Implementing ATP Systems

Unit 11: Optimizations and Extensions

Jens Otten

University of Potsdam

Outline

1. Optimizations
2. leanCoP 2.0
3. Performance
4. Extensions
5. Summary and Further Research

The Basic Connection Calculus

- Axiom $$\frac{}{\{\}, M, Path}$$
- Start Rule $$\frac{C_2, M, \{\}}{\varepsilon, M, \varepsilon} \qquad C_2 \text{ is copy of } C_1 \in M$$
- Reduction Rule $$\frac{C, M, Path \cup \{L_2\}}{C \cup \{L_1\}, M, Path \cup \{L_2\}} \qquad \{\sigma(L_1), \sigma(L_2)\} \text{ is a connection}$$
- Extension Rule $$\frac{C_2 \setminus \{L_2\}, M, Path \cup \{L_1\} \qquad C, M, Path}{C \cup \{L_1\}, M, Path} \qquad \begin{array}{l} C_2 \text{ is copy of } C_1 \in M,\ L_2 \in C_2, \\ \{\sigma(L_1), \sigma(L_2)\} \text{ is a connection} \end{array}$$
- Connection proof

  $\Leftrightarrow$ $\exists$ derivation for $\varepsilon, M, \varepsilon$ in which all leaves are axioms.

leanCoP 1.0: Implementing the Basic Calculus

```
prove(M,I) :- append(Q,[C|R],M), \+member(-_,C),
 append(Q,R,S), prove([!],[[-!|C]|S],[],I).
prove([],_,_,_).
prove([L|C],M,P,I) :- (-N=L; -L=N) -> (member(N,P);
 append(Q,[D|R],M), copy_term(D,E), append(A,[N|B],E),
 append(A,B,F), (D==E -> append(R,Q,S); length(P,K), K<I,
 append(R,[D|Q],S)), prove(F,S,[L|P],I)), prove(C,M,P,I).
```

- Connection driven proof search.
- "append technique" selects clause/literal from matrix/clause.
- Only positive start clauses are considered.
- Only copies of first-order clauses are made.
- Path limit only checked for connections to first-order clauses.

Optimizations

- Goal: Select a few highly effective techniques for pruning the search space in the basic connection calculus.

Definitional clausal form translation.	+
Regularity.	++
Lemmata.	+
Restricted backtracking.	+++
“Lean Prolog technology”.	+
Strategy scheduling.	++

(+: modest effect; ++: significant effect; +++: strong effect)

Definitional Clausal Form Translation

- Idea: Introduce definitions for certain subformulae.
- Example: $(A \vee \neg A) \wedge (B \vee \neg B) \wedge (C \vee \neg C)$

  Standard translation: $\begin{bmatrix} A \\ B \\ C \end{bmatrix} \begin{bmatrix} A \\ B \\ \neg C \end{bmatrix} \begin{bmatrix} A \\ \neg B \\ C \end{bmatrix} \begin{bmatrix} A \\ \neg B \\ \neg C \end{bmatrix} \begin{bmatrix} \neg A \\ B \\ C \end{bmatrix} \begin{bmatrix} \neg A \\ B \\ \neg C \end{bmatrix} \begin{bmatrix} \neg A \\ \neg B \\ C \end{bmatrix} \begin{bmatrix} \neg A \\ \neg B \\ \neg C \end{bmatrix}$

  Definitional translation:

  $((A \vee \neg A) \Rightarrow P) \wedge (B \vee \neg B) \Rightarrow Q) \wedge (C \vee \neg C) \Rightarrow R)) \Rightarrow (P \wedge Q \wedge R)$

  $$\begin{bmatrix} [A & \neg A] \\ [B & \neg B] \\ [C & \neg C] \end{bmatrix} \quad \leadsto \quad \begin{bmatrix} P \\ Q \\ R \end{bmatrix} \begin{bmatrix} \neg P \\ A \end{bmatrix} \begin{bmatrix} \neg P \\ \neg A \end{bmatrix} \begin{bmatrix} \neg Q \\ B \end{bmatrix} \begin{bmatrix} \neg Q \\ \neg B \end{bmatrix} \begin{bmatrix} \neg R \\ C \end{bmatrix} \begin{bmatrix} \neg R \\ \neg C \end{bmatrix}$$

- Important: Minimize number of possible connections, i.e. minimize number of possible extension and reduction steps.
- Other translations (e.g. E, FLOTTER) do not work well.

