Isabelle/RL progress report: reaping the first fruits of the project

Jonathan Julián Huerta y Munive
huertjon@cvut.cz
Czech Technical University in Prague
April, 2025

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.

General Takeaway

Me: Come here as a user of interactive theorem provers (ITPs)

DeepIsaHOL Objective: create tools that serve the Machine Learning (ML) and the ITP community for advancing the state of the art in ML and ITP

Let's collaborate!

Example

Chengsong Tan

Hi Jonathan,

Thanks very much for the reply!

The problem I am faced with is automatic repair of proofs. Say we have

Jonathan Julian Huerta y Munive

writing my own project (https://github.com/yonoteam/DeepIsaHOL/) but I have not documented how to set it up. I can read a theory, and find the places where an apply-style-proof is made (all with ML code). Both my

⋮

Accepted at FomaliSE2025:

The Burden of Proof: Automated Tooling for Rapid Iteration on Large Mechanised Proofs

Chengsong Tan[*†], Alastair F. Donaldson[*], Jonathan Julián Huerta y Munive[‡], and John Wickerson[*]
[*]Imperial College London, UK. Email: {c.tan, alastair.donaldson, j.wickerson}@imperial.ac.uk
[†]Kaihong, China. Email: tanchengsong@kaihong.com
[‡]Czech Technical University, Czech Republic. Email: huertjon@cvut.cz

Summary of this talk

- The plan is to showcase contributions with lessons learned and views to the future:
  1. Data extraction algorithm
  2. Training and evaluation loops for flan-T5 and Gemma
  3. Read-eval-print loops (REPL) for interacting with Isabelle
  4. Depth-first search (DFS) evaluation algorithm
  5. Small scale experiment comparing training set ups
  6. Proof fixing tool for large developments
  7. Tooling for calling an external next-step predictor from Isabelle

Approximate model of Isabelle

An Isabelle apply-style proof

User action: a command and its arguments

```
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
```

User states: the output the user sees in-between actions

```
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
```

An Isabelle Isar-style 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"
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
```

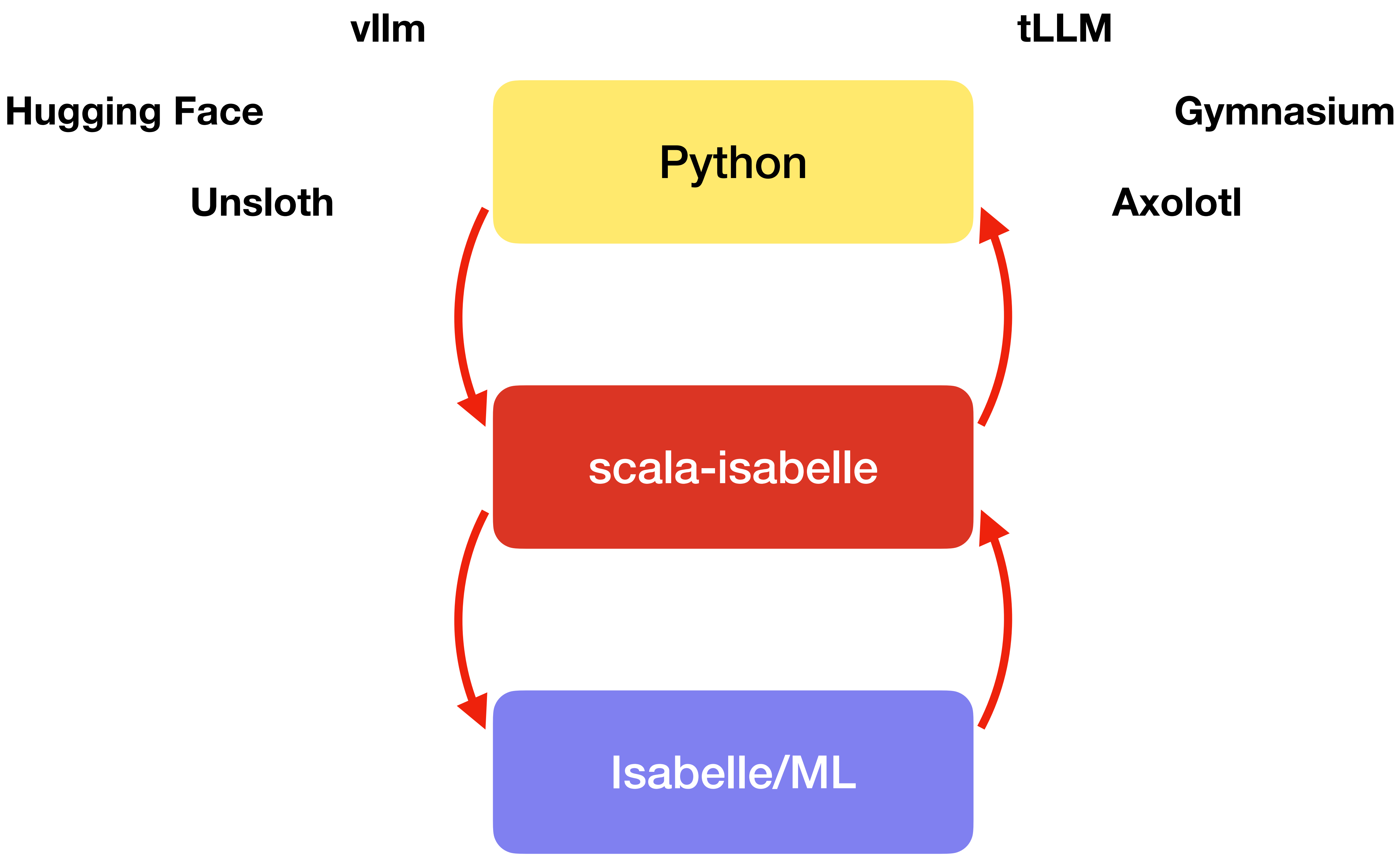
```
proof (prove)
using this:
  ∀n≥M. P i n

