Seminar Foundations of Dependable Systems

Sylvain Julmy

December 13, 2018

Plan for today I

1. Typed meta-interpretive learning
   - Introduction and context
   - PSGraph
   - Meta-Interpretive Learning : MIL
   - Adding types
   - Encoding of a proof tree
   - Results
   - Conclusion
2. HOLStep : A machine learning dataset for HOL theorem proving
   - Introduction and context
   - Dataset extraction
   - Machine Learning Tasks
   - Neural Network Architecture
   - Results
   - Conclusion
3. Conclusion

Interactive Theorem Prover

Interactive Theorem Prover (ITP) becomes a more common way of establishing the correctness of software and mathematical proofs :

- Compilers
- Operating Systems
- Kepler Conjecture
- Feit-Thompson Theorem

Interactive Theorem Prover

ITPs enable users to interact with a proof system and guide a proof. They also support automation in terms of user-provided tactics.

Table of Contents

1. Typed meta-interpretive learning
   - Introduction and context
   - PSGraph
   - Meta-Interpretive Learning : MIL
   - Adding types
   - Encoding of a proof tree
   - Results
   - Conclusion
2. HOLStep : A machine learning dataset for HOL theorem proving
   - Introduction and context
   - Dataset extraction
   - Machine Learning Tasks
   - Neural Network Architecture
   - Results
   - Conclusion
3. Conclusion

Introduction and context

Ability to learn and re-apply proof strategies would significantly increase the productivity and effectiveness.

```
lemma K: A -> B -> A
proof
  assume A
  show B -> A
  proof
    show A by fact
  qed
qed
```

Introduction and context : problem

The user must often manually provide guidance and proofs, often group into famillies, such that, once the user has discharged one proof, discharging the other is simpler.

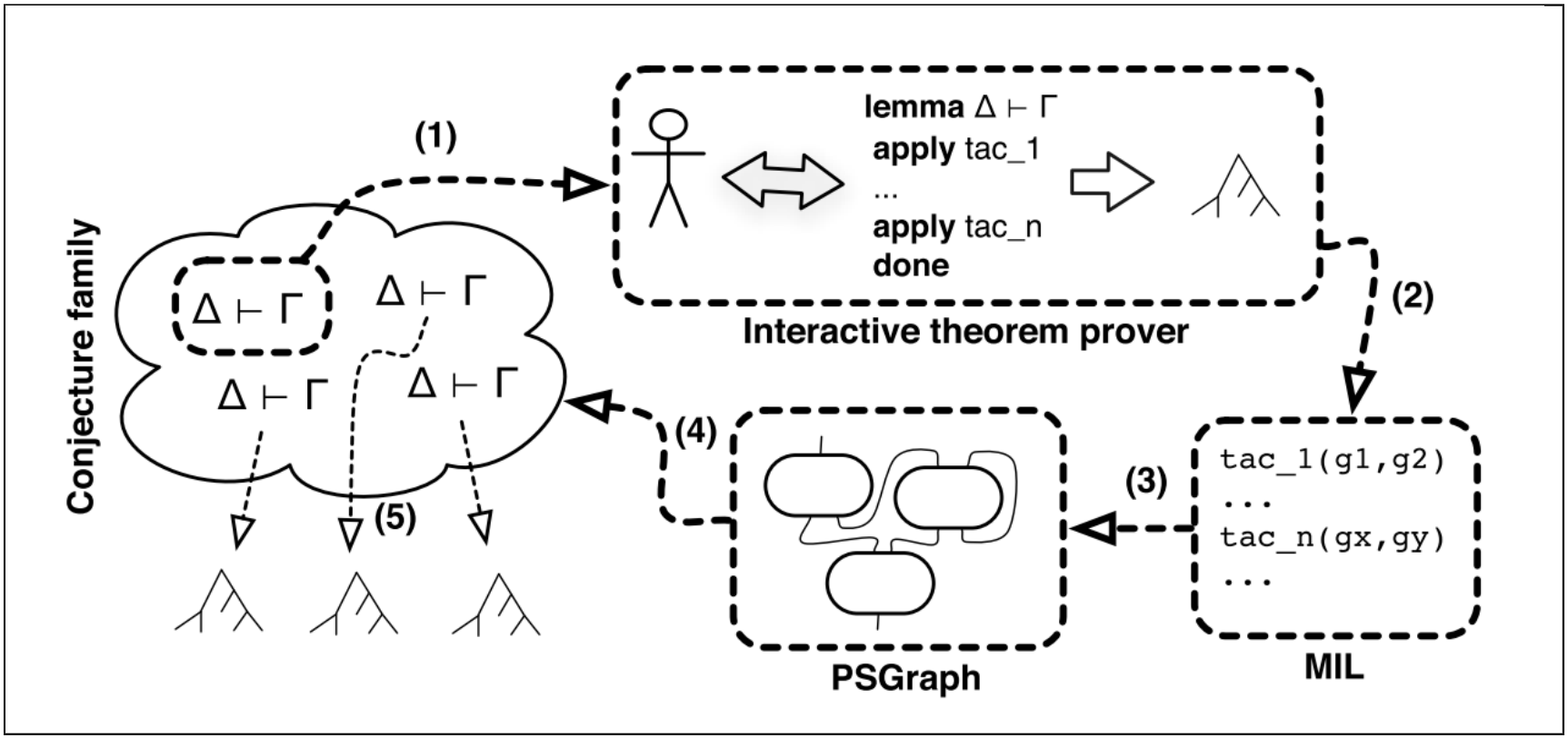
The idea is to learn and reapply proof strategies from a few examples.

Introduction and context

Previous work to learn proof strategies has only attempted to extract general strategies from a large corpus of proofs.

They are not addressing the desirable extraction of local strategies. Which was the motivation for developing the PSGraph language.

Overview of the approach



Overview of the approach.

Goal

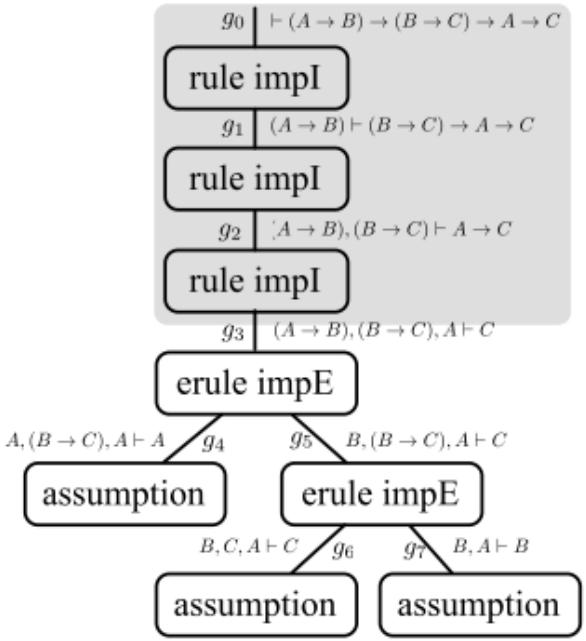
1. Show that MIL can learn proof strategies for the PSGraph language.
2. Extend MIL with types.
3. Demonstrate that "typed MIL learns more deterministic proof strategies than untyped MIL".
4. Show that introducing dependent learning reduces the time taken to learn a strategy.

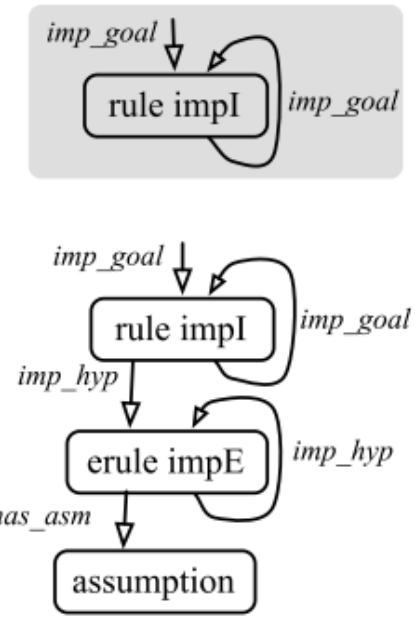
PSGraph

The language aims to represent proof strategies as graphs, where proof tactics are represented by boxes and goal information by predicate which label the wires.

