First Experiments with Learning Instantiations

Konstantin Korovin   Josef Urban

University of Manchester

Czech Technical University in Prague

ICMS 2018
July 27, 2018

European Research Council
Established by the European Commission

Motivation: Lemmatization and Instantiation

- Lemmatization:
- Add to a problem lemmas useful in previous proofs
- Instantiation:
- Add to a problem instances useful in previous proofs

Previous Lemmatization Experiments - HOL Light and Flyspeck

- Over 1B low-level (proof) lemmas in Flyspeck
- 1.5M-7M higher-level lemmas in Flyspeck
- Define fast preprocessing methods to extract the most important ones:
- PageRank, recursive dependency count, recursive use count, etc.
- Use the most important lemmas together with the toplevel theorems
- helps by 5-20% on HOL Light/Flyspeck
- Quite often the lemmas are instances or easy consequences

Example of Useful Flyspeck Proof Lemmas

Conjecture:

```
AFFINE_ALT: |- affine s <=> (!x y u. x IN s /\ y IN s
                  ==> (&1 - u) % x + u % y IN s)
```

Easy, but useful lemmas (derived inside previous proofs):

```
NEWDEP309638: |- &1 - a + a = &1
NEWDEP310357: |- -- &1 * -- &1 = &1
NEWDEP272099_conjunct1: |- !m. &m + -- &m = &0
```

Previous Lemmatization Experiments - E prover

- refutational TPTP proofs of many Mizar problems
- postprocess (redirect) them to obtain direct proofs
- collect the proof lemmas derived only from the axioms
- Use them along with the toplevel premises for proving new conjectures
- 6% improvement on a Mizar-based benchmark

Example of Useful Mizar Proof Lemmas

Lemma:

```
X1 \ (X2 \ X1) = X1
```

proved in

```
X c= Y implies  Z \ Y c= Z \ X
```

useful in

```
X \/ (Y \ X) = X \/ Y

(X \/ Y) \ Z = (X \ Z) \/ (Y \ Z)

etc.
```

Why Learn Instantiation?

- Many of the proof lemmas in the previous experiments are useful instances
- A targeted instance prevents redundant inferences
- Instantiation-based ATP calculi are reaching state of the art
- SMT: quantifier instantation
- iProver: calculus based on gradual instantiation (and SAT)

iProver

- instantiate (abstract) all clauses with a unique constant and call a SAT solver
- if unsatisfiable, we are done
- else add more complicated ground instances – the Inst-Gen rule:
- from $L \vee C$, $\neg L' \vee D$ create $(L \vee C)\theta$, $(\neg L' \vee D)\theta$
- and continue
- Creation of the new instances is guided by the propositional assignment.
- In particular, we instantiate clauses with unifiable literals that have different value in the ground model
- However, this is often still very non-deterministic

Our Idea and Plan

- If we immediately guess the right instantiations, we are done (if it's SAT-easy)
- How do we guess them?
- Machine learning from many related problems!
- Our research plan:
  - **Add all previously useful clause instances**
  - Train a ranker and add only some number of most useful clause instances for the current context (analogous to premise selection)
  - Generate new instances by learning useful terms for the current context (another variant of premise selection)
  - Generate new terms for the current context by a trained probabilistic grammar

First Experiment

- Run standard iProver on 32525 Mizar problems
- solves 21685 problems with 10s time limit
- from the 21685 proofs extract the useful instances for each initial clause C
- whenever an unsolved problem contains C, add *all* the previously useful instances
- rerun iprover on the unsolved problems
- This adds 621 problems
- Many problems are too big - we really need the pruning

Future and Thanks

- That’s all we did for now
- Encouraging, but we need to proceed with the next steps
- Thanks for your attention!