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!
```

The communication process



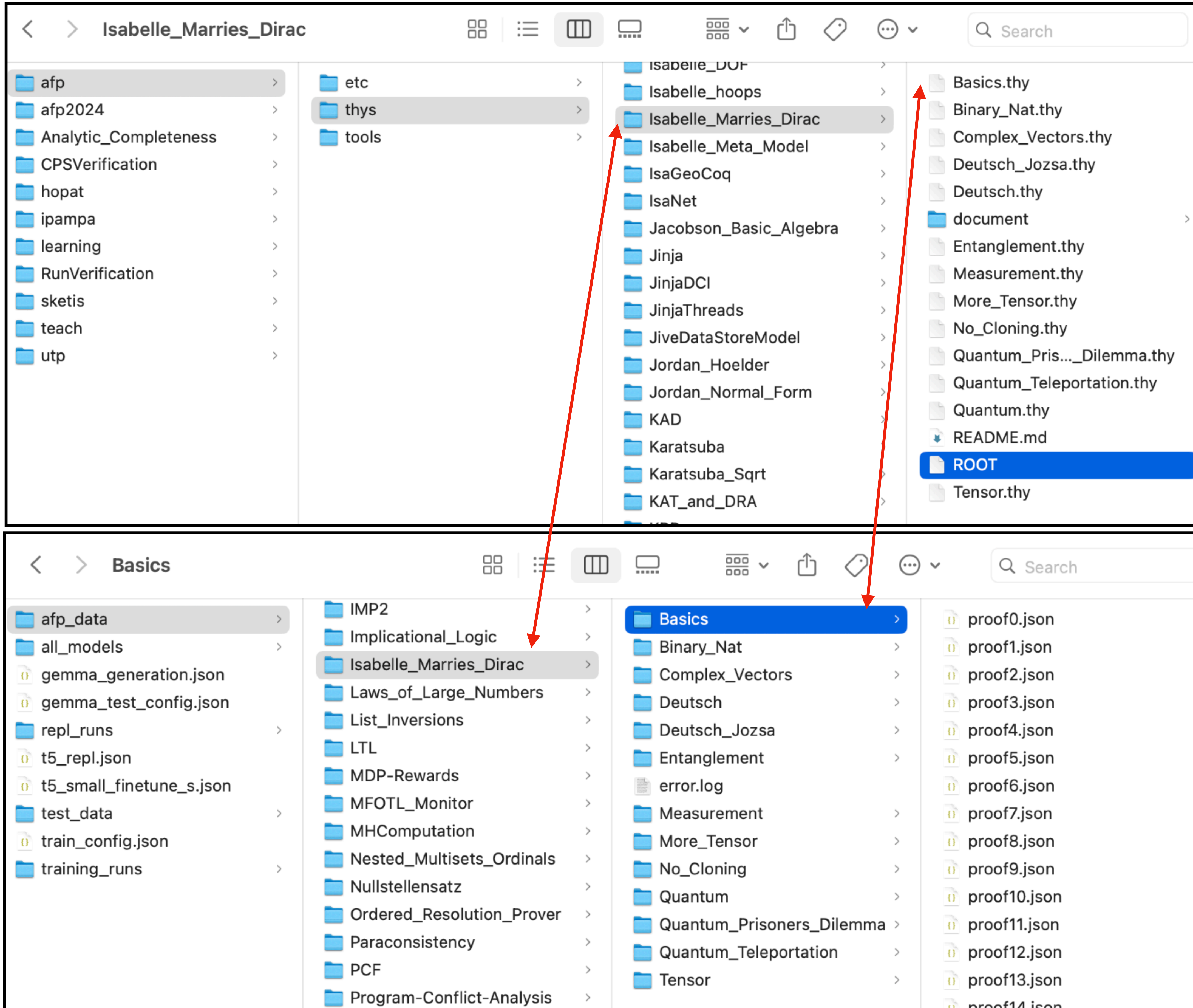
* plans to remove scala-isabelle in the future

Data Extraction Algorithm

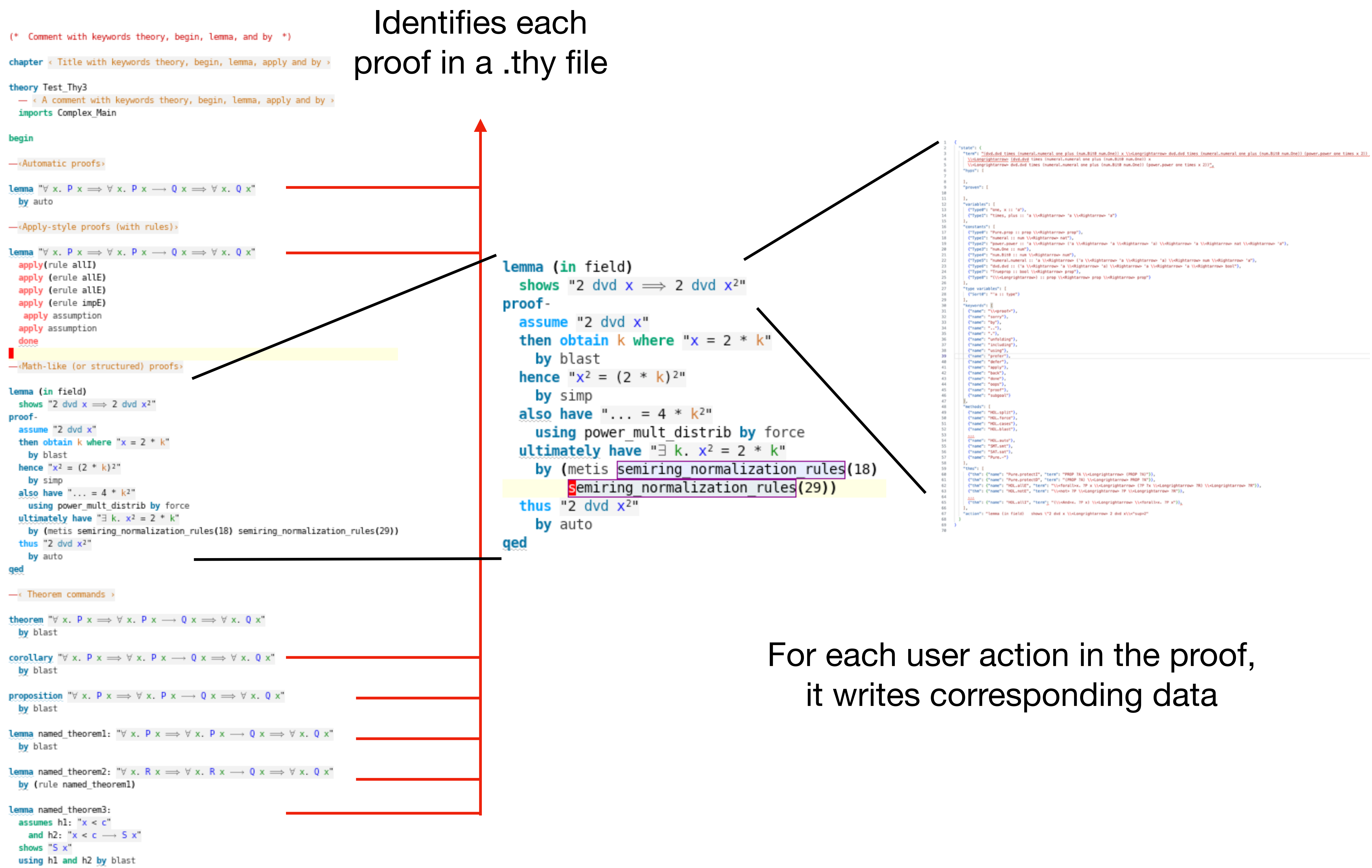
Data extraction process

AFP

Generated Data



Data extraction process



Content of the generated data

Contents:
- user action
- user state
- Isabelle state/term
- variables
- constants
- *type constants*
- *type variables*
- apply-keywords
- isar-keywords
- proof methods
- lemma dependencies

Easily extensible to include:
- classes
- locales
- definitions
- conjecturing
- Sledgehammer premises

Programs > deepIsaHOL > all_data > afp_data > Isabelle_Marries_Dirac > Basics > {} proof0.