ITP and PSGraph

lemma (A ⟶ B) ⟶
(B ⟶ C) ⟶
A ⟶ C
apply (rule impI)
apply (rule impI)
apply (rule impI)
apply (erule impE)
apply assumption
apply (erule impE)
apply assumption
apply assumption
done





Left to right: Isabelle proof script; proof tree; and strategy as PSGraph.

ITP and PSGraph

By proving a single conjecture a user will develop a proof strategy that he can reapply across a family of similar conjectures.

The strategies are normally in the head of a user, although an expert may manually encode them as tactics, the goal is to help automate the extraction and application of such strategies.

ITP and PSGraph

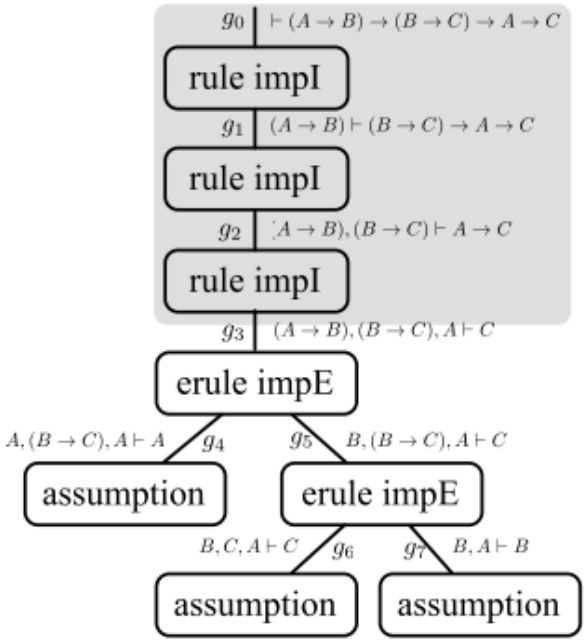
A proof strategy needs to include procedural information about which tactics to apply, the type of goal and progress made to show how a goal should evolve.

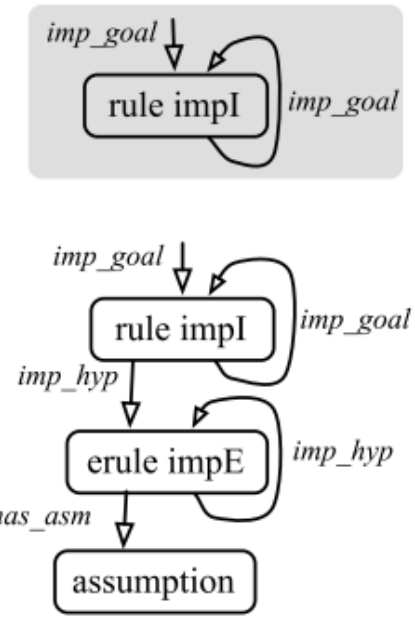
A directed labelled graph is used to capture proof strategies : the boxes of the graph contain the tactics, wires are labelled by wire predicates.

A graph is evaluated as a flow graph, where a goal flows between tactics on a directed edge if the predicate on the edge holds for that particular goal.

ITP and PSGraph

lemma (A ⟶ B) ⟶
(B ⟶ C) ⟶
A ⟶ C
apply (rule impI)
apply (rule impI)
apply (rule impI)
apply (erule impE)
apply assumption
apply (erule impE)
apply assumption
apply assumption
done





Left to right: Isabelle proof script; proof tree; and strategy as PSGraph.

Metagol

The framework is built on top of MIL by using Metagol.
Metagol[1] is an inductive logic programming (ILP) system based on meta-interpretive learning.

```
%% first-order background knowledge
mother(ann,amy).
mother(ann,andy).
mother(amy,amelia).
mother(linda,gavin).
father(steve,amy).
father(steve,andy).
father(gavin,amelia).
father(andy,spongebob).

%% predicates that can be used in the learning
prim(mother/2).
prim(father/2).
```

```
%% metarules
metarule([P,Q],([P,A,B]:-[[Q,A,B]])).
metarule([P,Q,R],([P,A,B]:-[[Q,A,B],[R,A,B]])).
metarule([P,Q,R],([P,A,B]:-[[Q,A,C],[R,C,B]])).

%% learning task
a :-
%% positive examples
  Pos = [
    grandparent(ann,amelia),
    grandparent(steve,amelia),
    grandparent(ann,spongebob),
    grandparent(steve,spongebob),
    grandparent(linda,amelia)
    ],
%% negative examples
  Neg = [
    grandparent(amy,amelia)
  ],
  learn(Pos,Neg).
```

Metagol : output

```
% clauses: 1
% clauses: 2
% clauses: 3
grandparent(A,B):-grandparent_1(A,C),grandparent_1(C,B).
grandparent_1(A,B):-mother(A,B).
grandparent_1(A,B):-father(A,B).
```

Metarules

Metagol requires higher-order metarules to define the form of clauses permitted in a hypothesis.

```
metarule([P,Q,R],([P,A,B]:-[[Q,A,C],[R,C,B]])).
```

Users need to supply Metarules. Work is in progress[1] for automatically identifying the necessary metarules.

[1] at the time of this paper

Typed Meta-Interpretive Learning

Typed Meta-Interpretive Learning

Typed MIL extends MIL by labelling each predicate P with a constant t representing its type. This is written $P : t$, such that $P(X, Y)$ becomes $P : t(X, Y)$. The type is treated as an extra constant argument, thus $P : t(X, Y)$ is internally represented as $P(t, X, Y)$.

Learning cases

Learning PSGraphs from example proofs can be reduced to two mutually de- pendent learning problems :

1. learning a graph's structure.
2. learning suitable wire predicates.

Previous learning attempts have simplified the problem to just the first point.

Learning cases

These two problems have different features :

1. requires learning clauses which can be translated into a graph with wire predicates
2. needs a large search space in order to learn unknown predicates.

Working with HOL this includes arbitrary recursive functions. In typed MIL we can control the structure of what is learned with metarules, and use types to separate the learning problems.

Used metarules

Typed MIL use the following metarules :

- *Lift* : construct a PSGraph consisting of a single node (tactic) and a labelled input edge.
- *Chain* : sequentially composes two such PSGraphs to find a larger strategy.
- *Loop* : introduce iteration.
- *WChain* : to learn wire predicate in chain.
- *WBin* : to learn binary wire predicate.

Lift

$$PSGraph(psgraph, g_x, g_y) \leftarrow predicate(wpred, g_x),$$
$$tactic(tactic, g_x, g_y).$$

Chain

$$PSGraph(psgraph, g_x, g_y) \leftarrow subgraph_1(psgraph, g_x, g_z),$$
$$subgraph_2(psgraph, g_z, g_y).$$

