Isabelle/RL Project Proposal: Reinforcement learning on the Isabelle proof assistant

Jonathan Julián Huerta y Munive
huertjon@cvut.cz
Czech Technical University in Prague
September, 2024

CZECH INSTITUTE OF INFORMATICS ROBOTICS AND CYBERNETICS CTU IN PRAGUE

Funded by the European Union

Funded by the European Union. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union. The European Union cannot be held responsible for them.

Summary

The AITP conference is the forum for discussing how to get to our inevitable future of large-scale semantic processing and strong computer assistance of mathematics and science

Discussion points:

- Isabelle is one of the most developed interactive theorem provers (ITPs)
- Machine learning (ML) methods have been successfully applied to other ITPs
- Many users are still not benefiting from tool integrations of ML and ITPs
- Reinforcement learning (RL) approaches with conjecturing have not been sufficiently attempted in these integrations
- There is still much to learn with respect to ML and ITPs
- We are currently working towards filling these gaps
- This is an open invitation to contribute to this project

Overview of the
Isabelle proof assistant

An Isabelle apply-style proof

Tactics for backward reasoning from goal to hypothesis
(tactics inputs: terms and premises)

```
lemma
  fixes P :: "('a::finite) ⇒ nat ⇒ bool"
  assumes "∀i. ∃N::nat. ∀n ≥ N. P i n"
  shows "∃N. ∀i. ∀n ≥ N. P i n"
  apply (rule_tac x="Max {Inf {N. ∀n≥N. P i n} |i::'a. i ∈ UNIV}" in exI)
  apply clarify
  subgoal for i n
    using wellorder_Inf_lemma[of "λn. P i n"] assms
      Max_ge[of "{Inf {N. ∀n≥N. P i n} |i::'a. i ∈ UNIV}", OF finite_image]
    apply -
    apply (erule_tac x=i in allE)
    apply (subgoal_tac "∀n≥Inf {N. ∀n≥N. P i n}. P i n")
```

```
proof (prove)
goal (1 subgoal):
 1. ∃N. ∀i n. N ≤ n ⟶ P i n
```

```
proof (prove)
goal (1 subgoal):
 1. ∀i n. Max {Inf {N. ∀n≥N. P i n} |i. i ∈ UNIV} ≤ n ⟶ P i n
```

```
proof (prove)
goal (1 subgoal):
 1. ⋀i n. Max {Inf {N. ∀n≥N. P i n} |i. i ∈ UNIV} ≤ n ⟹ P i n
```

```
proof (prove)
goal (1 subgoal):
 1. Max {Inf {N. ∀n≥N. P i n} |i. i ∈ UNIV} ≤ n ⟹ P i n
```

Hammers

Calls to automated theorem provers (ATPs) to reduce the number of premises needed to automatically prove the current proof obligation

```
lemma
  fixes P :: "('a::finite) ⇒ nat ⇒ bool"
  assumes "∀i. ∃N::nat. ∀n ≥ N. P i n"
  shows "∃N. ∀i. ∀n ≥ N. P i n"
  apply (rule_tac x="Max {Inf {N. ∀n≥N. P i n} |i::'a. i ∈ UNIV}" in exI)
  apply clarify
  subgoal for i n
    using wellorder_Inf_lemma[of "λn. P i n"] assms
      Max_ge[of "{Inf {N. ∀n≥N. P i n} |i::'a. i ∈ UNIV}", OF finite_image]
    apply -
    apply (erule_tac x=i in allE)
    apply (subgoal_tac "∀n≥Inf {N. ∀n≥N. P i n}. P i n")
    sledgehammer
```

```
proof (prove)
goal (1 subgoal):
 1. ∃N. ∀i n. N ≤ n ⟶ P i n
```

```
Sledgehammering...
vampire found a proof...
spass found a proof...
vampire: Try this: apply (metis (mono_tags, lifting) UNIV_I dual_order.trans mem_Collect_eq) (90 ms)
spass: Try this: apply (metis (mono_tags, lifting) iso_tuple_UNIV_I mem_Collect_eq order_trans) (40 ms)
Done
```