```
{
  "proof": {
    "steps": [
      {
      "step": {
        "action": "lemma set_2_atLeast0 [simp]: \"{0..<2::nat} = {0,1}",
        "user_state": "",
        "term": "{0..<2} = {0, 1} \\<Longrightarrow> ({0..<2} = {0, 1})",
        "hyps": [

        ],
        "proven": [

        ],
        "variables": [
```


{} proof0.json ×

> deepIsaHOL > all_data > afp_data > Isabelle_Marries_Dirac > Basics > {} proof0.json > ...


```
  "proof": {
    "deps": [
      {"thm": {"name": "Code_Numeral.arity_unique_euclidean_semiring_natural", "term": "OFCLASS(natural, unique_euclidean_semiring_class)"}},
      {"thm": {"name": "Code_Numeral.arity_discrete_linordered_semidom_natural", "term": "OFCLASS(natural, discrete_linordered_semidom_class)"}},
      {"thm": {"name": "Quickcheck_Narrowing.arity_type_narrowing_type_IITN_narrowing_type", "term": "OFCLASS(narrowing_type.narrowing_type_IITN_narrowing_type, type_class)"}}
    ]
  }
}
```

How to use it?

Simple interface for data extraction either as a single command:

```
sbt "run /path/to/a/read/directory/ /path/to/a/writing/directory/"
```

or via the Scala or Python REPL:

```
import isabelle_rl._
import de.unruh.isabelle.control.Isabelle

val logic = "LTL" // Isabelle session name

val writer = new Writer(
    "/path/to/a/read/directory/", // it can have a ROOT, or a .thy file
    "/path/to/a/writing/directory/",
    logic)

writer.write_data("your/file.thy")

writer.write_all()

val minion = writer.get_minion()

implicit val isabelle: Isabelle = minion.isabelle

isabelle.destroy()
```

Read-eval-print-loops (REPL)

How to use it?

Start a Py4j server running a scala-isabellle process with
`sbt “runMain isabelle_rl.Py4j_Gateway_Main” `

```
>>> import sys
>>> sys.path.append('/path/to/this/project/src/main/python')
>>> from repl import REPL

>>> repl = REPL("HOL")
REPL and minion initialized.

>>> repl.apply("lemma \"\\<forall>x. P x \\<Longrightarrow> P c\"")
'proof (prove)  goal (1 subgoal):   1. \\<forall>x. P x \\<Longrightarrow> P c'

>>> repl.apply("apply simp")
'proof (prove)  goal:  No subgoals!'

>>> repl.apply("done")
''

