms.

λENIGMA: HO Features for ENIGMA

- In λE, HO applications (F X Y) are stored as @(F, X, Y) using an internal symbol @.
  λ-abstractions (λx.T) become λ(x, T).
- ENIGMA extracts vertical (top-down paths) and horizontal (head + argument heads) features from clause terms.
- λENIGMA extensions:
  - @(F, X, Y): X, Y treated as children of F, not siblings: F(X,Y).
  - λ(x, T): variable name dropped; binder position marked.
  - Types embedded into features: arrow type (i→(i→i)→o) encoded as i-(i-i)-o and appended to symbol names.

λENIGMA: HO Features for ENIGMA

- In λE, HO applications (F X Y) are stored as @(F, X, Y) using an internal symbol @.
  λ-abstractions (λx.T) become λ(x, T).
- ENIGMA extracts vertical (top-down paths) and horizontal (head + argument heads) features from clause terms.
- λENIGMA extensions:
  - @(F, X, Y): X, Y treated as children of F, not siblings: F(X,Y).
  - λ(x, T): variable name dropped; binder position marked.
  - Types embedded into features: arrow type (i→(i→i)→o) encoded as i-(i-i)-o and appended to symbol names.
- Works on TF0 and TH0.

Deepire-2.0-HO: Neural Guidance for HO Vampire

- Deepire 2.0 architecture: GNN + RvNN (term structure) + RvNN (derivation history) + fully connected scoring network.

Deepire-2.0-HO: Neural Guidance for HO Vampire

- Deepire 2.0 architecture: GNN + RvNN (term structure) + RvNN (derivation history) + fully connected scoring network.
- Adapted to handle polymorphic HO problems:
  - New GNN node type for type constructors.
  - Sorts promoted to full expressions (with sort variables).
  - Distinct node labels for universal syntactic concepts: arrow type, apply, λ, de Bruijn indices, propositional connectives.

Deepire-2.0-HO: Neural Guidance for HO Vampire

- Deepire 2.0 architecture: GNN + RvNN (term structure) + RvNN (derivation history) + fully connected scoring network.
- Adapted to handle polymorphic HO problems:
  - New GNN node type for type constructors.
  - Sorts promoted to full expressions (with sort variables).
  - Distinct node labels for universal syntactic concepts: arrow type, apply, λ, de Bruijn indices, propositional connectives.
- Built on VAMPIRE's `hol` branch.
- Works on TF0, TH0, and TH1.