An Isabelle apply-style proof

Iterate based on feedback and finish the proof

```
lemma
  fixes P :: "('a::finite) ⇒ nat ⇒ bool"
  assumes "∀i. ∃N::nat. ∀n ≥ N. P i n"
  shows "∃N. ∀i. ∀n ≥ N. P i n"
  apply (rule_tac x="Max {Inf {N. ∀n≥N. P i n} |i::'a. i ∈ UNIV}" in exI)
  apply clarify
  subgoal for i n
    using wellorder_Inf_lemma[of "λn. P i n"] assms
      Max_ge[of "{Inf {N. ∀n≥N. P i n} |i::'a. i ∈ UNIV}", OF finite_image]
    apply -
    apply (erule_tac x=i in allE)
    apply (subgoal_tac "∀n≥Inf {N. ∀n≥N. P i n}. P i n")
    apply (metis (mono_tags, lifting) dual_order.trans iso_tuple_UNIV_I mem_Collect_eq)
    apply blast
    done
  done
```

```
proof (prove)
goal (1 subgoal):
 1. ∃N. ∀i n. N ≤ n ⟶ P i n
```

```
proof (prove)
goal (1 subgoal):
 1. ∀i n. Max {Inf {N. ∀n≥N. P i n} |i. i ∈ UNIV} ≤ n ⟶ P i n
```

```
proof (prove)
goal (1 subgoal):
 1. ⋀i n. Max {Inf {N. ∀n≥N. P i n} |i. i ∈ UNIV} ≤ n ⟹ P i n
```

```
proof (prove)
goal (1 subgoal):
 1. Max {Inf {N. ∀n≥N. P i n} |i. i ∈ UNIV} ≤ n ⟹ P i n
```

```
proof (prove)
goal:
No subgoals!
```

An Isabelle Isar-style Proof

- More human readable proofs
- Standard in the Archive of Formal Proofs (AFP)

```
lemma
  fixes P :: "('a::finite) ⇒ nat ⇒ bool"
  assumes "∀i. ∃N::nat. ∀n ≥ N. P i n"
  shows "∃N. ∀i. ∀n ≥ N. P i n"
proof-
  let "?bound i" = "Inf {N . ∀n ≥ N. P i n}"
  let ?N = "Max {?bound i |i. i ∈ UNIV}"
  {fix n::nat and i::'a
    from assms
    obtain M where "∀n ≥ M. P i n"
      \<proof>
    hence obs:"∀ m ≥ ?bound i. P i m"
      \<proof>
    assume "n ≥ ?N"
    also have "?N ≥ ?bound i"
      \<proof>
    ultimately have "n ≥ ?bound i"
      \<proof>
    hence "P i n"
      \<proof>
  }
  thus "∃N. ∀i n. N ≤ n ⟶ P i n"
    \<proof>
qed
```

```
proof (prove)
goal (1 subgoal):
 1. ∃N. ∀i n. N ≤ n ⟶ P i n
```

```
proof (prove)
using this:
  ∀i. ∃N. ∀n≥N. P i n

goal (1 subgoal):
 1. (⋀M. ∀n≥M. P i n ⟹ thesis) ⟹ thesis
```

```
proof (prove)
using this:
  ∀n≥M. P i n

goal (1 subgoal):
 1. ∀m≥Inf {N. ∀n≥N. P i n}. P i m
```

```
proof (state)
this:
  Max {Inf {N. ∀n≥N. P i n} |i. i ∈ UNIV} ≤ ?n2 ⟹ P ?i2 ?n2

goal (1 subgoal):
 1. ∃N. ∀i n. N ≤ n ⟶ P i n
```

