FEMaLeCoP: OCaml leanCoP and Fairly Efficient Learning

Cezary Kaliszyk Josef Urban Jiří Vyskočil

UIBK and ČVUT

09 November 2015

Talk Overview

- Connection Tableaux and leanCoP
- leanCoP in OCaml and HOL
  - ARLT and Hammers
  - Proof Certification
  - Reconstruction
  - Integration in HOL
- leanCoP and Learning
  - MaLeCoP
  - Features
  - Indexing and Learning
  - Advising

leanCoP: Lean Connection Prover (Jens Otten)

- Connected tableaux calculus
  - Goal oriented, good for large theories
- Regularly beats Metis and Prover9 in CASC
  - despite their much larger implementation
  - very good performance on some ITP challenges
- Compact Prolog implementation, easy to modify
- Has variants for other foundations
  - iLeanCoP

Lean connection Tableaux

$$\frac{}{\{\},\ M,\ Path}\quad \textit{Axiom}$$

$$\frac{C,\ M,\ \{\}}{M}\quad \textit{Start}\quad \text{where } C \in M, C \text{ is positive}$$

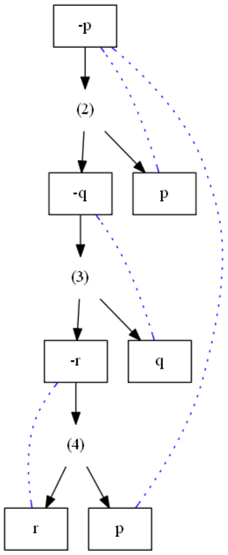
$$\frac{C,\ M,\ Path \cup \{L_2\}}{C \cup \{L_1\},\ M,\ Path \cup \{L_2\}}\quad \textit{Reduction}\quad \text{where } \sigma(L_1) = \sigma(\overline{L_2})$$

$$\frac{C_2 \setminus \{L_2\},\ M,\ Path \cup \{L_1\} \qquad C,\ M,\ Path}{C \cup \{L_1\},\ M,\ Path}\quad \textit{Extension}\quad \text{where } \begin{array}{c} \sigma(L_1) = \sigma(\overline{L_2}), \\ \sigma \text{ is rigid}, \\ C_1 \in M, L_2 \in C_2, \\ C_2 \text{ is a copy of } C_1 \\ \text{with vars renamed} \end{array}$$

leanCoP: Example of Connection Tableau

```
fof(1,conjecture,~p).
fof(2,axiom,p => q).
fof(3,axiom,q => r).
fof(4,axiom,r => ~p).
```

- DNF vs CNF approach (leanCoP vs most resolution provers)
- Axioms $\Longrightarrow$ Conjecture
- $\neg$ Axioms $\vee$ Conjecture
- CNF for Axioms: (2) (-p | q) & (3) (-q | r) & (4) (-r | -p)
- DNF for $\neg$ Axioms: (2) (p & -q) | (3) (q & -r) | (4) (r & p)
- starting DNF: (1) [-p] (2) [p,-q] (3) [q,-r] (4) [r,p]



leanCoP: Basic Code

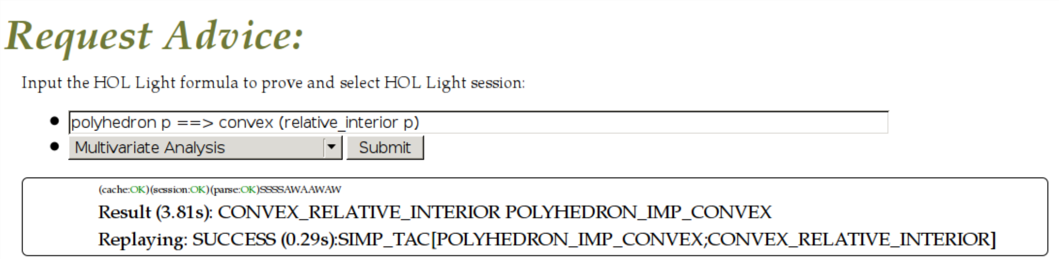
```
prove([Lit|Cla],Path,PathLim,Lem,Set) :-
  %
  (-NegLit=Lit;-Lit=NegLit) ->
    ( %
      %
      %
      member(NegL,Path), unify_with_occurs_check(NegL,NegLit)
    ;
      lit(NegLit,NegL,Cla1,Grnd1),
      unify_with_occurs_check(NegL,NegLit),
      %
      %
      %
      prove(Cla1,[Lit|Path],PathLim,Lem,Set)
    ),
    %
    prove(Cla,Path,PathLim,Lem,Set).
prove([],_,_,_,_).
```