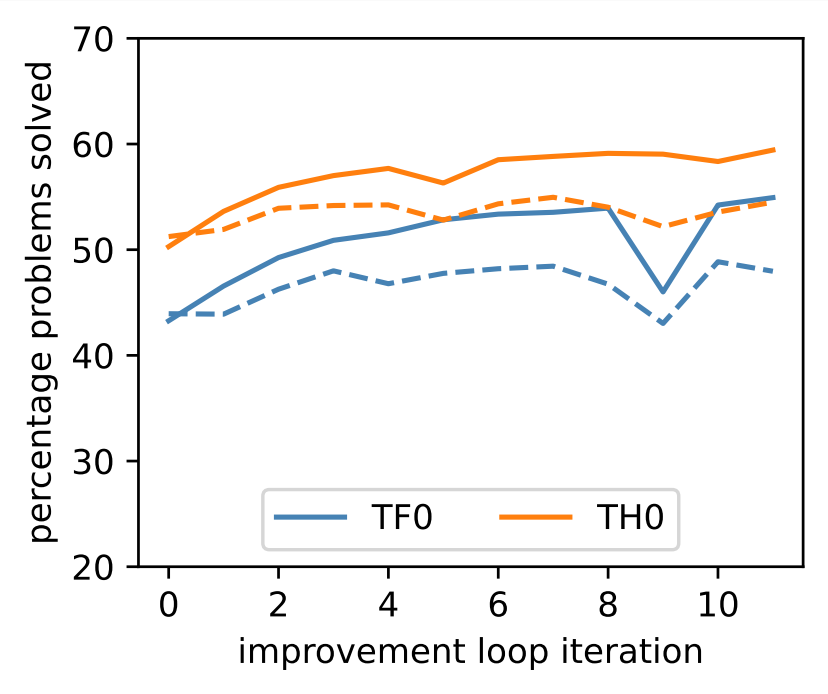
Outline

Experimental Setup

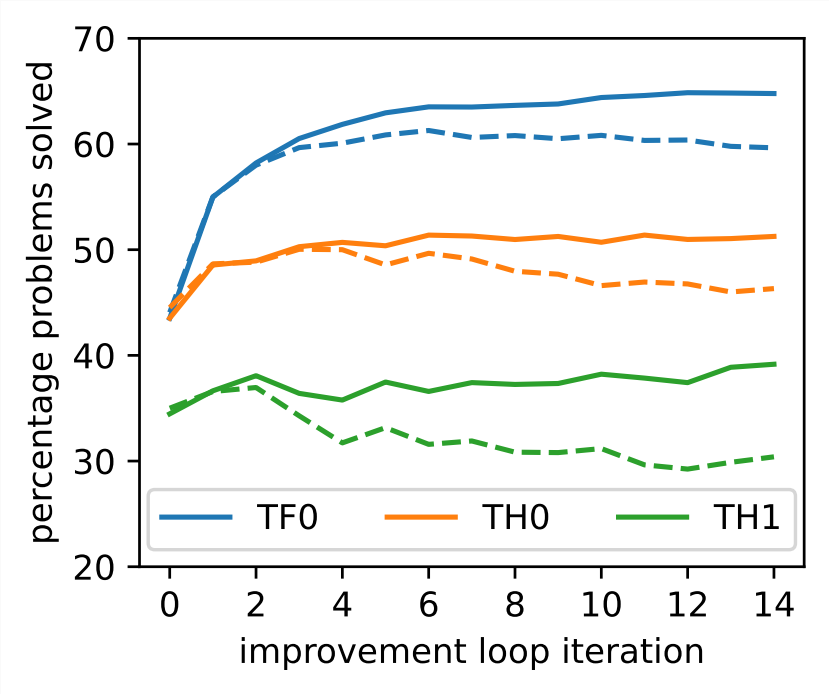
- 30,000 training problems; 5,000 test problems (aligned across dialects).
- λ**ENIGMA**:
  - 10 training iterations; $\sim$12 h per iteration.
  - 10 s per problem.
  - Evaluated on TF0 and TH0.
- **Deepire-2.0-HO**:
  - 10 training iterations; $\sim$4.5 days per dialect.
  - 16 billion CPU instructions ($\sim$6 s).
  - Evaluated on TF0, TH0, and TH1.