Definitional Clausal Form Translation – Formal

- Let F be a formula in negation normal form and let $cla(D)$ be the standard transformation of formula D into clausal form.
- The definitional tuple $(F', \mathcal{D})$ of F, where $\mathcal{D}$ is a set of formulae, is inductively defined as follows:
  1. F is a literal: $(F, \{\})$ is the definitional tuple of F; otherwise
  2. F is of the form $A \vee B$ and F occurs within a conjunction and $(A', \mathcal{D}_A)$ and $(B', \mathcal{D}_B)$ are the definitional tuples of A and B: $(S(x_1, ..., x_n), \{\neg S(x_1, ..., x_n) \wedge A', \neg S(x_1, ..., x_n) \wedge B'\} \cup \mathcal{D}_A \cup \mathcal{D}_B)$ is the definitional tuple of F, where S is a new predicate symbol and $x_1, ..., x_n$ are the variables occurring in $(A \vee B)$; otherwise
  3. F is of the form $A \circ B$ with $\circ \in \{\wedge, \vee\}$ and if $(A', \mathcal{D}_A)$ and $(B', \mathcal{D}_B)$ are the definitional tuples of A and B: $(A' \circ B', \mathcal{D}_A \cup \mathcal{D}_B)$ is the definitional tuple of F.
- $F' \vee cla(D_1) \vee \ldots \vee cla(D_n)$ is definitional clausal form of F where $(F', \{D_1, ..., D_n\})$ is the definitional tuple of F.
- A formula F is valid iff its definitional clausal form is valid.

Regularity and Lemmata

- Regularity: No (ground) literal occurs more than once in the active path (in the current branch of the tableau).

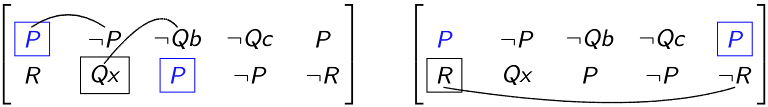
  Impose the following restriction on reduction/extension rule:
  $\forall L' \in C \cup \{L\} :\ \sigma(L') \notin \sigma(Path)$ (and L' is ground).

- Lemmata: If a branch with (ground) literal L has been closed, all branches containing L (below/to the right) can be closed.

  Add the following lemma rule to the connection calculus:

  Lemma rule $\dfrac{C, M, Path}{C \cup \{L\}, M, Path}$ and L is a lemma in that branch.

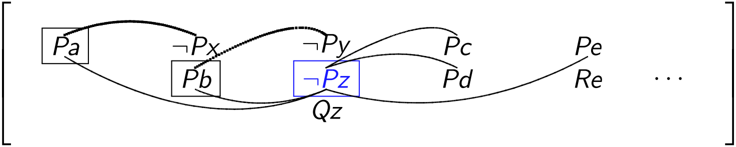
- Example (regularity and lemmata):



$$\begin{bmatrix} \boxed{P} & \neg P & \neg Qb & \neg Qc & P \\ R & \boxed{Qx} & \boxed{P} & \neg P & \neg R \end{bmatrix} \qquad \begin{bmatrix} P & \neg P & \neg Qb & \neg Qc & \boxed{P} \\ \boxed{R} & Qx & P & \neg P & \neg R \end{bmatrix}$$

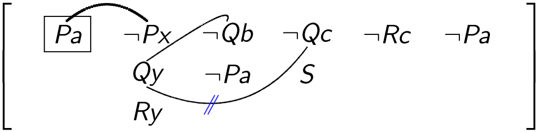
Restricted Backtracking

- Fact: In contrast to saturation-based calculi (e.g. resolution), connection calculi are not proof confluent.
- A significant amount of backtracking is required when
  - selecting start clause C_1 in start rule,
  - selecting literal L_2 in the reduction rule,
  - selecting clause C_1 and literal L_2 in the extension rule.
- Example:



Restricted Backtracking (cont.)

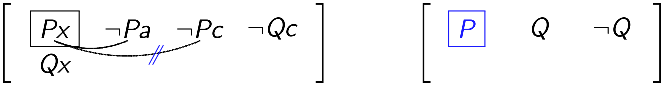
- Idea: Reduce amount of backtracking by restricting backtracking for start, reduction and extension rule.
- Restricted backtracking for start rule:
  → do not consider alternative start clauses.
- Restricted backtracking for reduction/extension rule:
  → once a branch has successfully been closed, do not consider alternative rule applications anymore.
- Example:



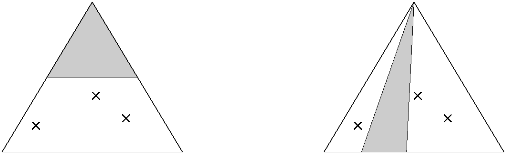
$$\left[\begin{array}{cccccc} \boxed{Pa} & \neg Px & \neg Qb & \neg Qc & \neg Rc & \neg Pa \\ & Qy & \neg Pa & S & & \\ & Ry & & & & \end{array}\right]$$

Restricted Backtracking (cont.)