Loop

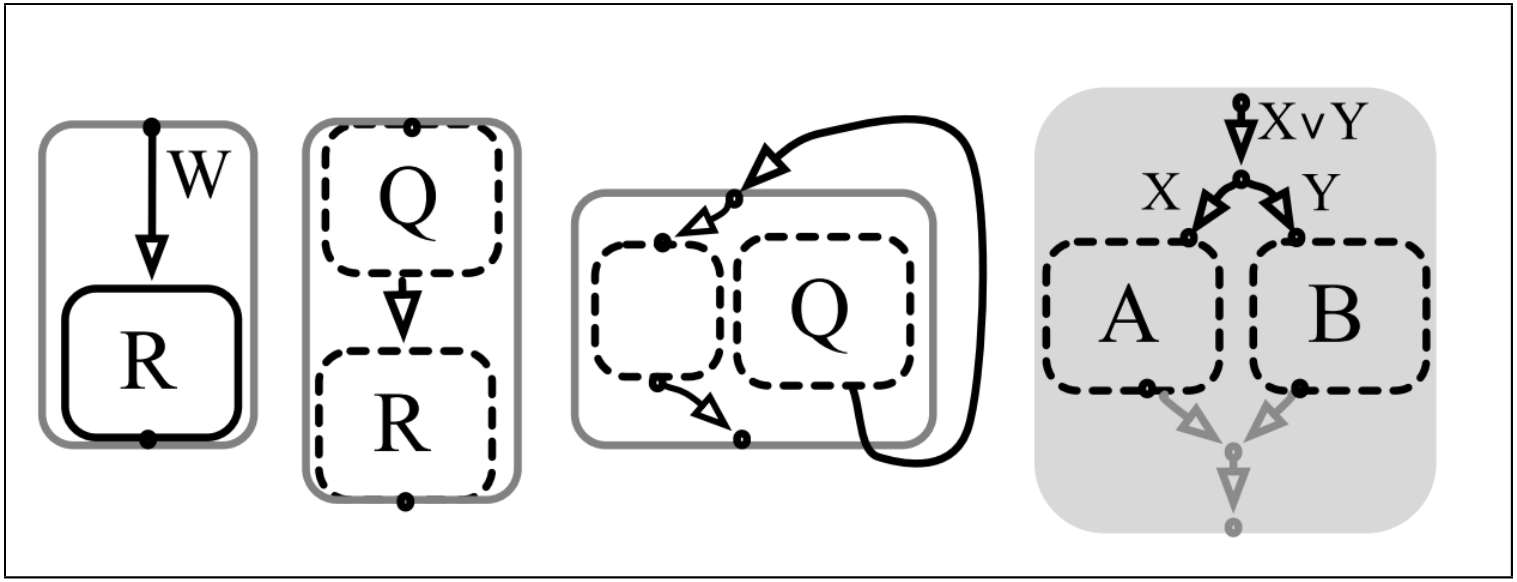
$$PSGraph\_base(psgraph, gx, gy) \leftarrow predicate(wpred, gx), tactic(tactic, gx, gy).$$
$$PSGraph\_rec(psgraph, gx, gy) \leftarrow PSGraph\_base(psgraph, gx, gz),$$
$$PSGraph\_rec(psgraph, gz, gy).$$

WBin and WChain

$$P : wpred(x) \leftarrow Q : gdata(x, z), R : gdata(x, z).$$

$$P : wpred(x) \leftarrow Q : gdata(x, z).$$

Metarules



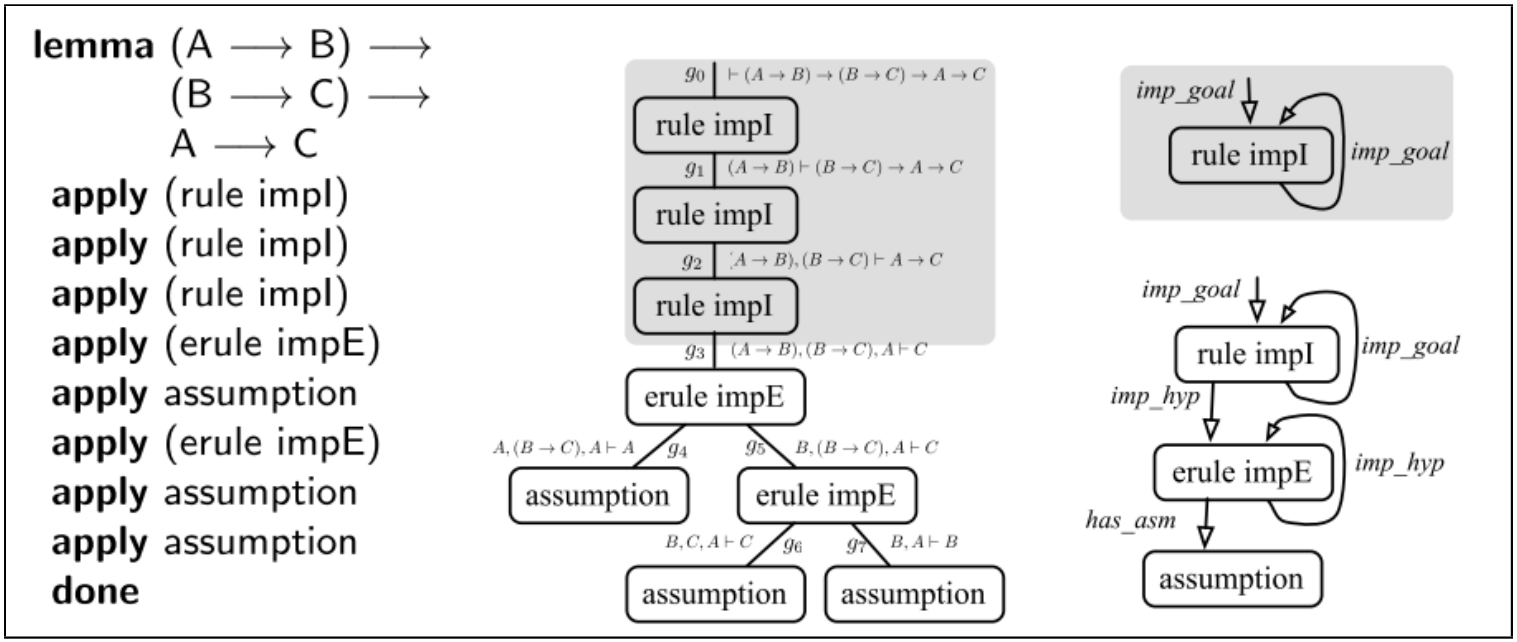
Metarules in PSGraph.

Proof tree

The metarules are used to learn proof strategies from an encoding of a proof tree.

A sub-tree is defined in terms of its boundary within a proof tree.

Proof tree encoding

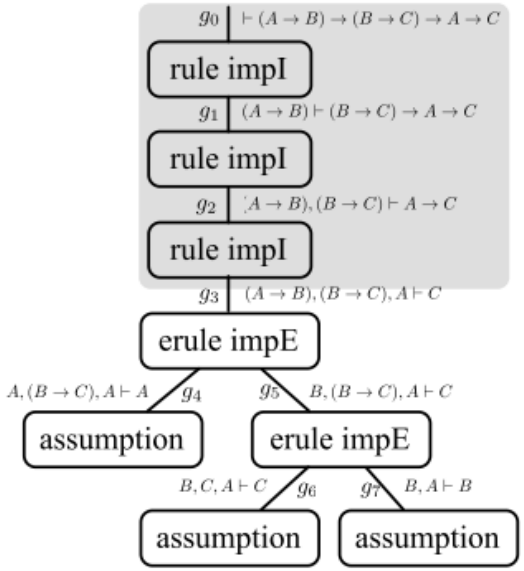


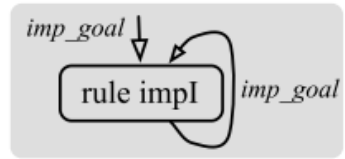
Previous example.

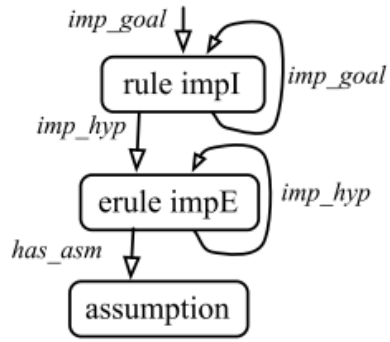
rule_impl : *tactic*($g0, g1$).*rule_impl* : *tactic*($g1, g2$).*rule_impl* : *tactic*($g2, g3$).