leanCoP: Actual Code (Optimizations, No history)

```
prove([Lit|Cla],Path,PathLim,Lem,Set) :-
  \+ (member(LitC,[Lit|Cla]), member(LitP,Path), LitC==LitP),
  (-NegLit=Lit;-Lit=NegLit) ->
    (
      member(LitL,Lem), Lit==LitL
    ;
      member(NegL,Path), unify_with_occurs_check(NegL,NegLit)
    ;
      lit(NegLit,NegL,Cla1,Grnd1),
      unify_with_occurs_check(NegL,NegLit),
        ( Grnd1=g -> true ;
          length(Path,K), K<PathLim -> true ;
          \+ pathlim -> assert(pathlim), fail ),
      prove(Cla1,[Lit|Path],PathLim,Lem,Set)
    ),
    ( member(cut,Set) -> ! ; true ),
    prove(Cla,Path,PathLim,[Lit|Lem],Set).
prove([],_,_,_,_).
```

Automated Reasoning in Large Theories

- Prove goals automatically in large formal theories
  - ATP translation of MML (≈ 50k proofs today) *[Urban03,...]*
  - Isabelle/HOL (≈ 60k proofs today) *[Paulson05,Blanchette]*
  - HOL Light/Flyspeck (≈ 30k proofs today) *[KU12]*
  - More proof assistant corpora: HOL4, ACL2, Coq
- Useful for ITP
  - Sledgehammer, HOL(y)Hammer, MizAR
  - But needs Reconstruction



Existing Proof Reconstruction

- General ATP search tactics producing ITP proof objects:
  - Metis (Isabelle, HOL4)
  - MESON, Prover9 (HOL Light)
  - Mizar `by`
- Parse TSTP/SMT proofs
  - Create subgoals that match ATP intermediate steps
  - Automatically solve all simple subgoals
  - Skolemization of type variables is an issue
- The smarter ATPs we can integrate in ITPs, the better
  - Not just for the Hammers
- Need for speed
  - Thousands of reconstruction steps in ITP projects
- ATP proof search blowup

Rewriting leanCoP in OCaml

- Parts of the Prolog technology missing in functional languages
- Use an explicit stack for keeping track of the current proof state (including the trail of variable bindings)
  - In the main `prove` function we add explicit arguments: `stack` (stack), `subst` (trail) and `off` (offset in the trail)
  - The stack keeps a list of tuples that are given as arguments to the recursive invocations of `prove`
  - When a proof is found, the exception ‘Solved’ is raised and the function exits with this exception.
- An alternative would be to use the continuation passing style

OCaml code (No history)

```
let rec prove path lim lem stack = function (lit :: cla) ->
  if not (exists2 eq path (lit :: cla)) then
  let neglit = negate lit in
  if not (exists (substeq lit) lem && (prove path lim lem stack cla; cut)) then
  if not (fold_left (fun sf plit -> sf ||
    try (unify_lit neglit plit; prove path lim (lit :: lem) stack cla; cut)
    with Unify -> sf) false path) then
  let iter_fun (lit2, cla2, ground) =
    if lim > 0 || ground then
    try let cla1 = unify_rename (snd lit) (lit2, cla2) in
    prove (lit :: path) (lim - 1) lem ((if cut then lim else -1),
    path, lim, lit :: lem, cla) :: stack) cla1 with Unify -> () in
  try iter iter_fun (try assoc neglit lits with Not_found -> [])
  with Cut n -> if n = lim then () else raise Cut n)))
| [] -> match stack with
      (ct, path, lim, lem, cla) :: t ->
              prove path lim lem t cla; if ct > 0 raise (Cut ct)
    | [] -> raise Solved;;
```

Differences

- Implementation of Prolog cut (!) in OCaml:
  - different mechanism in each of the three cases
- The Prolog code is elegant
  - But the OCaml code is a bit more efficient.
- A simple `List.exists` call is enough for finding a lemma
  - no need to backtrack
- For equality checking and unification under substitution we reuse MESON code:
  - substitutions as association lists
  - applications of substitutions are delayed until an equality check or a unification step

Eval I: HOL Light MESON calls without splitting (872 goals, 5s) [1]