- Correct, but not complete. Consider, e.g., for extension and start rule, respectively:



- Very successful in practice, in particular for problems containing many axioms, e.g. equality axioms.
- Illustration of complete proof search (left) and proof search using restricted backtracking (right).



leanCoP 2.0 ("Garlic"): Implementing Optimizations

- Definitional clausal from translation in module `def_mm.pl`.
- leanCoP v1.0: Implements basic calculus.
- leanCoP v2.0 "Garlic" (minimal code: 555 bytes):

```
prove(I,S) :- \+member(scut,S) -> prove([-(#)],[],I,[],S) ;
    lit(#,C,_) -> prove(C,[-(#)],I,[],S).
prove(I,S) :- member(comp(L),S), I=L -> prove(1,[]) ;
    (member(comp(_),S);retract(p)) -> J is I+1, prove(J,S).
prove([],_,_,_,_).
prove([L|C],P,I,Q,S) :- \+ (member(A,[L|C]), member(B,P),
    A==B), (-N=L;-L=N) -> ( member(D,Q), L==D ;
    member(E,P), unify_with_occurs_check(E,N) ; lit(N,F,H),
    (H=g -> true ; length(P,K), K<I -> true ;
    \+p -> assert(p), fail), prove(F,[L|P],I,Q,S) ),
    (member(cut,S) -> ! ; true), prove(C,P,I,[L|Q],S).
```

Lean Prolog Technology

- Use Prolog's indexing mechanism to quickly find connections.

  The set of clauses M is written into Prolog's database: $\forall$ clauses $C \in M$ and $\forall$ literals `L`$\in C$ the fact `lit(L,C1,Grnd)` is stored, where `C1`$= C \setminus \{$`L`$\}$ and `Grnd` is g iff C is ground.

  Example: The clause $\{a(x), \neg b, c\}$ is stored as
  `lit(a(X),[-b,c],n). lit(-b,[a(X),c],n). lit(c,[a(X),-b],n).`

- Main predicate: `prove(Cla,Path,PathLim,Lem,Set)`.
  - Realizes the proof search: `Cla` and `Path` are Prolog lists and represent the branch $Cla, M, Path$ in the connection calculus.
  - `Lem` and `Set` represent the lemma literals and the settings.
  - `PathLim` is the maximum length of the active path (used for iterative deepening to achieve completeness).
  - The substitution σ is stored implicitly by Prolog.

The leanCoP 2.0 Source Code

```
prove([],_,_,_,_).

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,Cla1,Grnd1),
         ( 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).
```

- The complete leanCoP core code (v2.0, without start rule).

The leanCoP 2.0 Source Code (cont.)

```
(a)  prove(PathLim,Set) :-
(b)      \+member(scut,Set) -> prove([-(#)],[],PathLim,[],Set) ;
(c)      lit(#,C,_) -> prove(C,[-(#)],PathLim,[],Set).
(d)  prove(PathLim,Set) :-
(e)      member(comp(Limit),Set), PathLim=Limit -> prove(1,[]) ;
(f)      (member(comp(_),Set);retract(pathlim)) ->
(g)      PathLim1 is PathLim+1, prove(PathLim1,Set).
```

- The leanCoP code of the start rule with iterative deepening and restricted backtracking.
- The special literal # has to be added to all possible start clauses (i.e. to positive clauses or conjecture clauses).
- leanCoP is invoked with, e.g., `prove(1,[cut]).`, where the formula is stored in Prolog's database using the `lit` predicate.

Strategy Scheduling

- Different settings control the proof search of the core prover.
- Possible settings $\subseteq$ {nodef,def,conj,cut,scut,reo(I),comp(I)}:

nodef/def: standard/definitional clausal form translation is used (default: definitional translation only for conjecture).

conj: start with conjecture clauses (default: positive clauses).

cut/scut: restricted backtracking is used for reduction & extension rule / start rule (default: no restricted backtracking).

reo(I): reorder clauses I times (default: no reordering).

comp(I): complete search strategy when proof depth I is reached.

- Different settings are consecutively invoked by shell script.
- The used fixed strategy scheduling preserves completeness.

Analyzing Restricted Backtracking

- TPTP library v3.7.0: 5051 FOF problems; 600 sec time limit.