Proof tree encoding

lemma (A ⟶ B) ⟶
(B ⟶ C) ⟶
A ⟶ C
apply (rule impI)
apply (rule impI)
apply (rule impI)
apply (erule impE)
apply assumption
apply (erule impE)
apply assumption
apply assumption
done







Previous example.

$$erule\_impE : tactic(g3, g4).erule\_impE : tactic(g3, g5).$$

Proof tree encoding

$$g_2 \text{ is } A \to B, B \to C \vdash A \to C$$

These terms are projected from the goals by $hyp : gdata$ and $concl : gdata$.

$$has\_asm : wpred(G) \leftarrow hyp : gdata(G, T), concl : gdata(G, T).$$

Isabelle encoding

Isabelle stores terms as typed lambda expressions, using De Bruijn indices.

$$b(I)\ \ c(S)\ \ v(S)\ \ app(T, U)\ \ lambda(V, T)\ \ exists(T)\ \ forall(Y)$$

Encoding for g_2

$$hyp : gdata(g2, app(app(c(\rightarrow), c(a)), c(b))).$$
$$hyp : gdata(g2, app(app(c(\rightarrow), c(b)), c(c))).$$
$$concl : gdata(g2, app(app(c(\rightarrow), c(a)), c(c))).$$

Advantages over machine learning techniques

An advantage that ILP techniques have over machine learning techniques is that we can enrich the background clauses and use this to guide and simplify learning.

For example, definitions to extract the top level symbol in a conclusion or hypothesis are provided :

$$topsymbol : gdata(G, X) \leftarrow concl : gdata(G, app(app(X, A), B)).$$
$$hypsymbol : gdata(G, X) \leftarrow hyp : gdata(G, app(app(X, A), B)).$$

Proof tree encoding

In a proof-tree encoding each step of the proof tree is encoded as a relation of type tactic, with associated encoding of the goal information in terms of their hypotheses and conclusion.

Atomic term operator

The following clauses are atomic term operators :

$const : gdata(c(X))$ $\quad$ $var : gdata(v(X))$

$bound : gdata(b(X))$ $\quad$ $left : gdata(app(X, Y), X)$

$right : gdata(app(X, Y), Y)$ $\quad$ $into : gdata(forall(X), X)$

$into : gdata(exists(X), X)$ $\quad$ $into : gdata(lambda(V, X), X)$

Atomic term operator : example

The X in $app(app(X, A), B)$ is projected by two consecutive $left : gdata$ application.

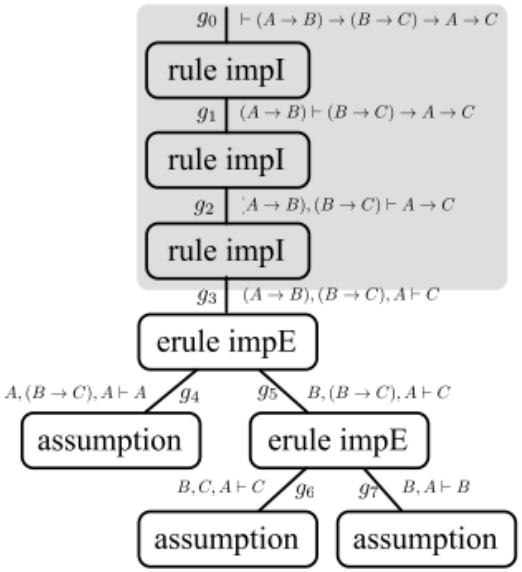
Learning problem

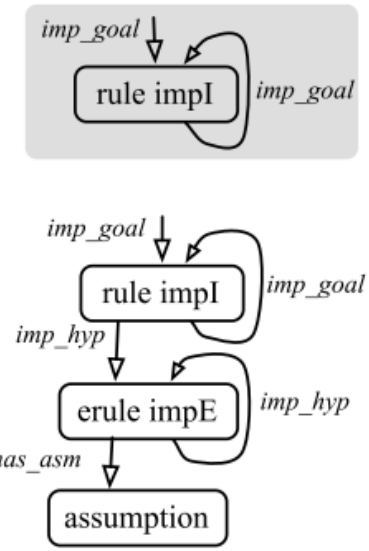
In typed MIL of PSGraph a binary relation of type PSGraph is learned where the background information at least contains encodings of one or more proof trees together with atomic term operators, and the given examples are one or more subtree boundaries of the encoded proof trees.

When using sub-trees and not the full trees we can learn sub-strategies and, as discussed later, we can apply dependent learning to learn increasily larger sub-strategies.

Learning problem : example

lemma (A ⟶ B) ⟶
(B ⟶ C) ⟶
A ⟶ C
apply (rule impI)
apply (rule impI)
apply (rule impI)
apply (erule impE)
apply assumption
apply (erule impE)
apply assumption
apply assumption
done



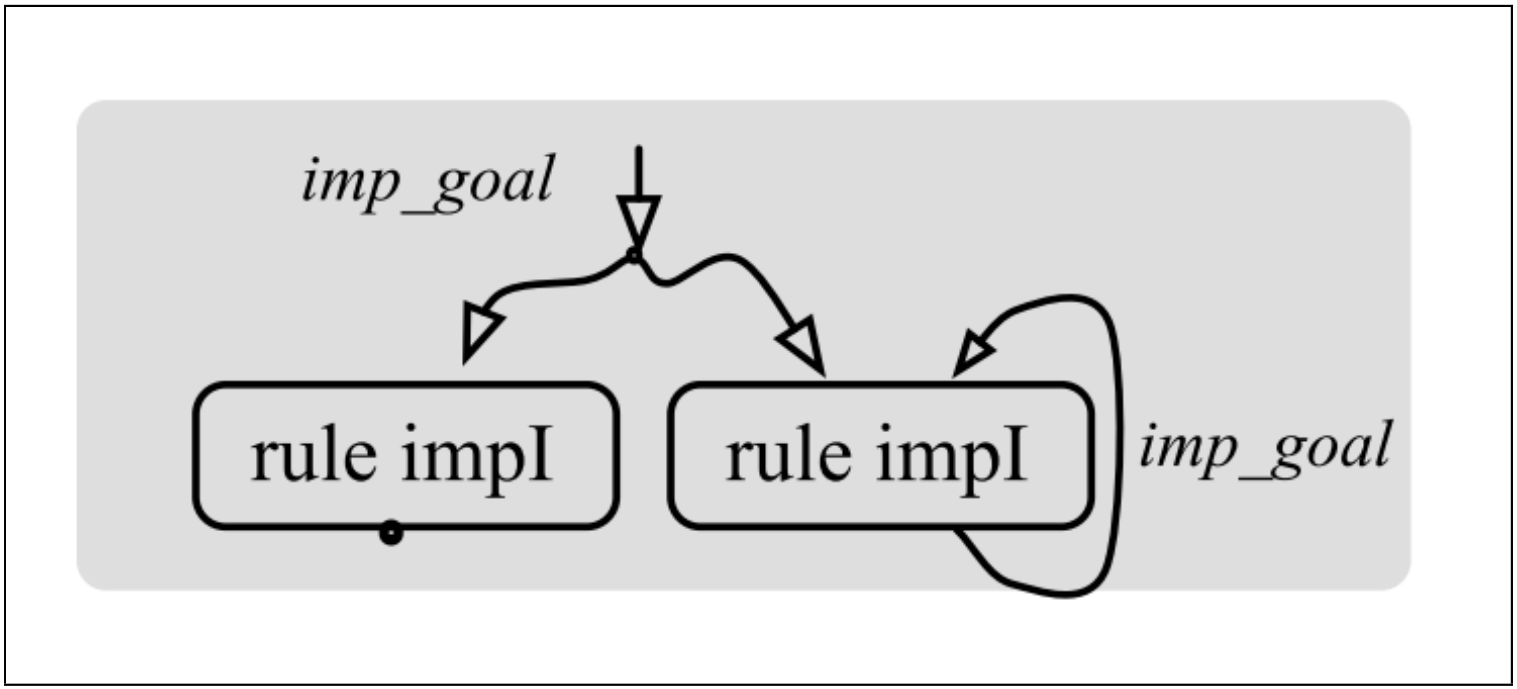


Previous example.

