Automated Theorem Proving

Lecture 2

Cezary Kaliszyk

May 26, 2020

Outline

Summary of Previous Lecture

- What is automated theorem proving
- Common uses and differences to other systems
- Tableaux: First version
- LeanTaP

Today

- Full details of Tableaux in functional style
- leanCoP
- Optimizations possible in lean tableaux
- Some Variants of the calculus

leanTaP: Reasons for inefficiency

The code of leanTaP was elegant: PTTP (Prolog Technology Theorem Proving) allowed for very short code and half of the implementation details could not be specified but handled by the programming language

But the efficiency of the very short code was limited. Not so much by Prolog, but by the calculus

We know from logic, that we can convert any formula to CNF and then things become easier, maybe we can do this here and introduce optimizations in a simpler problem?

So we will first introduce connection tableaux: Something between the tableaux that we have seen previously and resolution.

Resoultion vs Tableaux

Historical dispute since the times of Gentzen and Hilbert

- Today two communities: Resolution (and res-style) and Tableaux

Possible answer: What is better in practice?

- Say the competitions or on existing ITP libraries?
- Since the late 90s: resolution (superposition) has been winning

But still so far from humans?

- A human can more naturally think in tableaux style as the proof state is much smaller. Sometimes proofs are a bit bigger, but it does not matter if we can guide it better.
- Tableaux proofs more readable (closer to math books)
- Also more useful for machine learning (not this course)

leanCoP: Lean Connection Prover [Otten 2010]

Connected tableaux calculus

- Goal oriented, good for large theories

Regularly beats resolution-style Metis and Prover9 in CASC

(CADE ATP Systems Competition)

- despite their much larger implementation

Compact Prolog implementation, easy to modify

- (But we will look at the more efficient functional style one)
- Variants for other foundations: iLeanCoP, mLeanCoP
- First experiments with machine learning: MaLeCoP

Easy to imitate

- leanCoP tactic used in proof assistants

Lean connection Tableaux

The calculus deals with triples: The clause that we try to disprove, the original problem, and a path (which is a set of literals)

Very simple rules:

- Axiom means if we have the empty clause to disprove, we are done
- Extension unifies the current literal with a copy of an original clause (a bit like resolution)
- Reduction unifies the current literal with a literal on the path

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

$$\frac{C,\ M,\ Path \cup \{L_2\}}{C \cup \{L_1\},\ M,\ Path \cup \{L_2\}} \quad \textit{Reduction}$$

$$\frac{C_2 \setminus \{L_2\},\ M,\ Path \cup \{L_1\} \qquad C,\ M,\ Path}{C \cup \{L_1\},\ M,\ Path} \quad \textit{Extension}$$

Explanation (and justification) of the rules

In resolution calculus (known from the LICS course) we had resolution and factoring.

Here the resolution rule is weakened (we only try to resolve with the clauses in the original problem). This allows checking for much fewer unifications, and our proof state is rather small.

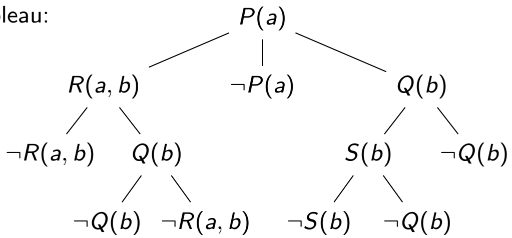
But the disadvantage is that instead of factoring we need a more complex rule. Namely, we remember a "path": All the literals on which we resolved. And if we encounter something that unifies with the path, we could have gotten rid of it in the same way, so we do not need to prove it. This rule takes care of factoring, but is in fact stronger than factoring.

Example lean connection proof

Clauses:

- $c_1 : P(x)$
- $c_2 : R(x, y) \vee \neg P(x) \vee Q(y)$
- $c_3 : S(x) \vee \neg Q(b)$
- $c_4 : \neg S(x) \vee \neg Q(x)$
- $c_5 : \neg Q(x) \vee \neg R(a, x)$
- $c_6 : \neg R(a, x) \vee Q(x)$

Tableau:



Observe that the substitutions have been applied implicitly, but they come from unification.