Domain	# of proofs	1st start clause	Essent. steps	Non-es. steps	Essent. proofs	Non-es. proofs
CSR	93	87	605	57	75	18
SET	193	141	2005	227	117	76
SWC	14	14	54	0	14	0
SWV	160	117	1297	51	135	25
SYN	204	190	1734	41	189	15
Total	1256	981	19403	2485	882	374
[%]	100%	78%	89%	11%	70%	30%

- 1st start clause: first start clause is used in the final proof.
- Essential steps: proof steps that did not require backtracking.
- Essential proofs: proofs that only contain essential steps.
- Observation: about 90%/70% essential proof steps/proofs.

Performance of Different Clausal Form Translations

- TPTP library v3.7.0: 5051 FOF problems; 600 sec time limit.

System	TPTP	Flotter	E	leanCoP 2.0		
Version	3.7.0	3.0	1.0	“def”	“nodef”	(default)
Proved	1205	1365	1369	1486	1514	1560
[%]	24%	27%	27%	29%	30%	31%
Rating						
0.00...0.24	53%	56%	58%	62%	60%	62%
0.25...0.49	39%	47%	47%	52%	51%	53%
0.50...0.74	10%	16%	16%	17%	22%	24%
0.75...1.00	1%	1%	1%	1%	2%	2%

- leanCoP 2.0 with strategy “`[cut,comp(7)]`”.
- Using clausal form translations of TPTP2X/SPASS(Flotter)/E and “def”/“nodef”/default translation of leanCoP 2.0.
- Best: leanCoP’s default translation (“def” only for conjecture).

Performance of Different Techniques

- TPTP library v3.7.0: 5051 FOF problems; 600 sec time limit.

	leanCoP 1.0	basic	define	regular	restrict	leanCoP 2.0
Proved	1105	1086	1094	1256	1560	1797
[%]	22%	22%	22%	25%	31%	36%
Average time	12.2 s	2.6 s	2.8 s	3.6 s	2.6 s	6.1 s
Rating 0.0	458	450	446	501	531	554
Rating >0.0	647	636	648	755	1029	1243
No equality	532	526	515	552	587	616
With equality	573	560	579	704	973	1181

- “basic”: using lean Prolog technology.
  “define”: plus (default) definitional clausal translation.
  “regular”: plus regularity and lemmata (leanCoP settings: `[]`).
  “restrict”: plus restricted backtracking (`[cut,comp(7)]`).
- leanCoP 2.0: plus strategy scheduling.

Performance of leanCoP 2.1 and Other ATP Systems

- TPTP library v3.7.0: 5051 FOF problems; 600 sec time limit.

System	leanTAP	Otter	Prover9	SNARK	leanCoP	E
Version	2.3	3.3	2009-02A	08/07	2.1	1.0
Proved	404	1389	1664	1735	1893	2541
[%]	8%	27%	33%	34%	37%	50%
Rating						
0.00...0.24	17%	64%	61%	69%	68%	75%
0.25...0.49	18%	47%	71%	68%	68%	92%
0.50...0.74	2%	3%	27%	21%	35%	74%
0.75...1.00	0%	0%	1%	1%	5%	12%

(rating 0.0: easy; rating 1.0: very difficult)

- leanCoP 1.0/2.0: 1105/1797 problems (SETHEO 3.3: 1296).
- leanCoP 2.1 accepts TPTP syntax and outputs a proof.
- Ranked 3rd at CADE system competition 2010 (CASC-J5) of provers that output a proof in the first-order division (FOF).

The ileanCoP 1.2 Source Code

```
prove(I,S) :- ( \+member(scut,S) ->
    prove([(-(#)):(-[])],[],I,[],[Z,T],S) ;
    lit((#):_,G:C,_) -> prove(C,[(-(#)):(-[])],I,[],[Z,R],S),
    append(R,G,T) ), check_addco(T), prefix_unify(Z).
prove(I,S) :- member(comp(L),S), I=L -> prove(1,[]) ;
    (member(comp(_),S);retract(p)) -> J is I+1, prove(J,S).
prove([],_,_,_,[[],[]],_).
prove([L:U|C],P,I,Q,[Z,T],S) :- \+ (member(A,[L:U|C]),member(B,P),
    A==B), (-N=L;-L=N) -> ( member(D,Q), L:U==D, X=[], O=[] ;
    member(E:V,P), unify_with_occurs_check(E,N),
    \+ \+ prefix_unify([U=V]), X=[U=V], O=[] ;
    lit(N:V,M:F,H), \+ \+ prefix_unify([U=V]),
    (H=g -> true ; length(P,K), K<I -> true ;
    \+ p -> assert(p), fail), prove(F,[L:U|P],I,Q,[W,R],S),
    X=[U=V|W], append(R,M,O) ), (member(cut,S) -> ! ; true),
    prove(C,P,I,[L:U|Q],[Y,J],S), append(X,Y,Z), append(J,O,T).
```

- ileanCoP 1.2 core prover plus prefix unification (23 more lines).