An Isabelle Isar-style Proof

User completes apply-style intermediate steps

```
lemma
  fixes P :: "('a::finite) ⇒ nat ⇒ bool"
  assumes "∀i. ∃N::nat. ∀n ≥ N. P i n"
  shows "∃N. ∀i. ∀n ≥ N. P i n"
proof-
  let "?bound i" = "Inf {N . ∀n ≥ N. P i n}"
  let ?N = "Max {?bound i |i. i ∈ UNIV}"
  {fix n::nat and i::'a
    from assms
    obtain M where "∀n ≥ M. P i n"
      by blast
    hence obs:"∀ m ≥ ?bound i. P i m"
      using wellorder_Inf_lemma[of "λn. P i n"]
      by blast
    assume "n ≥ ?N"
    also have "?N ≥ ?bound i"
      using finite_image
      by(fastforce intro: Max_ge)
    ultimately have "n ≥ ?bound i"
      using order.trans
      by blast
    hence "P i n"
      using obs
      by blast
  }
  thus "∃N. ∀i n. N ≤ n ⟶ P i n"
    by blast
qed
```

```
proof (prove)
goal (1 subgoal):
 1. ∃N. ∀i n. N ≤ n ⟶ P i n
```

```
proof (prove)
using this:
  ∀i. ∃N. ∀n≥N. P i n

goal (1 subgoal):
 1. (⋀M. ∀n≥M. P i n ⟹ thesis) ⟹ thesis
```

```
proof (prove)
using this:
  ∀n≥M. P i n

goal (1 subgoal):
 1. ∀m≥Inf {N. ∀n≥N. P i n}. P i m
```

```
proof (prove)
using this:
  • ∀n≥M. P i n
  • ∃N. ∀n≥N. P i n ⟹ P i (Inf {N. ∀n≥N. P i n})
  • ∃N. ∀n≥N. P i n ⟹ ∀n≥Inf {N. ∀n≥N. P i n}. P i n

goal (1 subgoal):
 1. ∀m≥Inf {N. ∀n≥N. P i n}. P i m
```

```
proof (state)
this:
  Max {Inf {N. ∀n≥N. P i n} |i. i ∈ UNIV} ≤ ?n2 ⟹ P ?i2 ?n2

goal (1 subgoal):
 1. ∃N. ∀i n. N ≤ n ⟶ P i n
```

```
goal:
No subgoals!
```

Previous machine learning approaches on interactive theorem provers

Previous ML approaches on ITPs: TacticToe

Uses k-nearest neighbours (kNN) and Monte-Carlo tree search (MCTS) to predict the next proof-step (proves 66.4% of HOL4 standard library)

```
lemma
  fixes P :: "('a::finite) ⇒ nat ⇒ bool"
  assumes "∀i. ∃N::nat. ∀n ≥ N. P i n"
  shows "∃N. ∀i. ∀n ≥ N. P i n"
  apply (rule_tac x="Max {Inf {N. ∀n≥N. P i n} |i::'a. i ∈ UNIV}" in exI)
  apply clarify
  subgoal for i n
    using wellorder_Inf_lemma[of "λn. P i n"] assms
      Max_ge[of "{Inf {N. ∀n≥N. P i n} |i::'a. i ∈ UNIV}", OF finite_image]
    apply -
    apply (erule_tac x=i in allE)
    apply (subgoal_tac "∀n≥Inf {N. ∀n≥N. P i n}. P i n")
    apply (metis (mono_tags, lifting) dual_order.trans iso_tuple_UNIV_I mem_Collect_eq)
    apply blast
    done
  done
```

```
proof (prove)
goal (1 subgoal):
 1. ⋀i n. Max {Inf {N. ∀n≥N. P i n} |i. i ∈ UNIV} ≤ n ⟹ P i n
```
$\equiv \varphi$