Note

leanCoP proofs as found in literature are usually presented in DNF rather than CNF, but as these are dual, we will only for one slide switch to the other representation. The reason is, that proofs can then be represented as somewhat more compact "matrices". This is only a historical distinction, no impact on learning is known and is presented here only as a curiosity.

More complex example

Consider the formula already in clausal normal form:

$$F = \forall x.(Q \wedge P(a) \wedge (\neg P(x) \vee \neg P(s^2x)) \wedge (\neg P(x) \vee P(sx) \vee \neg Q))$$

We will here present the matrix of the formula: Columns are the clauses, and entries are literals and the proof written as connections giving the substitution as is it derived

$$\left[[Q] \begin{bmatrix} \neg P(x') \\ P(sx') \\ \neg Q \end{bmatrix} [P(a)] \begin{bmatrix} \neg P(\bar{x}) \\ \neg P(s^2\bar{x}) \end{bmatrix} \right]$$

$$\sigma = \{\}$$

As we have a complete connection (all considered clauses have all literals connected) this corresponds to a closed tableaux, and means that the problem is proved.

More complex example

Consider the formula already in clausal normal form:

$$F = \forall x.(Q \wedge P(a) \wedge (\neg P(x) \vee \neg P(s^2x)) \wedge (\neg P(x) \vee P(sx) \vee \neg Q))$$

We will here present the matrix of the formula: Columns are the clauses, and entries are literals and the proof written as connections giving the substitution as is it derived

$$\begin{tikzcd} {[Q]} \arrow[r, bend right, "\textcircled{1}"'] & {\begin{bmatrix}\neg P(x') \\ P(sx') \\ \neg Q\end{bmatrix}} & {[P(a)]} & {\begin{bmatrix}\neg P(\bar{x}) \\ \neg P(s^2\bar{x})\end{bmatrix}} \end{tikzcd}$$

$$\sigma = \{\}$$

As we have a complete connection (all considered clauses have all literals connected) this corresponds to a closed tableaux, and means that the problem is proved.

More complex example

Consider the formula already in clausal normal form:

$$F = \forall x.(Q \wedge P(a) \wedge (\neg P(x) \vee \neg P(s^2 x)) \wedge (\neg P(x) \vee P(sx) \vee \neg Q))$$

We will here present the matrix of the formula: Columns are the clauses, and entries are literals and the proof written as connections giving the substitution as is it derived

$$\begin{tikzcd}
 & \neg P(x') \arrow[dr, bend left, "\textcircled{2}"] & & \\
{[Q]} \arrow[dr, bend right, "\textcircled{1}"'] & P(sx') & {[P(a)]} & \neg P(\bar{x}) \\
 & \neg Q & & \neg P(s^2 \bar{x})
\end{tikzcd}$$

$$\sigma = \{\}$$

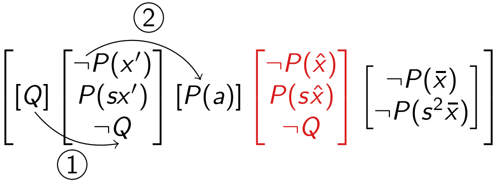
As we have a complete connection (all considered clauses have all literals connected) this corresponds to a closed tableaux, and means that the problem is proved.

More complex example

Consider the formula already in clausal normal form:

$$F = \forall x.(Q \wedge P(a) \wedge (\neg P(x) \vee \neg P(s^2x)) \wedge (\neg P(x) \vee P(sx) \vee \neg Q))$$

We will here present the matrix of the formula: Columns are the clauses, and entries are literals and the proof written as connections giving the substitution as is it derived



$$\sigma = \{x' \mapsto a\}$$

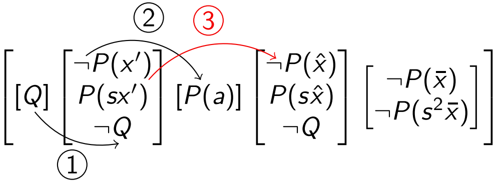
As we have a complete connection (all considered clauses have all literals connected) this corresponds to a closed tableaux, and means that the problem is proved.