>>> repl.shutdown_gateway() # do not forget to close the Isabelle process
```

Small-scale experiment

Experimental setup

- Use small language models to predict the **next user action** in Isabelle proofs

- Text-to-text transfer transformer (T5) language models (encoder-decoders) requiring the SentencePiece tokenizer:
  - **small** (77 million parameters)
  - **base** (248 million parameters)

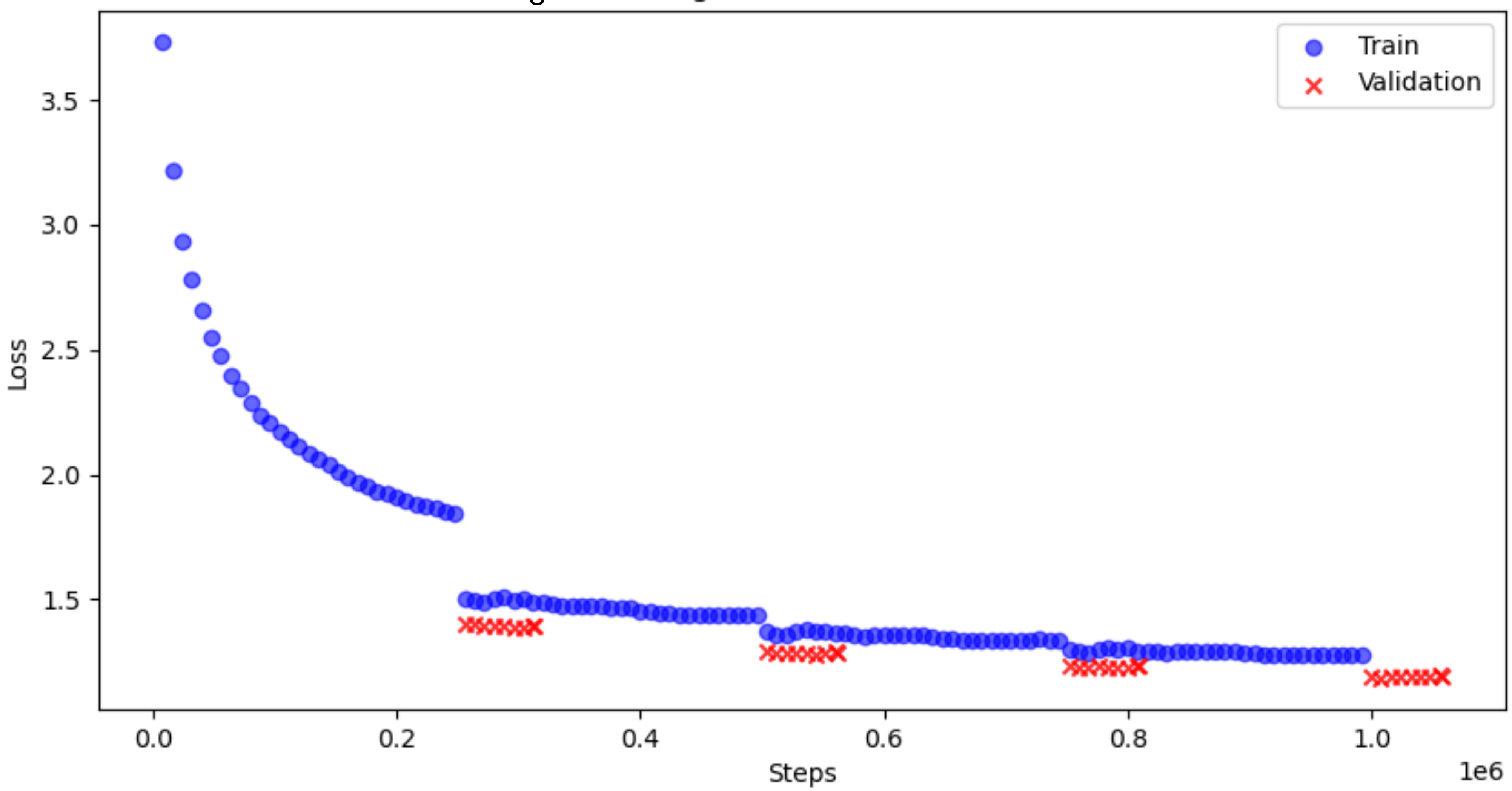
- To compare:
  - training from **scratch** vs **fine-tuning** them (T5 models are pre-trained with question-answering datasets on math *word* problems, coding problems for popular programming languages, and various English-language novels)
  - benefits of extra data: proof up to a given and user state **(s)** vs augmenting to that the premises to prove the theorem and the keywords available at that point **(spk)**

Experimental setup

AFP official numbers	2025 data	2024 data
~296,000 lemmas	198,108 proofs (~66.93%)	187,210 proofs (~63.24%)
926 AFP entries	922 AFP entries (99.57%)	861 AFP entries (92.98%)
	1,555,653 training proof steps	1,854,901 training proof steps