$d(\varphi_1, \varphi_2)$

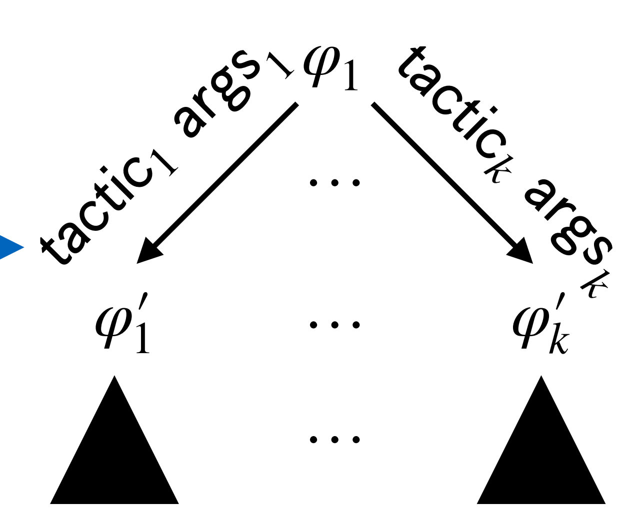
$d(\text{tactic}_1, \text{tactic}_2)$

$d(\text{args}_1, \text{args}_2)$

kNN

Tactic	Goal
clarify	φ_1
blast	φ_2
auto	φ_3

MCTS



Previous ML approaches on ITPs: “Draft, Sketch, Prove” vs Magnushammer

Isabelle

```
lemma
  fixes P :: "('a::finite) ⇒ nat ⇒ bool"
  assumes "∀i. ∃N::nat. ∀n ≥ N. P i n"
  shows "∃N. ∀i. ∀n ≥ N. P i n"
proof-
  let "?bound i" = "Inf {N . ∀n ≥ N. P i n}"
  let ?N = "Max {?bound i |i. i ∈ UNIV}"
  {fix n::nat and i::'a
    from assms
    obtain M where "∀n ≥ M. P i n"
      \<proof>
    hence obs:"∀ m ≥ ?bound i. P i m"
      \<proof>
    assume "n ≥ ?N"
    also have "?N ≥ ?bound i"
      \<proof>
    ultimately have "n ≥ ?bound i"
      \<proof>
    hence "P i n"
      \<proof>
  }
  thus "∃N. ∀i n. N ≤ n ⟶ P i n"
    \<proof>
qed
```

Formal proof sketching

Informal to LaTeX

LaTeX

Call hammers to complete the proofs

Since ATPs only serve for premise selection, replace them with neural networks (NNs)

Combined approach: Proof-rate of 71% on publicly available benchmark of 183K Isabelle proofs from the AFP

Other previous ML approaches on ITPs

- **TacticZero for HOL4:** It is an RL approach using NNs. It achieves ~49% success on a test set of 268 problems.

- **Tactician for Coq:** It is similar to TacticToe but extends the approach with a graph neural network (GNN) and compares it with the kNN. It also uses an LLM that is not context aware. Intersection between kNN and GNN is thin. GNN almost fully covers CoqHammer's results while proving some that CoqHammer does not prove.

- **HOL-list environment for HOL Light:** It is a light-weight tool for testing different approaches for tactic and premise selection. It has tested GNNs and an RL algorithm. The RL approach solves 56.3% of a set of 3225 theorems of the HOL Light's mathematical library.

- **Abduction prover, smart induct, and SeLFiE for Isabelle:** Combinations of proof strategy languages and conjecturing heuristics specialised to induction proofs, where hammers are weaker.

- **LLMs for Lean:** Many recent articles on using LLMs for autoformalization, premise selection, tactic suggestion, and proof search.

Questions

Questions after reading the literature

- TacticToe for HOL4
- TacticZero for HOL4 → *How do the kNN and the NN-based RL approaches compare?*
- Tactician for Coq → *Does context-awareness improve the LLM results?*
- HOL-list environment for HOL Light → *How does it compare with LLM-approaches?*
- Abduction prover for Isabelle → *How does it compare with ML approaches?*
- Magnushammer for Isabelle → *Can it be integrated as a tool into Isabelle? Does context-awareness improve the results?*
- LLMs for Lean (at least three research groups) → *Do any of these methods perform well on new problems?*

Observations:

- Isabelle is the only one that does not have an ML-focused integration
- Only the Tactician and Abduction prover seem to conjecture
- Only the Tactician offers comparisons of different ML models AND comparisons to hammers