More complex example

Consider the formula already in clausal normal form:

$$F = \forall x.(Q \wedge P(a) \wedge (\neg P(x) \vee \neg P(s^2x)) \wedge (\neg P(x) \vee P(sx) \vee \neg Q))$$

We will here present the matrix of the formula: Columns are the clauses, and entries are literals and the proof written as connections giving the substitution as is it derived



$$\sigma = \{x' \mapsto a\}$$

As we have a complete connection (all considered clauses have all literals connected) this corresponds to a closed tableaux, and means that the problem is proved.

More complex example

Consider the formula already in clausal normal form:

$$F = \forall x.(Q \land P(a) \land (\neg P(x) \lor \neg P(s^2x)) \land (\neg P(x) \lor P(sx) \lor \neg Q))$$

We will here present the matrix of the formula: Columns are the clauses, and entries are literals and the proof written as connections giving the substitution as is it derived

$$\begin{tikzcd}
 & \neg P(x') \arrow[dr, bend left, "\textcircled{2}"] & & \neg P(\hat{x}) & \neg P(\bar{x}) \\
{[Q]} \arrow[dr, bend right, "\textcircled{1}"'] & P(sx') \arrow[urr, bend left, "\textcircled{3}"] & {[P(a)]} & P(s\hat{x}) \arrow[r, bend right, red, "\textcircled{4}"'] & \neg P(s^2\bar{x}) \\
 & \neg Q & & \neg Q &
\end{tikzcd}$$

$$\sigma = \{x' \mapsto a, \color{red}{\hat{x} \mapsto sx'}\}$$

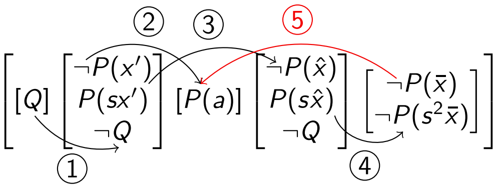
As we have a complete connection (all considered clauses have all literals connected) this corresponds to a closed tableaux, and means that the problem is proved.

More complex example

Consider the formula already in clausal normal form:

$$F = \forall x.(Q \wedge P(a) \wedge (\neg P(x) \vee \neg P(s^2x)) \wedge (\neg P(x) \vee P(sx) \vee \neg Q))$$

We will here present the matrix of the formula: Columns are the clauses, and entries are literals and the proof written as connections giving the substitution as is it derived



$$\sigma = \{x' \mapsto a, \hat{x} \mapsto sx', \bar{x} \mapsto x'\}$$

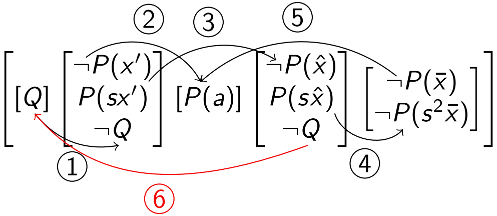
As we have a complete connection (all considered clauses have all literals connected) this corresponds to a closed tableaux, and means that the problem is proved.

More complex example

Consider the formula already in clausal normal form:

$$F = \forall x.(Q \wedge P(a) \wedge (\neg P(x) \vee \neg P(s^2x)) \wedge (\neg P(x) \vee P(sx) \vee \neg Q))$$

We will here present the matrix of the formula: Columns are the clauses, and entries are literals and the proof written as connections giving the substitution as is it derived



$$\sigma = \{x' \mapsto a, \hat{x} \mapsto sx', \bar{x} \mapsto x'\}$$

As we have a complete connection (all considered clauses have all literals connected) this corresponds to a closed tableaux, and means that the problem is proved.

The proof search

We have seen how proofs in the calculus work. But what about searching for the proofs?