Prover	Theorem (%)	Unique
OcaML-leanCoP (cut)	759 (87.04)	2
OcaML-leanCoP (nocut)	759 (87.04)	2
Prolog-leanCoP (cut)	752 (86.23)	0
Prolog-leanCoP (nocut)	751 (86.12)	0
Metis (2.3)	708 (81.19)	26
Meson	683 (78.32)	4
any	832 (95.41)	

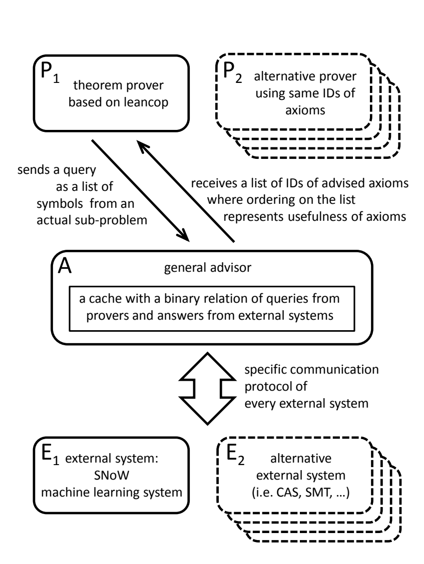
[1] Evaluation outside HOL Light

Reconstruction of leanCoP proofs in HOL Light

- Transformation from HOL to FOL and clausification
  - Tactics reuse Harrison's MESON code
  - Needs to preserve leanCoP's goal-directed approach
  - The conjecture is separated from the axioms
- All transformations are done on the CNF rather than DNF
  - The two are dual
- MESON's proof reconstruction needs modification for the use of lemmas

Very secure HOL Light certification of leanCoP proofs

Learning: MaLeCoP



FEMaLeCoP: Advice Overview and Used Features

- Advise the:
  - **selection of clause for every tableau extension step**
- Proof state: weighted vector of symbols (or terms)
  - extracted from all the literals on the active path
  - Frequency-based weighting (IDF)
  - Simple decay factor (using maximum)
- Consistent clausification
  - formula `?[X]: p(X)` becomes `p('skolem(?[A]:p(A),1)')`
- Advice using custom sparse naive Bayes
  - association of the features of the proof states
  - with contrapositives used for the successful extension steps

FEMaLeCoP: Data Collection and Indexing

- Slight extension of the saved proofs
  - Training Data: pairs (path, used extension step)
- External Data Indexing (incremental)
  - `te_num`: number of training examples
  - `pf_no`: hashtable from features to number of occurrences $\in \mathbb{Q}$
  - `cn_no`: hashtable from contrapositives to numbers of occurrences
  - `cn_pf_no`: hashtable of maps of cn/pf co-occurrences
- Problem Specific Data
  - Upon start FEMaLeCoP reads
    - **only current-problem relevant** parts of the training data
  - cn_no and cn_pf_no filtered by contrapositives in lit matrix
  - pf_no and cn_pf_no filtered by possible features in the problem

Naive Bayes

If more than one possible extension step

Estimate relevance of each contrapositive cn by

$$\sigma_1 \ln t + \sum_{f \in (\overline{f} \cap \overline{s})} i(f) \ln \frac{\sigma_2 s(f)}{t} + \sigma_3 \sum_{f \in (\overline{f} - \overline{s})} i(f) + \sigma_4 \sum_{f \in (\overline{s} - \overline{f})} i(f) \ln(1 - \frac{s(f)}{t})$$

where

- $\overline{f}$ are the features of the path
- $\overline{s}$ are the features that co-occurred with cn
- $t = cn\_no(cn)$
- $s = cn\_fp\_no(cn)$
- i is the IDF
- σ_* are experimentally chosen parameters

It cannot work?

Inference speed ...

It cannot work? But it does!

Inference speed … drops to about 40%, but:

Prover	Proved (%)
OCaml-leanCoP	574 (27.6%)
FEMaLeCoP	635 (30.6%)
together	664 (32.0%)

(MPTP bushy problems, 60 s)

Summary

- OCaml version of leanCoP
  - outperforms Metis and MESON, sometimes very significantly
- Reconstruction of leanCoP proofs in HOL Light
  - Useful as reconstruction component of HOL(y)Hammer
  - Certification of leanCoP TPTP proofs
- Learning
  - Three levels of indexing
  - Proper integration
- Future Work:
  - Strategies, more evaluation, more HOL-based ITPs
  - Intuitionistic version
  - More learning algorithms