Progress

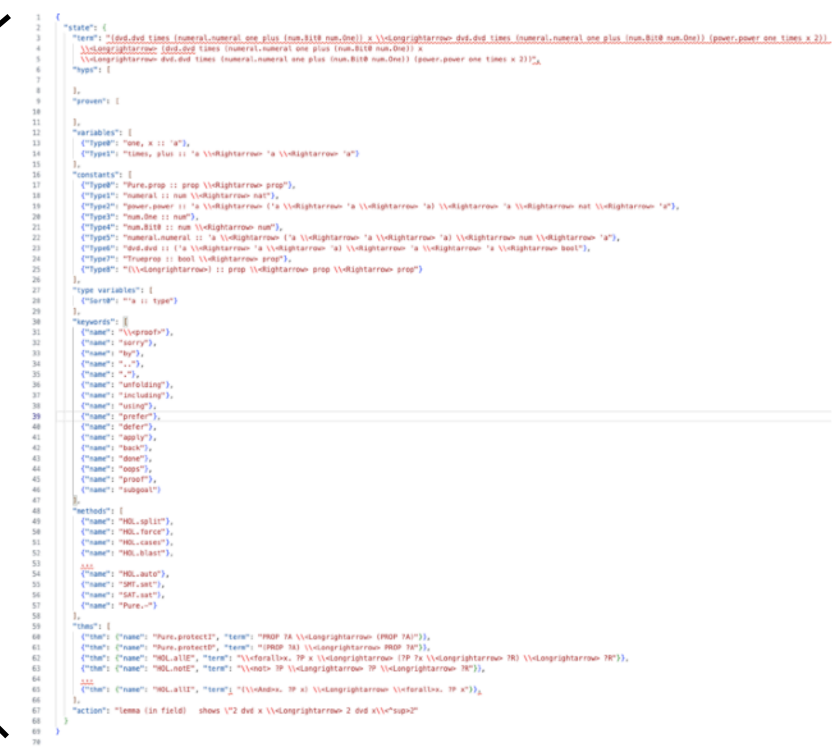
Project's current outputs

```
(*  Comment with keywords theory, begin, lemma, and by  *)

chapter ‹ Title with keywords theory, begin, lemma, apply and by ›

theory Test_Thy3
  — ‹ A comment with keywords theory, begin, lemma, apply and by ›
  imports Complex_Main

begin

—‹Automatic proofs›

lemma "∀ x. P x ⟹ ∀ x. P x ⟶ Q x ⟹ ∀ x. Q x"
  by auto

—‹Apply-style proofs (with rules)›

lemma "∀ x. P x ⟹ ∀ x. P x ⟶ Q x ⟹ ∀ x. Q x"
  apply(rule allI)
  apply (erule allE)
  apply (erule allE)
  apply (erule impE)
   apply assumption
  apply assumption
  done

—‹Math-like (or structured) proofs›

lemma (in field)
  shows "2 dvd x ⟹ 2 dvd x²"
proof-
  assume "2 dvd x"
  then obtain k where "x = 2 * k"
    by blast
  hence "x² = (2 * k)²"
    by simp
  also have "... = 4 * k²"
    using power_mult_distrib by force
  ultimately have "∃ k. x² = 2 * k"
    by (metis semiring_normalization_rules(18) semiring_normalization_rules(29))
  thus "2 dvd x²"
    by auto
qed

—‹ Theorem commands ›

theorem "∀ x. P x ⟹ ∀ x. P x ⟶ Q x ⟹ ∀ x. Q x"
  by blast

corollary "∀ x. P x ⟹ ∀ x. P x ⟶ Q x ⟹ ∀ x. Q x"
  by blast

proposition "∀ x. P x ⟹ ∀ x. P x ⟶ Q x ⟹ ∀ x. Q x"
  by blast

lemma named_theorem1: "∀ x. P x ⟹ ∀ x. P x ⟶ Q x ⟹ ∀ x. Q x"
  by blast

lemma named_theorem2: "∀ x. R x ⟹ ∀ x. R x ⟶ Q x ⟹ ∀ x. Q x"
  by (rule named_theorem1)

lemma named_theorem3:
  assumes h1: "x < c"
    and h2: "x < c ⟶ S x"
  shows "S x"
  using h1 and h2 by blast
```