Learning problem : example

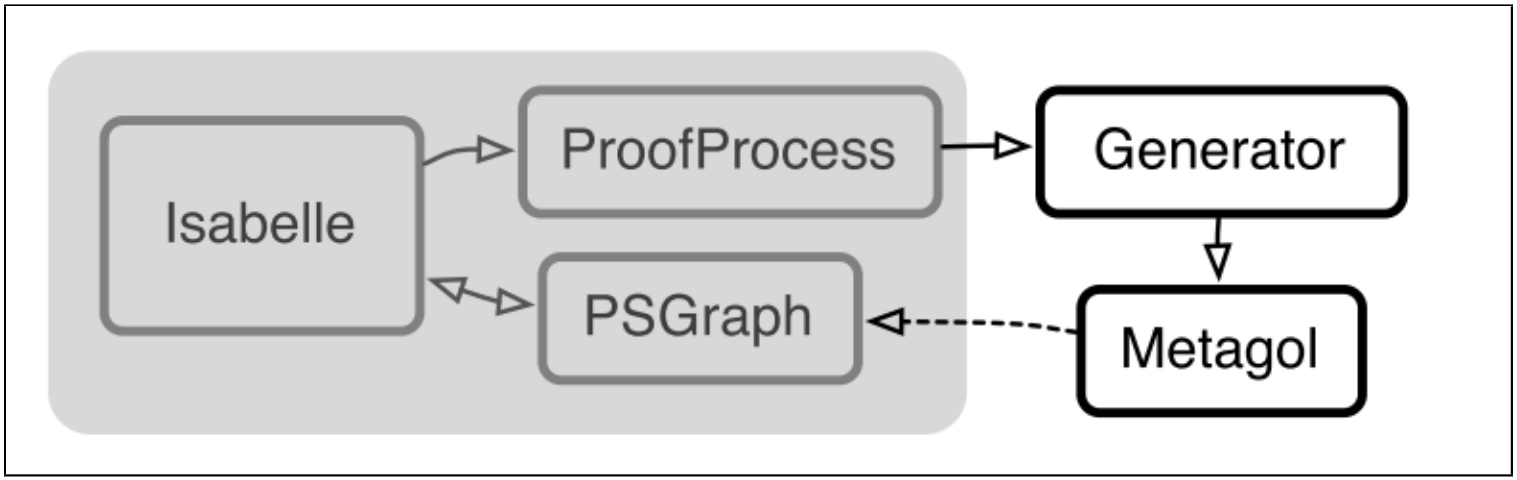
$$rimpl : psgraph(A, B) \leftarrow simpl : psgraph(A, C), rimpl : psgraph(C, B).$$
$$rimpl : psgraph(A, B) \leftarrow impgoal : wpred(A), ruleimpl : tactic(A, B).$$
$$simpl : psgraph(A, B) \leftarrow impgoal : wpred(A), ruleimpl : tactic(A, B).$$
$$impgoal : wpred(A) \leftarrow topsymbol : gdata(A, c(imp)).$$

Learning problem : example



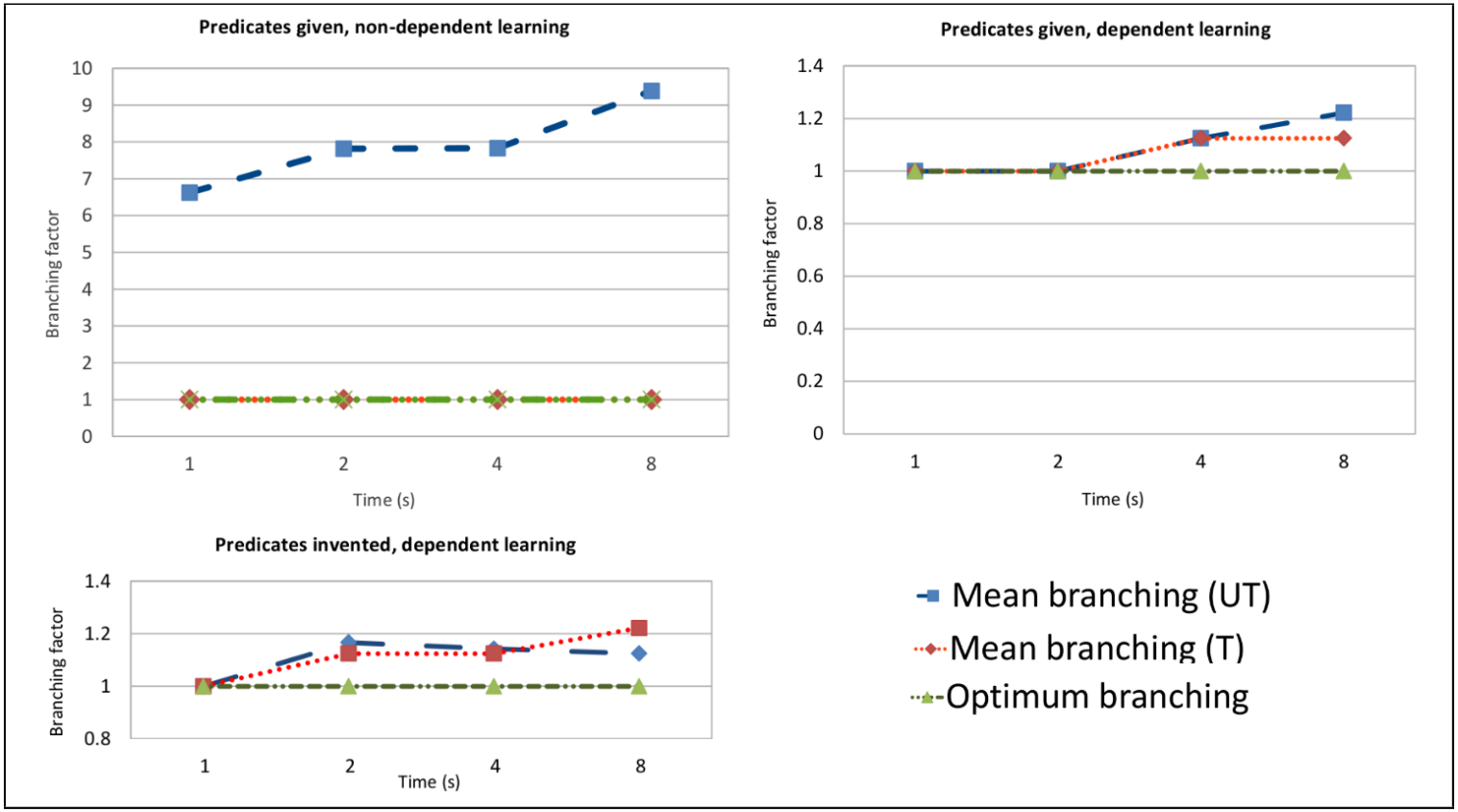
PSGraph encoding for the example.

Architecture



Tool architecture.