The calculus is not proof confluent: Making a wrong decision may mean that we will not find a proof that exists. So we need to backtrack to the different options. To do this various limits are introduced. For example a depth limit, that says we first search for tableaux of depth at most 1, then at most 2 etc.

This already gives a sound and complete proof search procedure for FOL. Lets look at a slightly simplified implementation.

leanCoP: CPS Implementation

```
prove [] path lim sub alt rem = rem sub alt
prove (lit : cla) path lim sub alt rem = reduce path where
  reduce (plit : path) =
    let alt1 = reduce path
    in case unify sub (negate lit) plit of
      Nothing -> alt1
      Just sub1 -> prove cla path lim sub1 alt1 rem
  reduce [] = extend (unifyDB sub (negate lit))

  extend ((sub1, cla1) : contras) =
    let alt1 = extend contras
    in if lim <= 0 then alt1
       else
         let rem1 sub alt = prove cla path lim sub alt rem
         in prove cla1 (lit : path) (lim - 1) sub1 alt1 rem1
  extend [] = alt
```

leanCoP: Implementation Explanation

The prove function takes the clause, the path, a depth limit, a substitution, list of alternatives, and a list of remaining things to do.

If the clause is empty, the remaining things to do (further calls to prove passed as a function) are called.

Otherwise reductions with all items of the path are tried. If unification succeeds but the proof failed further reduction are passed as alternatives.

When no more reductions are possible, extensions with clauses in the problem ("contrapositives") are tried with a decreased depth limit, alternatives specified as before, and the remaining thing to do being proving the rest of the clause

leanCoP: More implementation optimizations

The majority of the time is spent looking at what we can unify. For this efficient indexing structures have been proposed.

We will very shortly introduce two kinds of indexing structures, that allow us to store our original clauses and quickly extract the ones that unify with the given literal

We will see more indexing structures and more details on them in resolution style calculi, where these need to be modified throughout the proof.

Substitution Trees

A substitution tree is a term indexing structure that stores terms in the form of a tree, where nodes are terms and edges are substitutions.

The tree starts with a variable as the root, and stored terms are obtained by subsequently applying substitutions going from the root to the leaf

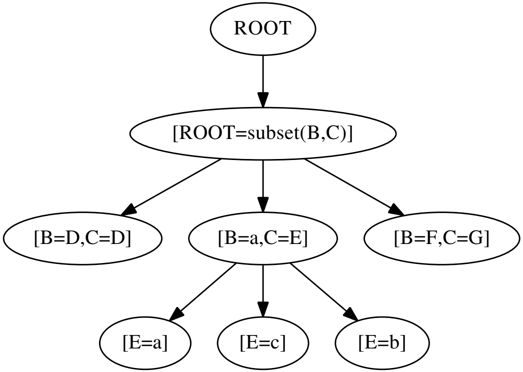
Finding terms in a substitution tree requires checking compatibility with the substitution and inserting terms may involve factoring a substitution

The further details are only given by example

Substitution Tree example

The tree that contains the terms on the left is as follows:

```
subset(A,B)
subset(a,b)
subset(a,c)
subset(C,C)
subset(a,a)
```



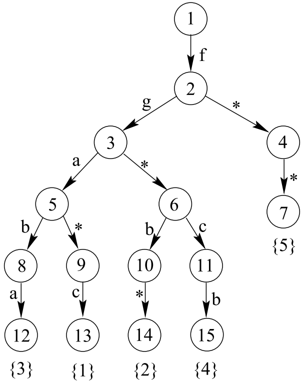
Discrimination Tree

A discrimination tree is an indexing structure that also stores terms and allows the recovery of terms that unify or match a given one

It is a variant of a trie, where the terms are input in the order of their reading. This means that a term $f(g(a, b), a)$ is represented as a path from the root $f - g - a - b - a$. Additionally variables are represented as $*$ and when looking up terms or inserting them subparts of terms need to be skipped.

Discrimination Tree example

For terms $f(g(a,*),c)$, $f(g(*,b),*)$, $f(g(a,b),a)$, $f(g(*,c),b)$, and $f(*,*)$



Restricted Backtracking

The connection calculus that we have looked at so far is complete. This is good theoretically. However, to explore all possible options many similar branches are searched.