64% of each .thy file is reserved for *training*, 16% for *validation*, and 20% for *testing*

Average Training and Validation Loss for the fine-tuned T5 small model



Experimental setup

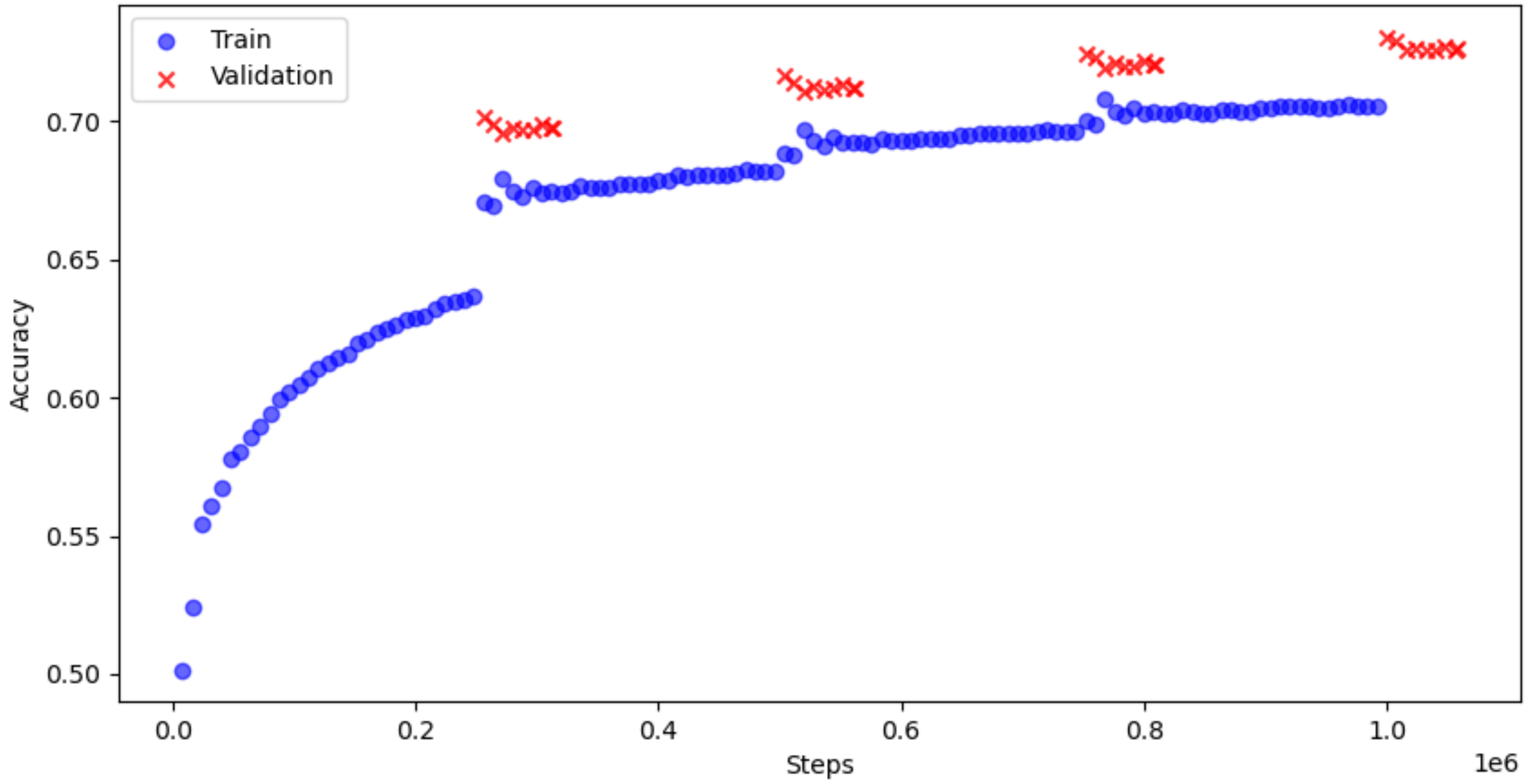
AFP official numbers
~296,000 lemmas
926 AFP entries

2025 data
198,108 proofs (~66.93%)
922 AFP entries (99.57%)
1,555,653 training proof steps

2024 data
187,210 proofs (~63.24%)
861 AFP entries (92.98%)
1,854,901 training proof steps

64% of each .thy file is reserved for *training*, 16% for *validation*, and 80% for *testing*

Average Training and Validation Accuracy for the fine-tuned T5 small model



Loss, accuracy, and other metrics (as LLMs)

Model	Train Loss	Train Acc.	Valid Loss	Valid Acc.
t5_small_s (4ep)	3.02	0.445	3.13	0.436
t5_small_finetune_s (4ep)	1.18	0.720	1.14	0.733
t5_small_spk (3ep)	2.96	0.417	3.29	0.375
t5_small_spk_trim (4ep)	2.71	0.438	3.21	0.371
t5_base_s (4ep)	2.84	0.474	3.06	0.441

Model	Test Exact Acc.	Test 1st-Tok Acc.	Test All Wrong
t5_small_s (4ep)	0.137	0.316	0.303
t5_small_finetune_s (4ep)	0.150	0.302	0.231
t5_small_spk (2ep)	0.042	0.254	0.553
t5_small_spk_trim (4ep)	0.022	0.194	0.371
t5_base_s (4ep)	0.170	0.307	0.299