Results



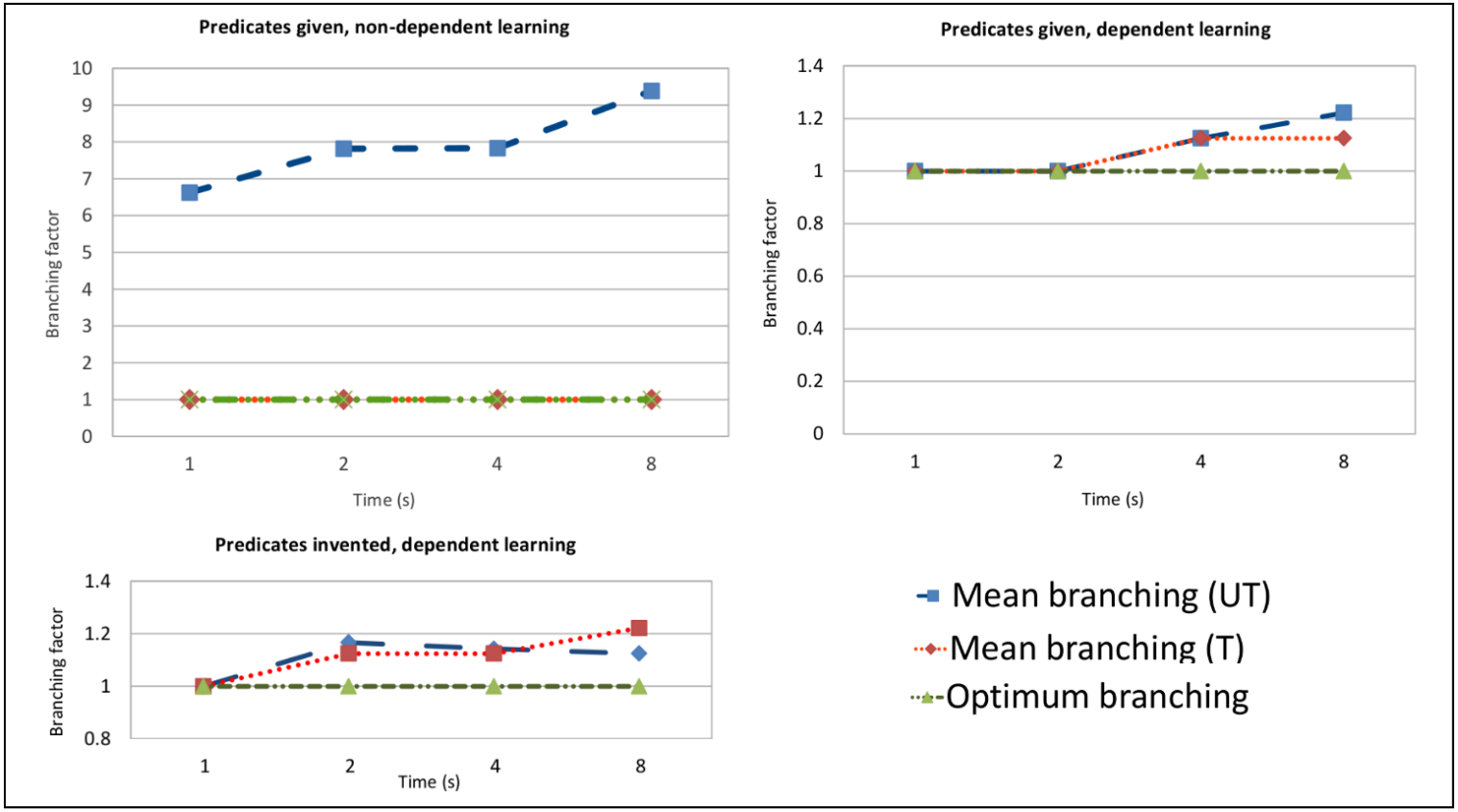
Mean branching factor ρ for untyped (UT) and typed (T) MIL. Experiments are running on 15 proofs in CL.

Results

These differences are due to how Metagol constructs its' solutions.
Consider a strategy consisting of repeated applications of a single tactic.

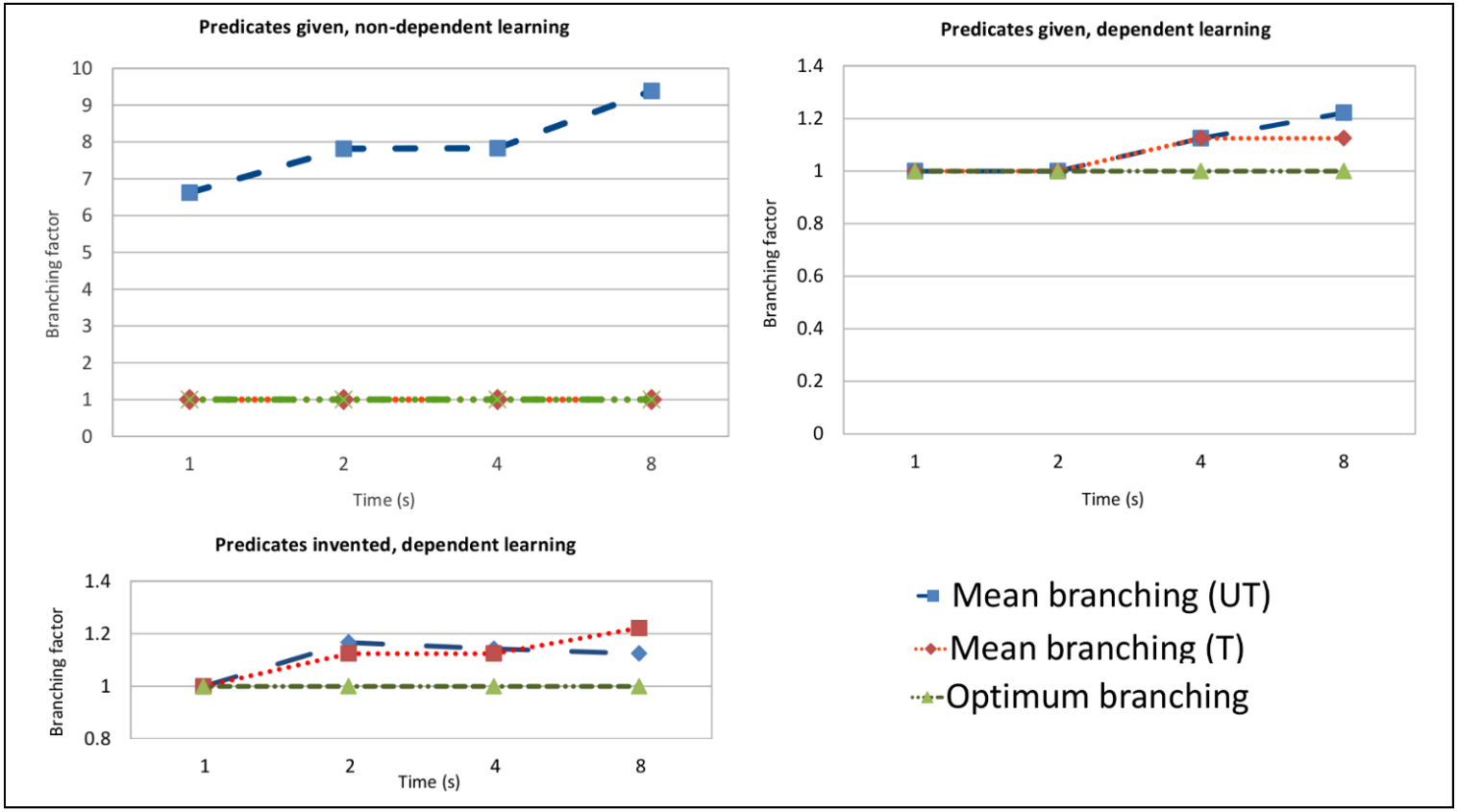
- Typed MIL forms this using the Loop rule.
- With untyped MIL there is no requirement to use psgraph clauses.