To achieve better performance on actual benchmarks often incomplete variants are considered. For the connection calculus Otten proposed "restricted backtracking", an incomplete strategy that in practical benchmarks finds twice more proofs, but definitely can miss some more intricate settings.

The idea is that whenever we have managed to successfully close a literal by some subtree (sub-tableaux), we do not try to close it again in a different way. This may be incomplete, as closing it in a different way may give rise to a different substitution. But this happens very rarely in practice.

LeanCoP Variant: Non Clausal

One exciting feature of the Connection Calculus, is that it does not require complete CNF. It is enough to have a formula in prenex/negation normal form and skolemized, but not in CNF. Consider:

$$Q \wedge P(a) \wedge \forall x.(\neg P(x) \vee (\neg P(s^2x) \wedge (P(sx) \vee \neg Q)))$$

Then the matrix of the formula has columns for conjuncts and rows for disjunts. And we can still build connections between them like before building a substitution:

$$\begin{bmatrix} [Q][P(a)] \begin{bmatrix} \neg P(x') \\ \begin{bmatrix} [\neg P(s^2x')] \begin{bmatrix} P(sx') \\ \neg Q \end{bmatrix} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

$$\sigma = \{\}$$

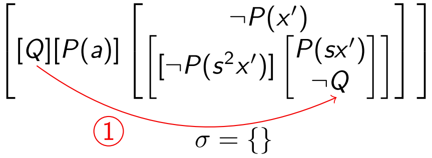
Note that there are proofs that are linearly shorter in the non-clausal variant (but implementation is more complex and therefore so far comparably performant)

LeanCoP Variant: Non Clausal

One exciting feature of the Connection Calculus, is that it does not require complete CNF. It is enough to have a formula in prenex/negation normal form and skolemized, but not in CNF. Consider:

$$Q \land P(a) \land \forall x.(\neg P(x) \lor (\neg P(s^2x) \land (P(sx) \lor \neg Q)))$$

Then the matrix of the formula has columns for conjuncts and rows for disjunts. And we can still build connections between them like before building a substitution:



$$\left[[Q][P(a)] \left[\begin{matrix} \neg P(x') \\ \left[[\neg P(s^2x')] \begin{bmatrix} P(sx') \\ \neg Q \end{bmatrix} \right] \end{matrix} \right] \right]$$

$\sigma = \{\}$

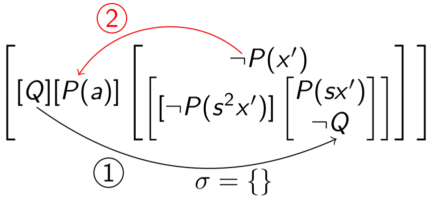
Note that there are proofs that are linearly shorter in the non-clausal variant (but implementation is more complex and therefore so far comparably performant)

LeanCoP Variant: Non Clausal

One exciting feature of the Connection Calculus, is that it does not require complete CNF. It is enough to have a formula in prenex/negation normal form and skolemized, but not in CNF. Consider:

$$Q \wedge P(a) \wedge \forall x.(\neg P(x) \vee (\neg P(s^2x) \wedge (P(sx) \vee \neg Q)))$$

Then the matrix of the formula has columns for conjuncts and rows for disjunts. And we can still build connections between them like before building a substitution:



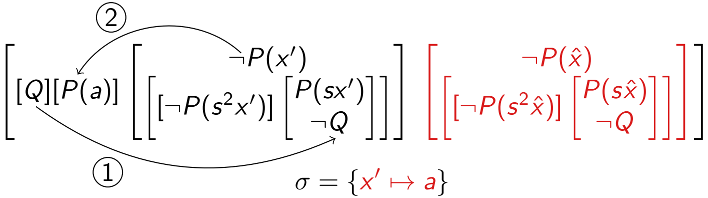
Note that there are proofs that are linearly shorter in the non-clausal variant (but implementation is more complex and therefore so far comparably performant)

