Inductive Logic Programming for Interactive Theorem Proving

Liao Zhang

University of Innsbruck,

Czech Technical University

September 7, 2023

Motiviation

- AITP has many challenges:
  - bad at incremental learning
  - poor explainability
  - very large action space
  - bad at planning
  - low accuracy
  - ...
- Only statistical ML is applied for TP.
- I feel statistical ML is not enough for TP.
- Could we combine a very different ML technique called inductive logic programming (ILP) with statistical ML to develop stronger ML for TP?
- Use ILP to predict tactics for Coq.

ILP

- A subfield of symbolic artificial intelligence
- Goal: induce hypotheses that generalize training examples with background knowledge.
- Represented by first- or higher-order logic

Background Knowledge
mathematician(alice), mathematician(bob), artist(clark), artist(david), use_itp(alice), use_itp(clark), use_itp(david)

Positive Example
happy(alice)

Negative Examples
happy(bob), happy(clark), happy(david)

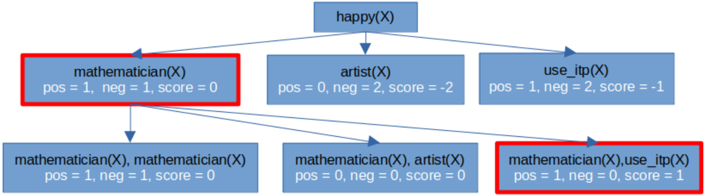
- Hypothesis: `happy(X) :- mathematician(X), use_itp(X)`
- Interpretation: forall X, if X is a mathematician and uses ITP, he is happy.

Aleph

Background Knowledge
mathematician(alice), mathematician(bob),
artist(clark), artist(david),
use_itp(alice), use_itp(clark), use_itp(david)

Positive Example
happy(alice)
Negative Examples
happy(bob), happy(clark),
happy(david)

- Arguably the most influential ILP system
- With appropriate parameters, we make Aleph roughly work as below.
- score = pos - neg



Encoding 1

```
n : nat
i : nat
H : i < n
H0 : ∀ n : nat, fact (S n) = S n * fact n
H1 : n - i = S (n - S i)
----------------------------------------------------
i < n
```

- Every node in the abstract syntax tree (AST) is converted to a fact.
- The encoding of the goal:

```
coq_Init_Peano_lt(57,[0]). goal_coq_var(57,"i",[0,0]).
goal_coq_var(57,"n",[0,1]).
```

- 57: the id of the proof state
- $[0,1]$: the position of the node in the AST
- i: the name of the hypothesis that the node refers to

Encoding 2

```
n : nat
i : nat
H : i < n
H0 : ∀ n : nat, fact (S n) = S n * fact n
H1 : n - i = S (n - S i)
--------------------------------------------------
i < n
```

- The encoding of `H`:

```
coq_Init_Peano_lt(57, "coq_H",["coq_H",hyp_ass,0]).
hyp_coq_var(57, "i", "coq_H",["coq_H",hyp_ass,0,0]).
hyp_coq_var(57, "n", "coq_H",["coq_H",hyp_ass,0,1]).
```

- coq_H: the hypothesis that the node belongs to
- [“coq_H”, hyp_ass, 0, 1]: the position of the node in the hypothesis
- The texts “coq_H” and hyp_ass are used to distinguish the position of a hypothesis from the position of a goal.

Predicates

```
dif(HypName1, HypName2).
dif(HypPosition1, HypPosition2).
dif(GoalPosition1, GoalPosition2).
left(HypPosition1, HypPosition2).
left(GoalPosition1, GoalPosition2).
above(HypPosition1, HypPosition2).
above(GoalPosition1, GoalPosition2).
% Two subterms rooted at HypPosition1 and HypPosition2 are equal.
eq_subterm(ProofStateId, HypPosition1, HypPosition2).
eq_subterm(ProofStateId, GoalPosition1, GoalPosition2).
eq_subterm(ProofStateId, HypPosition, GoalPosition).
hyp_coq_var(ProofStateId, HypName, HypName, HypPosition).
goal_coq_var(ProofStateId, HypName, GoalPosition).
```

Example 1

```
n : nat
i : nat
H : i < n
H0 : ∀ n : nat, fact (S n) = S n * fact n
H1 : n - i = S (n - S i)
----------------------------------------------------
i < n
```

```
tac(A,"assumption") :-
    coq_Init_Peano_lt(A,B), coq_Init_Peano_lt(A,C,D), eq_subterm(A,B,D).
```

Example 2

```
H1 : i <= N
---------------
y = y ^ 1
```

run simpl

```
H1 : i <= N
---------------
y = y * 1
```

```
tac(A,"simpl") :-
    coq_Init_Peano_le(A,B,C), coq_Init_Datatypes_O(A,D),
    coq_Reals_Rpow_def_pow(A,E), position_above(E,D),
    coq_Init_Datatypes_S(A,F), position_above(F,D).
```

Example 3

- Sometimes we can learn complicated structures.
- But the structures may not correspond to the rules in Coq users' minds.
- Simplify the power of one to the multiplication of one.

```
x : R
y : R
n : nat
Hrecn : (x + y) ^ n = sum_f_R0
  (fun i : nat ⇒ C n i * x ^ i * y ^ (n - i)) n
----------------------------------------
x + y = (x + y) ^ 1
```

```
tac(A,"simpl") :-
    coq_Init_Datatypes_O(A,B), coq_Reals_Rdefinitions_RbaseSymbolsImpl_Rplus(A,C),
    coq_Reals_Rpow_def_pow(A,D), position_above(D,C), coq_Init_Datatypes_S(A,E),
    coq_Reals_Rdefinitions_RbaseSymbolsImpl_Rplus(A,F), dif(F,C).
```

Compared to Statistical ML

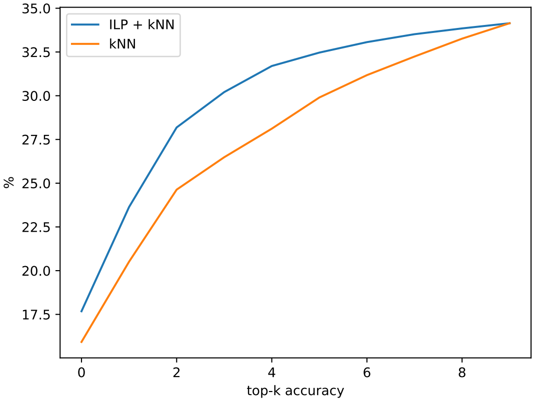
- ILP can represent relations:
  - horizontal relation and vertical relation
    - Features used in existing systems are only AST walks up to certain lengths.
  - equality
  - reference: a node and the hypothesis that it refers to
- ILP can characterize tactics.
- ILP is more explainable.

Connecting to Statistical ML

- The k-nearest neighbors classifier
  - The k-NN classifier ranks tactics by the distance measurement.
  - $jacard(f_1, f_2) = \frac{f_1 \cap f_2}{f1 \cup f_2}$ where f_i is a set of features
- Train ILP
  - tactic t
  - positive examples: proof states applied with t.
  - negative examples: proof states that are not applied with t.
  - Use ILP to generate many rules for each tactic.
- Testing
  - Use k-NN to predict 10 tactics $t_1, .., t_{10}$ for a proof state.
  - $\forall t_i$, if t_i does not satisfy any rule generated by ILP, add it to bad.
  - Else, add it to $good$.
  - Return $good + bad$

Results

- Train k-NN and ILP in 10% data from the Coq standard library.
- Test the performance in another 10%.



The End