Results



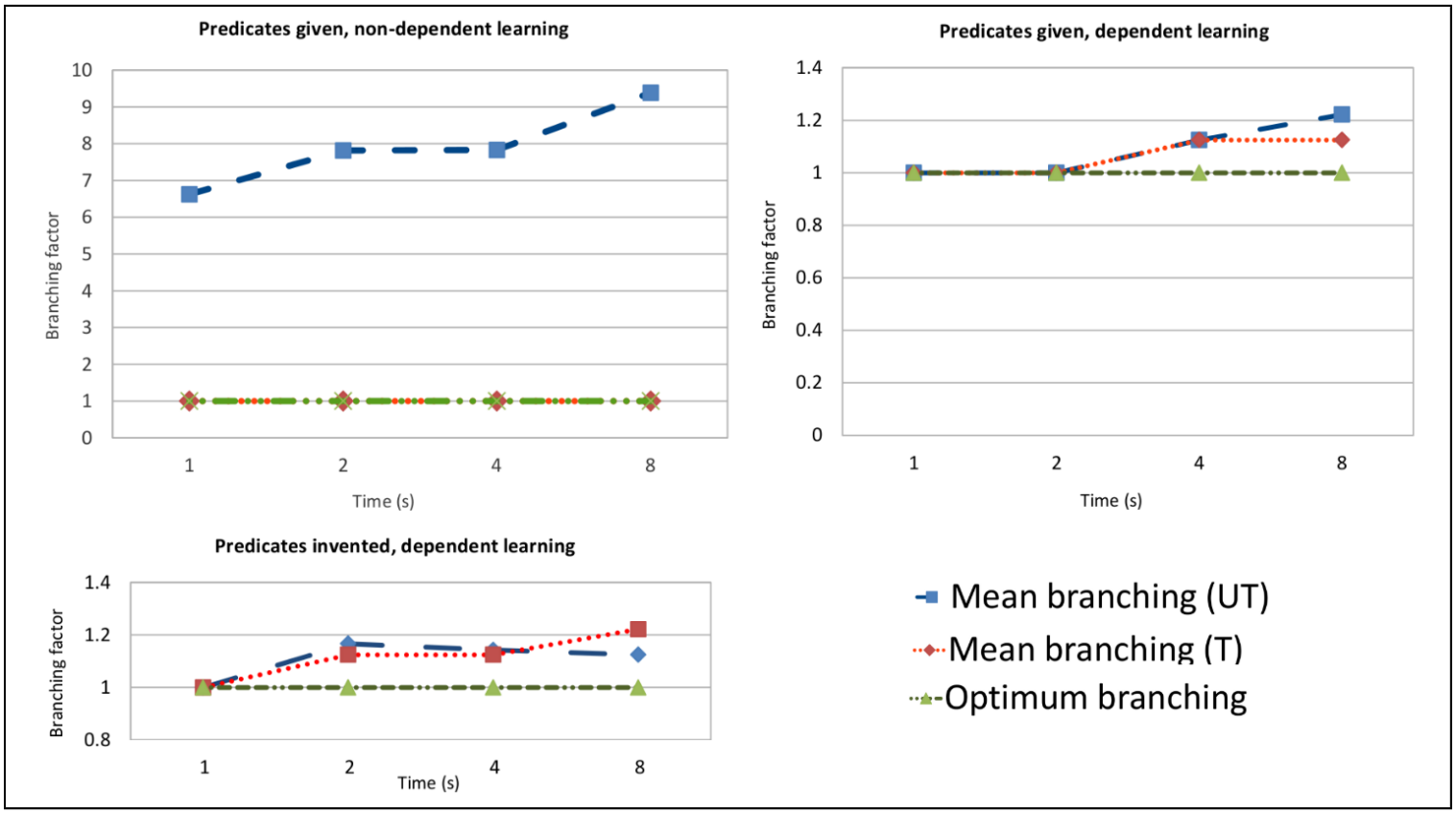
Mean branching factor ρ for untyped (UT) and typed (T) MIL.

Dependent learning



(Upper right) Smaller branching factor for UT.

Inventing pre-conditions



(Upper right) Smaller branching factor for UT.

Conclusion

1. Show that MIL can learn proof strategies for the PSGraph language.
2. Extend MIL with types.
3. Demonstrate that "typed MIL learns more deterministic proof strategies than untyped MIL".
4. Show that introducing dependent learning reduces the time taken to learn a strategy.

Further work

- Improving learned strategies by using a combinator-based approach.
- Introducing combinators (such as OR and LOOP) to handle multiple outputs.
- Learn from higher order logic (predicate logic).

Table of Contents

1. Typed meta-interpretive learning
   - Introduction and context
   - PSGraph
   - Meta-Interpretive Learning : MIL
   - Adding types
   - Encoding of a proof tree
   - Results
   - Conclusion
2. HOLStep : A machine learning dataset for HOL theorem proving
   - Introduction and context
   - Dataset extraction
   - Machine Learning Tasks
   - Neural Network Architecture
   - Results
   - Conclusion
3. Conclusion

Introduction and context

Deep learning has proven to be a powerful tool for embedding semantic meaning and logical relationships into geometric spaces (CNN).

- AlphaGo
- Premise selection
- Automated translation
- ...

Goal

- Create a dataset for machine learning based on the proof steps (focus on HOL Light).
- Discuss the proof step classification tasks that can be attempted using the dataset.
- Propose baseline models.
- Evaluate the models.

HOL Light

Focus on HOL Light for two reasons :

- follows the LCF approach.
- implements higher-order logic as its foundation.

Theorem selection

The theorems that are derived by most common proof functions are extracted by patching these functions. The remaining theorems are

extracted from the underlying OCaml programming language interpreter.

Testing and training sets

Training and testing examples are grouped by proof, for each proof we have

- the conjecture.
- the dependencies of the theorem.
- list of used and not used intermediate statements.

Statements

For each statements (conjecture, proof dependency, or intermediate statement) :

- human-like printout (with parenthesis).
- predefined tokenization.

Statistics

	Train	Test	Positive	Negative
Examples	2013046	196030	1104538	1104538
Avg. length	503.18	440.20	535.52	459.66
Avg. tokens	87.01	80.62	95.48	77.40
Conjectures	9999	1411	-	-
Avg. dependencies	29.58	22.82	-	-

HolStep dataset statistics.

Possible tasks

- Predicting whether a statement is useful in the proof of a given conjecture;
- Predicting the dependencies of a proof statement (premise selection);
- Predicting whether a statement is an important one (human named);
- Predicting which conjecture a particular intermediate statement originates from;
- Predicting the name given to a statement;
- Generating intermediate statements useful in the proof of a given conjecture;
- Generating the conjecture the current proof will lead to.

Is a statement useful ?

Focus on the first task, which can be specialized into two different tasks :

- Unconditioned classification of proof steps.
- Conditioned classification of proof steps.

interaction with an ITP

Tasks that require most human time :

- the search for good intermediate steps.
- the search for automation techniques able to justify the individual steps.
- searching theorem proving libraries for the necessary simpler facts.

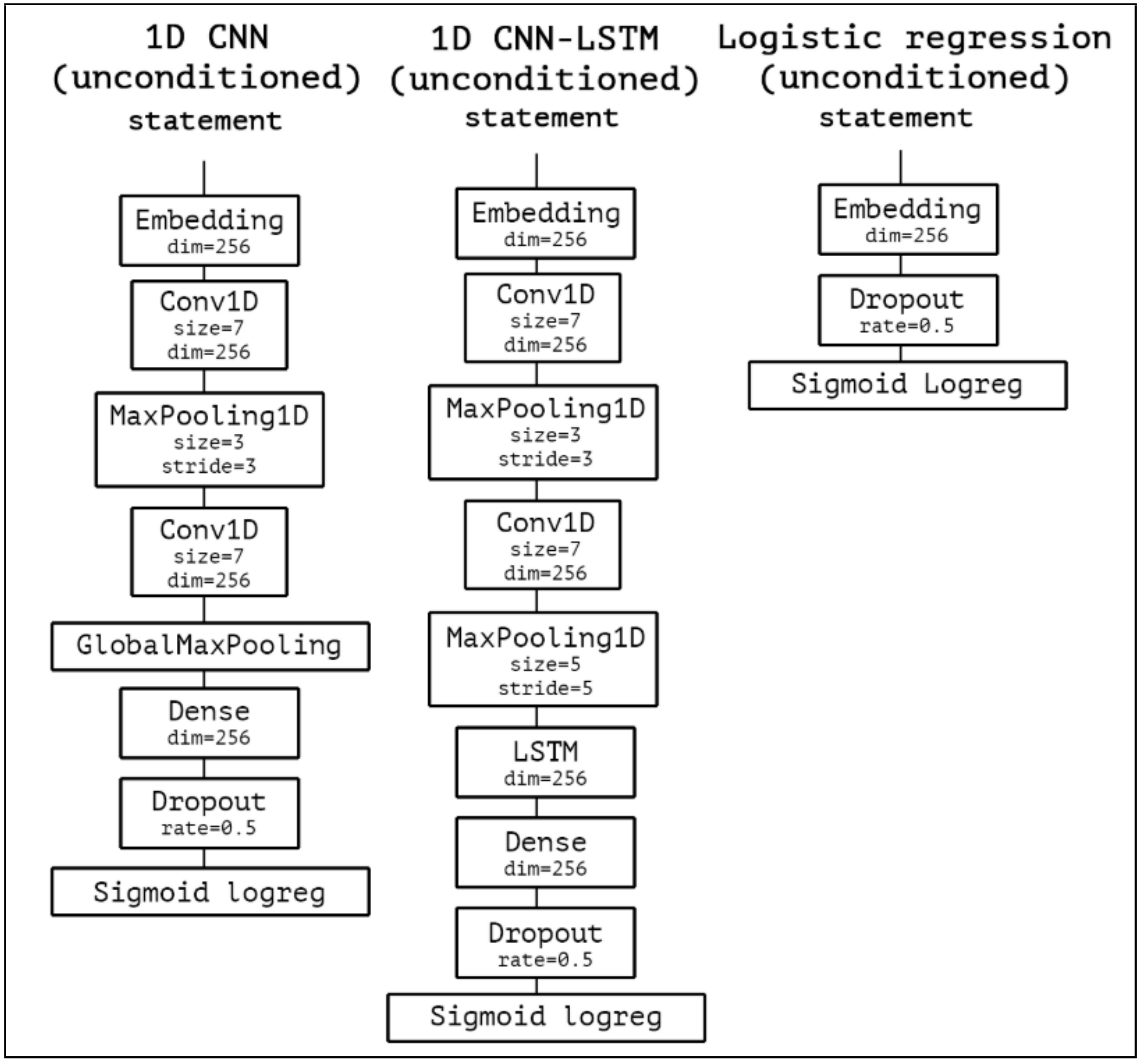
Corresponds to the machine learning tasks proposed previously.