Performance of ileanCoP 1.2

- TPTP library v3.3.0: 3644 FOF problems; 600 sec time limit.

System	JProver	ileanTAP	ft (C)	ileanSeP	Imogen	ileanCoP
Version	11-2005	1.17	1.23	1.0	2.1	1.2
Proved	186	255	262	303	842	1127
[%]	5%	7%	7%	8%	23%	31%
Rating						
0.00...0.24	13.1%	15.3%	16.1%	17.1%	48.3%	54.5%
0.25...0.49	0.4%	4.5%	5.0%	9.1%	20.1%	34.1%
0.50...0.74	0.0%	1.2%	0.2%	0.0%	4.2%	20.2%
0.75...1.00	0.0%	0.3%	0.2%	0.0%	0.5%	2.3%

- CADE system competition 2007 (CASC-21): ileanCoP proved more problems than some classical provers and proved two problems for which Vampire did not find a classical proof.
- Intuitionistic problem library ILTP (Raths/Otten/Kreitz ’07).

leanCoP-Ω: First-Order Logic with Linear Arithmetic

```
prove([],_,_,_,_,[],Eq,Eq).
prove([L|C],P,I,Q,S,Pr,Eq,Eq1) :-
    Pr=[[[R|F]|Pr1]|Pr2], \+ (member(A,[L|C]), member(B,P), A==B),
    (-N=L;-L=N) -> ( member(D,Q), L==D, F=[], Pr1=[], Eq2=Eq ;
    member(E,P), unify_with_arith(E,N,EqU,S), append(EqU,Eq,Eq2),
    omega(Eq2), F=[], Pr1=[] ; lit(N,E,F,H), unify_with_arith(E,N,EqU,S),
    append(EqU,Eq,Eq3), omega(Eq3), (H=g -> true ; length(P,K), K<I ->
    true ; \+ pathlim -> assert(pathlim), fail),
    prove(F,[L|P],I,Q,S,Pr1,Eq3,Eq2) ;
    (L=(_=_);-(_=_)=L;L=(_<_);-(_<_)=L) -> (leanari(L) ->  Eq2=Eq, F=[],
    Pr1=[] ; member(eq(_),S), path_eq(P,L,EqP), (omega([EqP|Eq]) ->
    Eq2=[EqP|Eq], F=[], Pr1=[] ; member(eq(2),S), lit(_,R,F,H),
    (R=(_=_);-(_=_)=R;R=(_<_);-(_<_)=R), (H=g -> true ; length(P,K),
    K<I -> true ; \+ pathlim -> assert(pathlim), fail),
    prove([R|F],[L|P],I,Q,S,Pr1,Eq,Eq2) ) ) ), (var(R) -> R=N ; true),
    ( member(cut,S) -> ! ; true ), prove(C,P,I,[L|Q],S,Pr2,Eq2,Eq1).
```

+ Omega test system (Pugh ’92) for linear integer arithmetic.

Performance of leanCoP-Ω

- CADE system competition 2010 (CASC-J5):
  New division (TFA) containing problems in first-order logic with linear integer arithmetic.
- CASC-J5, TFA division: 75 problems; 300 sec time limit.

System	leanCoP-Ω	SPASS+T	...	...	...
Version	0.1	2.2.12	...	...	...
Proved	64	62	46	39	35
[%]	85%	83%	61%	52%	47%

- leanCoP-Ω still in a very experimental state.
- Joint work with Holger Trölenberg (interface to Omega) and Thomas Raths (parsing of type information and testing).

Summary

- Connection calculus well suited to automate logic reasoning in classical and non-classical logics.
  - leanCoP currently fastest connection/tableau prover.
  - ileanCoP currently fastest prover for intiutionistic logic.
  - leanCoP-Ω CASC-winner for linear integer arithmetic.

  Web: `www.leancop.de`

- Restricted backtracking is single most effective technique to reduce the search space in connection calculi.

- First-order logic = propositional logic + term unification.
- Non-classical logics = classical logic + prefix unification.

Further Research

- Develop non-clausal connection calculus that does require any translation steps into clausal form.
- Improving algorithm for prefix unification.
- Extend calculus/implementation to other non-classical logics.
  - Modal logics, e.g. D, D4, K, K4, T, S4, S5.
  - Fragments of linear logic, e.g. multiplicative fragment.
- Build problem libraries for other non-classical logics, e.g. for some first-order modal logics (QMLTP library).