1. Identifies each proof in a .thy file

```
lemma (in field)
  shows "2 dvd x ⟹ 2 dvd x²"
proof-
  assume "2 dvd x"
  then obtain k where "x = 2 * k"
    by blast
  hence "x² = (2 * k)²"
    by simp
  also have "... = 4 * k²"
    using power_mult_distrib by force
  ultimately have "∃ k. x² = 2 * k"
    by (metis semiring_normalization_rules(18)
      semiring_normalization_rules(29))
  thus "2 dvd x²"
    by auto
qed
```

3. Can output JSON object for each Data.T



2. Creates data (type Data.T) for each Isabelle transition (**command** + args) in each proof

Data generation example output as JSON attributes

```
"term": "(dvd.dvd times (numeral.numeral one plus (num.Bit0 num.One)) x
  \\<Longrightarrow> dvd.dvd times (numeral.numeral one plus (num.Bit0 num.One)) (power.power one times x 2))
  \\<Longrightarrow> (dvd.dvd times (numeral.numeral one plus (num.Bit0 num.One)) x
  \\<Longrightarrow> dvd.dvd times (numeral.numeral one plus (num.Bit0 num.One)) (power.power one times x 2))",
```

```
"thms": [
  {"thm": {"name": "Pure.protectI", "term": "PROP ?A \\<Longrightarrow> (PROP ?A)"}},
  {"thm": {"name": "Pure.protectD", "term": "(PROP ?A) \\<Longrightarrow> PROP ?A"}},
  {"thm": {"name": "HOL.allE", "term": "\\<forall>x. ?P x \\<Longrightarrow> (?P ?x \\<Longrightarrow> ?R) \\<Longrightarrow> ?R"}},
  {"thm": {"name": "HOL.notE", "term": "\\<not> ?P \\<Longrightarrow> ?P \\<Longrightarrow> ?R"}},
  ...
  {"thm": {"name": "HOL.allI", "term": "(\\<And>x. ?P x) \\<Longrightarrow> \\<forall>x. ?P x"}},
],
```

```
"keywords": [
  {"name": "\\<proof>"},
  {"name": "sorry"},
  {"name": "by"},
  {"name": ".."},
  {"name": "."},
  {"name": "unfolding"},
  {"name": "including"},
  {"name": "using"},
  {"name": "prefer"},
  {"name": "defer"},
  {"name": "apply"},
  {"name": "back"},
  {"name": "done"},
  {"name": "oops"},
  {"name": "proof"},
  {"name": "subgoal"}
],
```

- More lemmas found per .thy file than in previous Isabelle-focused approaches
- More context-aware data generation: syntax, keywords, methods and theorems are user-extensible, and the data-generation accounts for that

```
"methods": [
  {"name": "HOL.split"},
  {"name": "HOL.force"},
  {"name": "HOL.cases"},
  {"name": "HOL.blast"},
  ...
  {"name": "HOL.auto"},
  {"name": "SMT.smt"},
  {"name": "SAT.sat"},
  {"name": "Pure.-"}
],
```

Next steps

Next steps

- Currently working on retrieving Data.T in Scala (we can already retrieve Isabelle/ML terms t:term, and we have already accessed Isabelle functions in Scala from Python)
- Reproduce premise selection, tactic suggestion and proof-search results
- Use these results for generating “conjecturing” data
- Combine these approaches into various RL algorithms, similar to those used for playing Go or StarCraft, and compare them
- Create an Isabelle tool suggesting “next steps” and/or “complete (legible) proofs” for a given proof obligation

Conclusion

There are **many things to learn** from integrations of ITPs and machine learning methods. This is an **open invitation** to Isabelle or machine learning experts to collaborate on the implementation of the project or to provide constructive feedback that accelerates its completion.

Thanks!

- Isabelle is one of the most developed interactive theorem provers (ITPs)
- Machine learning (ML) methods have been successfully applied to other ITPs
- Many users are still not benefiting from tool integrations of ML and ITPs
- Reinforcement learning (RL) approaches with conjecturing have not been sufficiently attempted in these integrations
- There is still much to learn with respect to ML and ITPs
- We are currently working towards filling these gaps
- This is an open invitation to contribute to this project

Jonathan Julián Huerta y Munive
huertjon@cvut.cz
Czech Technical University in Prague
September, 2024