Baseline models

For each tasks (conditioned and unconditioned classification) : three different deep learning architectures. Models are implemented in

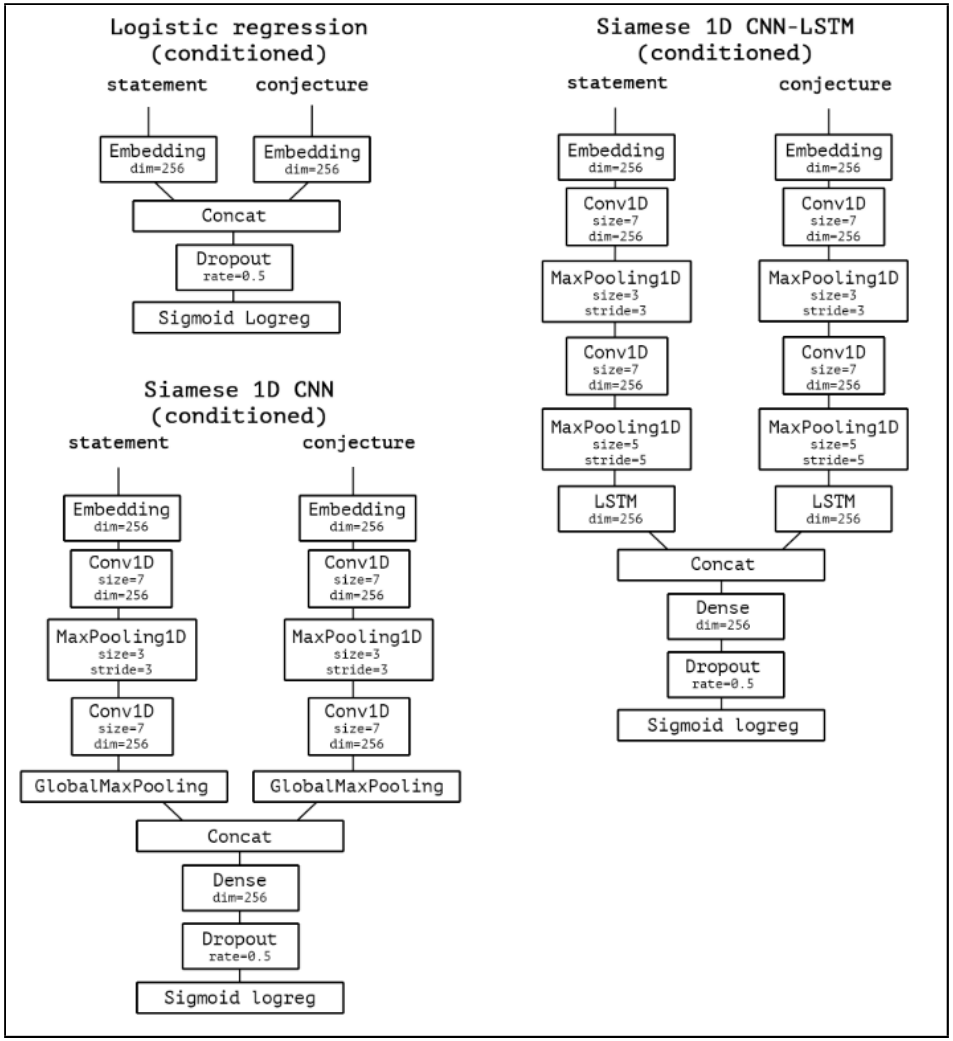
Tensorflow using the Keras framework.

Unconditioned classification models



Unconditioned classification model architectures.

Conditioned classification models



Conditioned classification model architectures.

Input statement encoding

Two types of encoding :

- Character-level encoding of the human-readable versions of the statements.
- Token-level encoding of the versions of the statements.

Results : architecture

Table 2: HolStep proof step classification accuracy without conditioning

	Logistic regression	1D CNN	1D CNN-LSTM
Accuracy with char input	0.71	0.82	**0.83**
Accuracy with token input	0.71	**0.83**	0.77

Table 3: HolStep proof step classification accuracy with conditioning

	Logistic regression	Siamese 1D CNN	Siamese 1D CNN-LSTM
Accuracy with char input	0.71	0.81	**0.83**
Accuracy with token input	0.71	0.82	0.77

Results : input encoding

Table 2: HolStep proof step classification accuracy without conditioning

	Logistic regression	**1D CNN**	**1D CNN-LSTM**
Accuracy with char input	0.71	0.82	**0.83**
Accuracy with token input	0.71	**0.83**	0.77

Table 3: HolStep proof step classification accuracy with conditioning

	Logistic regression	**Siamese 1D CNN**	**Siamese 1D CNN-LSTM**
Accuracy with char input	0.71	0.81	**0.83**
Accuracy with token input	0.71	0.82	0.77

Results : conditioning

Table 2: HolStep proof step classification accuracy without conditioning

	Logistic regression	1D CNN	1D CNN-LSTM
Accuracy with char input	0.71	0.82	**0.83**
Accuracy with token input	0.71	**0.83**	0.77

Table 3: HolStep proof step classification accuracy with conditioning

	Logistic regression	Siamese 1D CNN	Siamese 1D CNN-LSTM
Accuracy with char input	0.71	0.81	**0.83**
Accuracy with token input	0.71	0.82	0.77

Conclusion

- The baseline deep learning models is fairly weak, but still able to predict statement usefulness.
- Not able to leverage order in the input sequences, nor conditioning on the conjectures.
- No formal reasoning.
- Use pattern matching on tokens and characters.
- Improvement : HOL graph structure.

Future work

- Focus on more ITP and ATP.
- Premise selection and intermediate sentence generation.

Table of Contents

1. Typed meta-interpretive learning
   - Introduction and context
   - PSGraph
   - Meta-Interpretive Learning : MIL
   - Adding types
   - Encoding of a proof tree
   - Results
   - Conclusion
2. HOLStep : A machine learning dataset for HOL theorem proving
   - Introduction and context
   - Dataset extraction
   - Machine Learning Tasks
   - Neural Network Architecture
   - Results
   - Conclusion
3. Conclusion

References

Andrew Cropper and Stephen H Muggleton. *Metagol System*. https://github.com/metagol/metagol. 2016. URL: https://github.com/metagol/metagol.