Training Plots: λENIGMA and Deepire-2.0-HO

left = λENIGMA



right = Deepire-2.0-HO

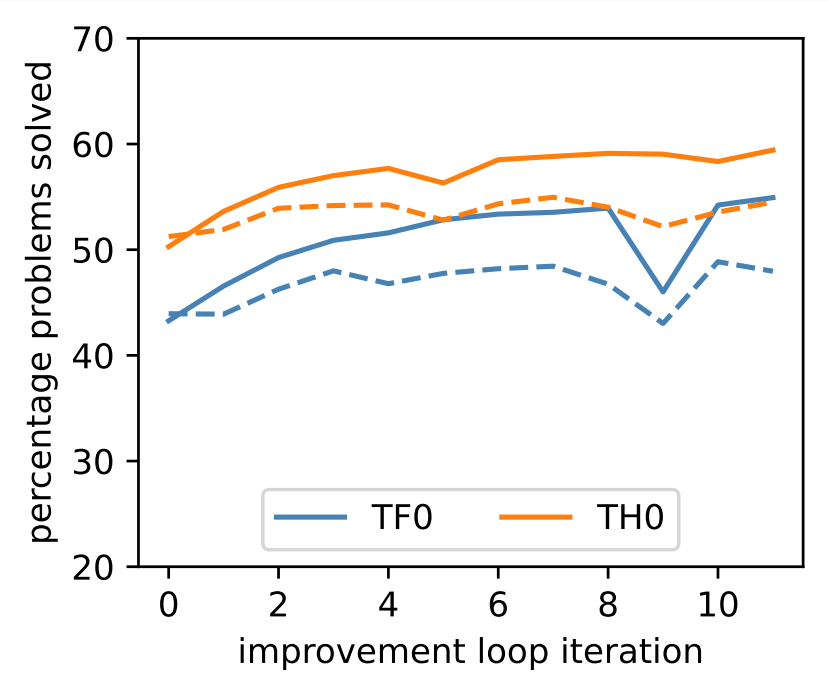


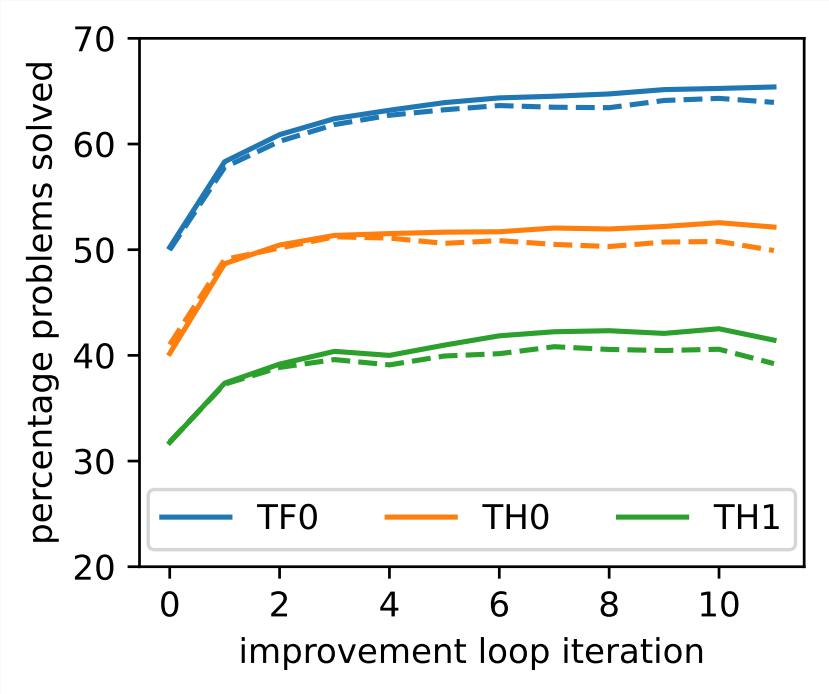
Solid = train, dashed = test. X-axis: training iteration (0 = baseline).

Training Plots: λENIGMA and Deepire-2.0-HO

left = λENIGMA

right = Deepire-2.0-HO





Solid = train, dashed = test. X-axis: training iteration (0 = baseline).

Results: λENIGMA

- λE baseline on TH0 outperforms TF0 — and even VAMPIRE.
- λENIGMA improves consistently on both fragments:
  - +7.26 % on TH0 (test).
  - +11.2 % on TF0 (test).
- Occasional dips (e.g. iteration 5 for TF0) recover; train and test curves remain correlated.

Results: Deepire-2.0-HO

- Vampire + Deepire 2.0 on TF0 is still the strongest single dialect: ∼65 % test coverage.
- Deepire-2.0-HO brings comparable relative gains on HO fragments:
  - TH0: 40.2 % → 51.2 % (+27 % relative).
  - TH1: 31.8 % → 40.8 % (+28 % relative).
- HO starting points are lower than FO — understanding why is an open research question.

Complementarity of Strategies

All five best strategies (2 FO + 3 HO) evaluated on the 5,000 test problems:

System	*Fragment*	*Solves*	*Total*	*Cumulative*
Deepire 2.0	TF0	3064	3064	61.28 %
λENIGMA	TH0	2748	3329	66.58 %
Deepire-2.0-HO	TH1	1848	3377	67.54 %
Deepire-2.0-HO	TH0	2502	3404	68.08 %
λENIGMA	TF0	2443	**3425**	**68.50** %

- 3 HO strategies alone: 60.92 %
- 2 FO strategies alone: 62.58 %
- All five together: 68.50 %.
- HO strategies add +5.92 % over FO alone.

Outline

Summary

Contributions:

- λENIGMA: first learning-guided HO clause guidance for λE (TH0 + TF0 from Isabelle).
- Deepire-2.0-HO: neural HO clause guidance for Vampire (TH0 + TH1).
- Improved Sledgehammer export: fixed bugs in HOL-Analysis and HOL-Nominal; improved monomorphisation.
- Consistent improvement over baselines; HO strategies are complementary to FO ones.

Future Work

- Why does native TH1 solving currently lag behind TH0?
  $\Rightarrow$ Investigate calculus, encoding, and strategy differences.
- Use HO premise selection to improve TH1 training data quality.
- Scale training to the full 246,277 problems
  (currently too slow; GPU training would help).
- Closer integration between λENIGMA and Deepire-2.0-HO strategies.

Future Work

- Why does native TH1 solving currently lag behind TH0?
  $\Rightarrow$ Investigate calculus, encoding, and strategy differences.
- Use HO premise selection to improve TH1 training data quality.
- Scale training to the full 246,277 problems
  (currently too slow; GPU training would help).
- Closer integration between λENIGMA and Deepire-2.0-HO strategies.

Thank you!