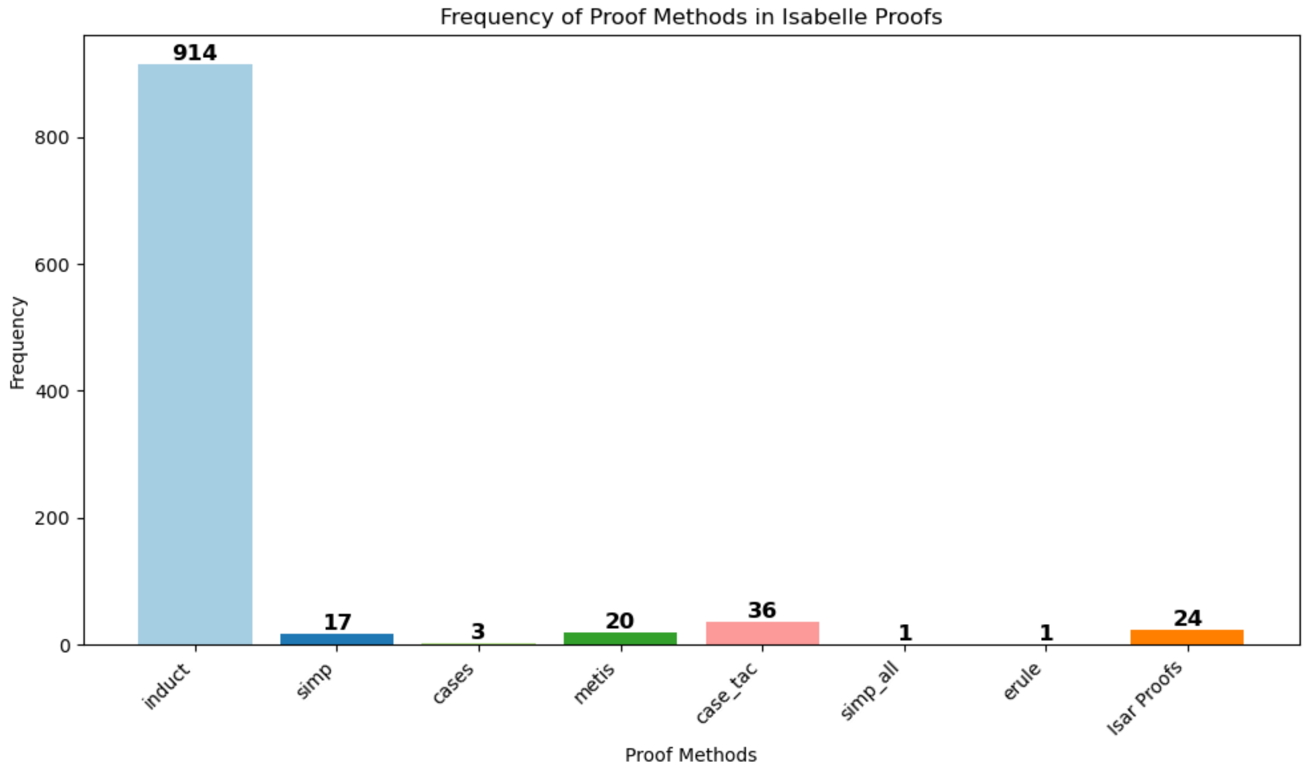
Model performance metrics after 4 epochs over the training data set
(except for the spk which was only trained over 3 epochs)

3. Training of T5 LLMs for simple experiments

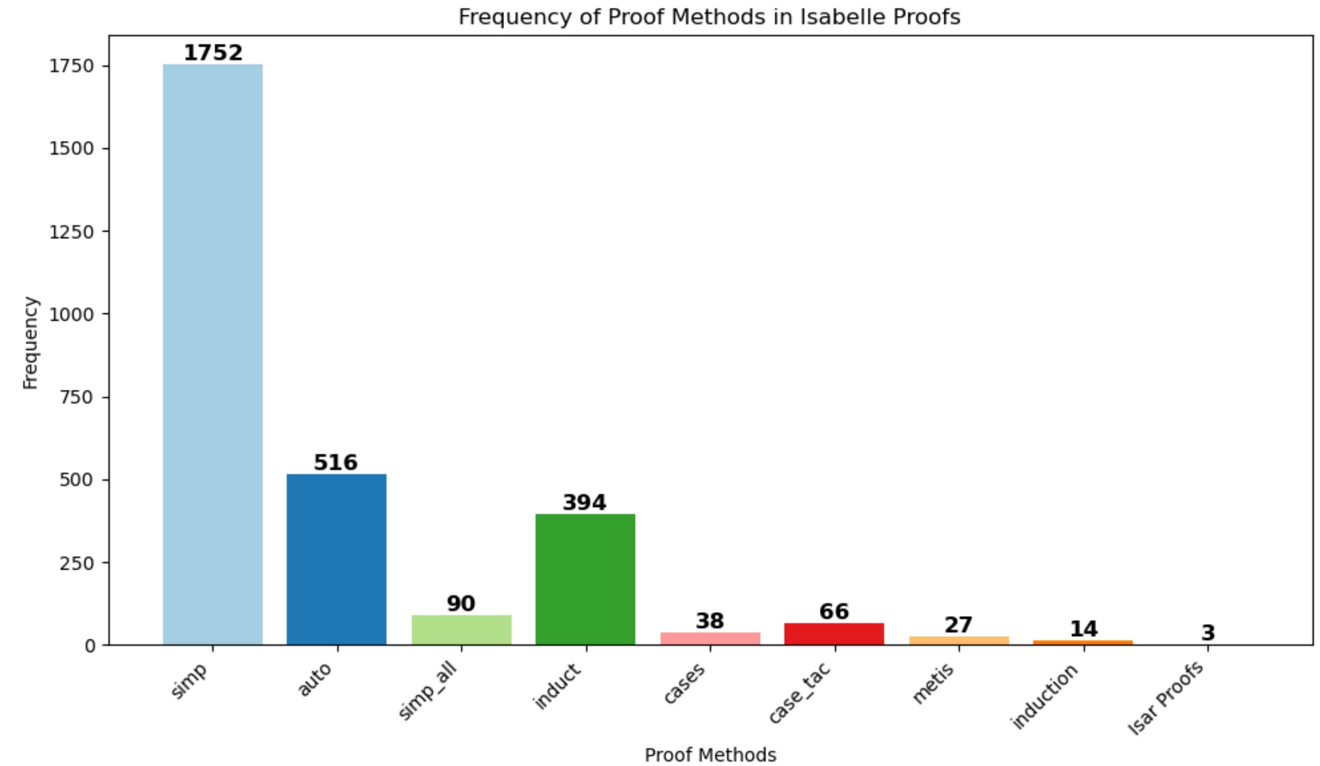
Let the models suggest the next user-action (5 opportunities) in a depth-first search (DFS) fashion for at most 5 consecutive user actions

Model	Predictions	Progress	by commands	Correct by	Proofs	Finished proofs	Avg. time
small_s	7,426,213	61.58 %	187,683	42.02 %	43,538	7.13 %	48.92 s
small_fs	3,508,777	37.79 %	162,986	8.40 %	42,408	11.28 %	40.09 s
small_spk	2,939,773	48.82 %	55,195	14.76 %	43,218	8.81 %	29.84 s
base_s	5,955,871	59.45 %	101,268	25.36 %	43,001	9.51 %	50.33 s
base_fspk	2,315,016	52.57 %	118,959	36.70 %	38,532	16.71 %	53.28 s

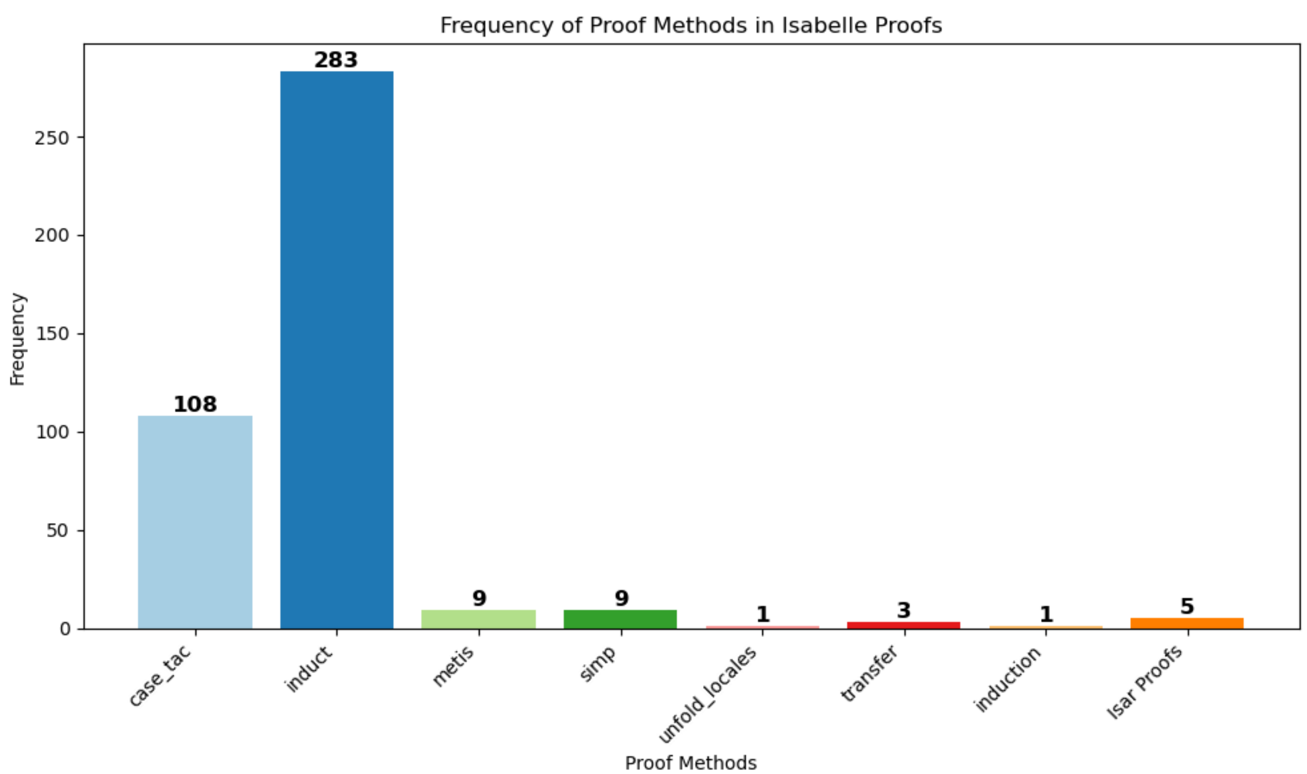
Extras on the small-scale experiment 3



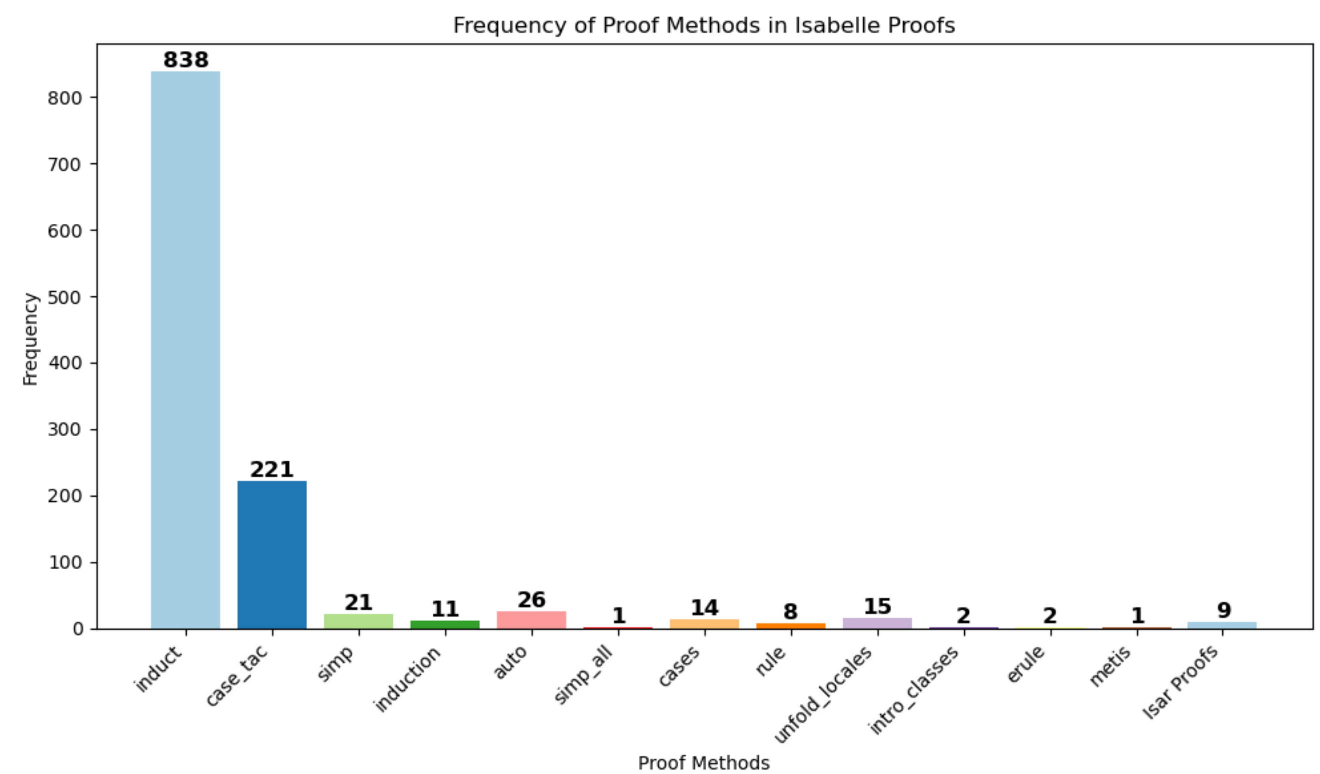
t5_small (s)



t5_small (fs)



t5_small (spk)

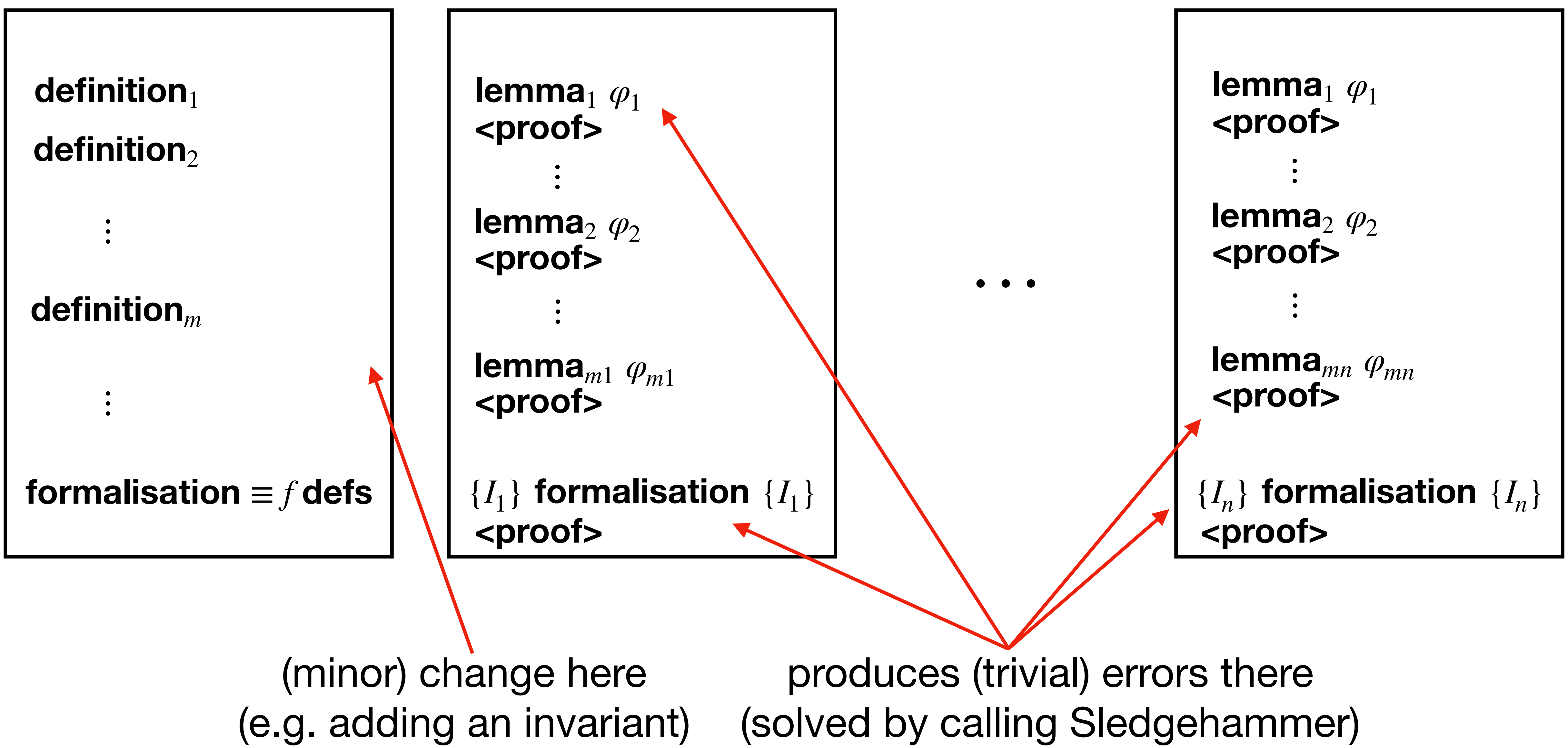


t5_base (s)

Proof-fixing tool

General problem

- Prove the partial correctness of a protocol: { **inv** } **formalisation** { **inv** }