LeanCoP Variant: Non Clausal

One exciting feature of the Connection Calculus, is that it does not require complete CNF. It is enough to have a formula in prenex/negation normal form and skolemized, but not in CNF. Consider:

$$Q \wedge P(a) \wedge \forall x.(\neg P(x) \vee (\neg P(s^2x) \wedge (P(sx) \vee \neg Q)))$$

Then the matrix of the formula has columns for conjuncts and rows for disjunts. And we can still build connections between them like before building a substitution:



$$\sigma = \{x' \mapsto a\}$$

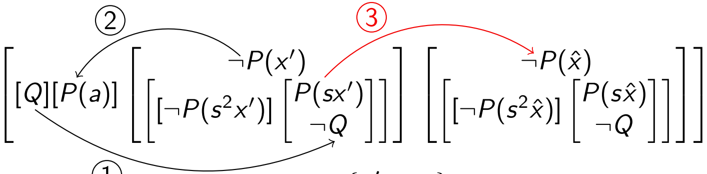
Note that there are proofs that are linearly shorter in the non-clausal variant (but implementation is more complex and therefore so far comparably performant)

LeanCoP Variant: Non Clausal

One exciting feature of the Connection Calculus, is that it does not require complete CNF. It is enough to have a formula in prenex/negation normal form and skolemized, but not in CNF. Consider:

$$Q \wedge P(a) \wedge \forall x.(\neg P(x) \vee (\neg P(s^2x) \wedge (P(sx) \vee \neg Q)))$$

Then the matrix of the formula has columns for conjuncts and rows for disjunts. And we can still build connections between them like before building a substitution:



$$\sigma = \{x' \mapsto a\}$$

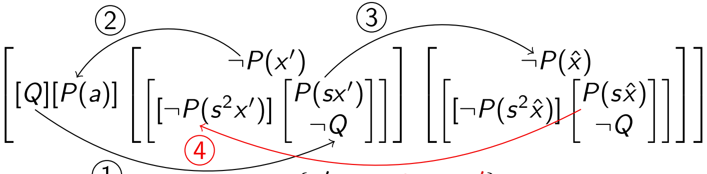
Note that there are proofs that are linearly shorter in the non-clausal variant (but implementation is more complex and therefore so far comparably performant)

LeanCoP Variant: Non Clausal

One exciting feature of the Connection Calculus, is that it does not require complete CNF. It is enough to have a formula in prenex/negation normal form and skolemized, but not in CNF. Consider:

$$Q \wedge P(a) \wedge \forall x.(\neg P(x) \vee (\neg P(s^2 x) \wedge (P(sx) \vee \neg Q)))$$

Then the matrix of the formula has columns for conjuncts and rows for disjunts. And we can still build connections between them like before building a substitution:



$$\sigma = \{x' \mapsto a, \hat{x} \mapsto sx'\}$$

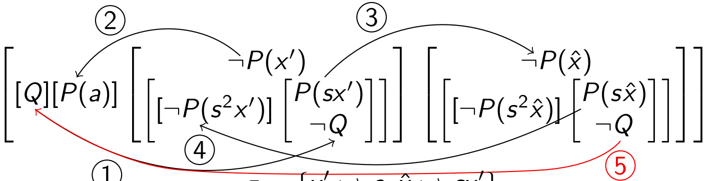
Note that there are proofs that are linearly shorter in the non-clausal variant (but implementation is more complex and therefore so far comparably performant)

LeanCoP Variant: Non Clausal

One exciting feature of the Connection Calculus, is that it does not require complete CNF. It is enough to have a formula in prenex/negation normal form and skolemized, but not in CNF. Consider:

$$Q \wedge P(a) \wedge \forall x.(\neg P(x) \vee (\neg P(s^2x) \wedge (P(sx) \vee \neg Q)))$$

Then the matrix of the formula has columns for conjuncts and rows for disjunts. And we can still build connections between them like before building a substitution:



$\left[[Q][P(a)] \left[\begin{matrix} \neg P(x') \\ \left[[\neg P(s^2x')] \begin{bmatrix} P(sx') \\ \neg Q \end{bmatrix} \right] \end{matrix} \right] \left[\begin{matrix} \neg P(\hat{x}) \\ \left[[\neg P(s^2\hat{x})] \begin{bmatrix} P(s\hat{x}) \\ \neg Q \end{bmatrix} \right] \end{matrix} \right] \right]$
1
2
3
4
5

$$\sigma = \{x' \mapsto a, \hat{x} \mapsto sx'\}$$

Note that there are proofs that are linearly shorter in the non-clausal variant (but implementation is more complex and therefore so far comparably performant)

leanCoP: More variants [*Otten*]

- leanCoP-Ω: leanCoP extended to linear integer arithmetic
- leanCoP-SiNE, leanCoP-ARDE: integrate preprocessing to deal with very large problems/formulae (several millions of axioms)
- ileanCoP: leanCoP extended to first-order intuitionistic logic (using prefixes and an additional prefix unification)
- MleanCoP: leanCoP extended to first-order modal logic (using prefixes and an additional prefix unification)
- First-order logic = propositional logic + term unification
  Non-classical logics = classical logic + prefix unification
- ileanCoP and MleanCoP are currently strongest provers for first-order intuitionistic and some first-order modal logics

mleanCoP and ileanCoP [*Otten*]

- constructive/intuitionistic logic used when formalizing computation (proof-as-programs, e.g. in NuPRL, Coq)
- modal logics used within formal verification (of, e.g., circuits, protocols, and programs)

Idea: Extend the classical prover leanCoP based on a clausal connection calculus to intuitionistic and modal logics.

Connection-based proof search:

- clausal form translation adds prefix(-string) to each literal
- Skolemization extended to prefix variables and constants
- first, a classical proof search is performed
- during the proof search all prefixes of connections are collected
- after classical proof is found, prefixes (of connections) are unified

Tableaux summary

Until now Tableaux based proof search is the state of art in intuitionistic logic and modal logic, and in particular leancop based systems are major ones

But for the most common classical first-order logic this is not the case. Resolution based systems are comparably powerful.

This distinction becomes even more evident when considering equality. Combining term-rewriting related techniques that handle equality well with resolution has been very successful, and we'll see it in the next part of the course. Handling equality efficiently in Tableaux is still an open problem.

Additional Literature (not required)

More on indexing structures; details of a C implementation of leanCoP; details of restricted backtracking.

Peter Graf.
Term Indexing, volume 1053 of *Lecture Notes in Computer Science*.
Springer, 1996.

Cezary Kaliszyk.
Efficient low-level connection tableaux.
In Hans de Nivelle, editor, *Automated Reasoning with Analytic Tableaux and Related Methods - 24th International Conference, TABLEAUX 2015*, volume 9323 of *Lecture Notes in Computer Science*, pages 102–111. Springer, 2015.

Jens Otten.
Restricting backtracking in connection calculi.
AI Commun., 23(2-3):159–182, 2010.

Summary

This Lecture

- Full details of Tableaux in functional style
- leanCoP
- Optimizations possible in lean tableaux
- Variants for other logics

Next

- Resolution
- Orderings
- Equality

Exercises

To be submitted before June 2

- Figure out how to yse the discrimination tree from slide 17 to find (a) the same, (b) matching, and (c) unifying terms with $f(X, Y)$ where X, Y are variables.
- Install and run leanCoP (any version from last 5 years) on a simple provable TPTP problem and study the short proof and the full proof. How does the DNF proof translate to a CNF proof, in particular what is the reason that reduction in CNF is a valid inference?
- Choose your favorite programming language. Write a TPTP parser in that language. CNF is enough, but if you do FOF you may have less to do in future weeks of the course. Note that this task will be extended by further functionality in next weeks (adding unification, and some reasoning), so do choose a language where these will be doable.

Send the answers to `cezary.kaliszyk@uibk.ac.at`