- The invariant is really a huge conjunction: **inv** $\triangleq I_1 \wedge I_2 \wedge I_3 \wedge \cdots \wedge I_n$
- Each (minor) edition in the formalisation, breaks downstream proof progress:



General solution

super_fix: traverses your .thy file, it fixes misaligned proof obligations and replaces failed (or infinitely looping) proof methods with sorry's. Then, it traverses the .thy file again and calls Sledgehammer where each sorry appears. If Sledgehammer finds a proof, it replaces the sorry with that proof. Otherwise, it leaves the sorry to the user.

definition$_1$
definition$_2$
⋮
definition$_m$
⋮
formalisation $\equiv f$ **defs**

lemma$_1$ φ_1
<proof>
⋮
lemma$_2$ φ_2
<proof>
⋮
lemma$_{m1}$ φ_{m1}
<proof>

$\{I_1\}$ **formalisation** $\{I_1\}$
<proof>

. . .

lemma$_1$ φ_1
<proof>
⋮
lemma$_2$ φ_2
<proof>
⋮
lemma$_{mn}$ φ_{mn}
<proof>

$\{I_n\}$ **formalisation** $\{I_n\}$
<proof>

this helped complete the large scale proof of a cache-coherence protocol!

Conclusion

Next steps

- Call an external tool via an Isabelle tactic

- Continue the small scale experiment
  - Compare agains Sledgehammer
  - Variation of the model architecture (from encoder-decoder to autoencoder)
  - Usage of the latest optimised models (Gemma or DeepSeek-R1)
  - Train on small-scale premise selection
  - Train on “conjecturing” predicting the next state
  - Variations on the proof-exploration tree (best-first search, MCTS)

- Complete the infrastructure for reinforcement learning experiments

- Focus on cleverly engineered premise selection to improve the models’ performance

Previous ML approaches on ITPs

Prover\Strategy	Premise selection	Reinforcement Learning	Next step prediction for proof completion	Autoformalisation
HOL4	TacticToe (kNN)	Tactic Zero	TacticToe (kNN)	
Coq (now Rocq)			CoqPilot, Tactician (kNN, GNN, and LLM)	
Isabelle	Magnushammer		Thor, ***DeepIsaHOL***	Draft-Sketch-Prove, LEGO-Prover
Lean			LeanDojo's ReProver	Deep Seek Prover
HOL Light		*HOL-list (RL)*	*HOL-list (GNN)*	

Final invitation

DeepIsaHOL is an open project intended to serve the machine learning community and the ITP community.

Everyone is invited to collaborate, use it, and provide feedback

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

https://github.com/yonoteam/DeepIsaHOL