Computational Logic

Automated Theorem Proving

Lecture 4

Cezary Kaliszyk

June 9, 2020

Outline

Previous Lecture

- Resolution
- Efficient Unification
- Orderings

Today

- Resolution in practice
- Inefficiencies
- Use of orderings

Administration News: Olat

A student has asked to set up an Olat forum for the course, so you can easier ask questions not only to me but also to discuss among each other.

It is now active. Note however that I am not notified about messages there, so if you want to contact me, email is preferred.

Last Lecture Add-Ons

For the student that have not attended the University of Innsbruck obligatory Bachelor level courses, unification was taught twice:

- In Functional Programming:
  `http://cl-informatik.uibk.ac.at/teaching/ws18/fp/slides/10x1.pdf`
- In Logic in Computer Science ("LICS"):
  `http://cl-informatik.uibk.ac.at/teaching/ss20/lics/slides/12x1.pdf`

(also in Prolog course and others)

Summary of last lecture

Last time we've seen resolution

Given a problem to prove, we negate the conjecture and transform it to a set of clauses

(*) Given the set of clauses, we apply these two inferences to the set

$$\frac{C \vee A \quad D \vee \neg B}{(C \vee D)\sigma} \qquad \frac{C \vee A \vee B}{(C \vee A)\sigma}$$

all resulting clauses are added to the set

if the empty clause is not derived yet, we repeat from (*)

The procedure is sound and complete: If the problem is provable, eventually the empty clause will be derived

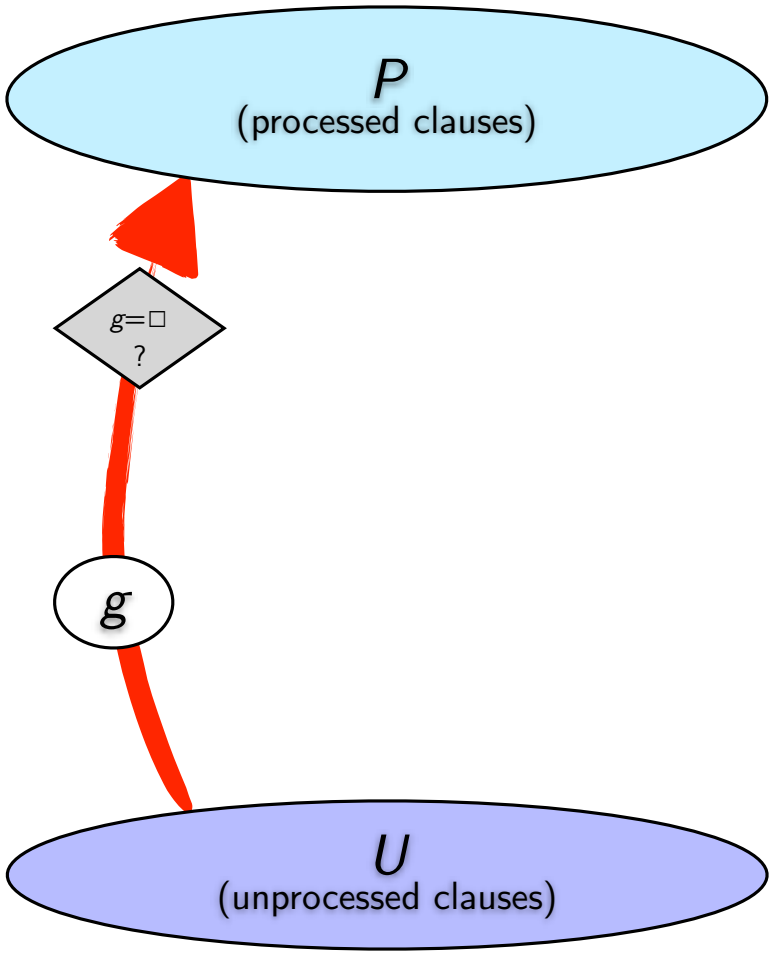
How to use the inferences more practically

The algorithm from the previous slide is already sound and complete but somewhat impractical

There is no need to test every clause with every clause in every step: some of these have been tried before. Also the cardinality of the clauses can potentially square in every repeat of the loop

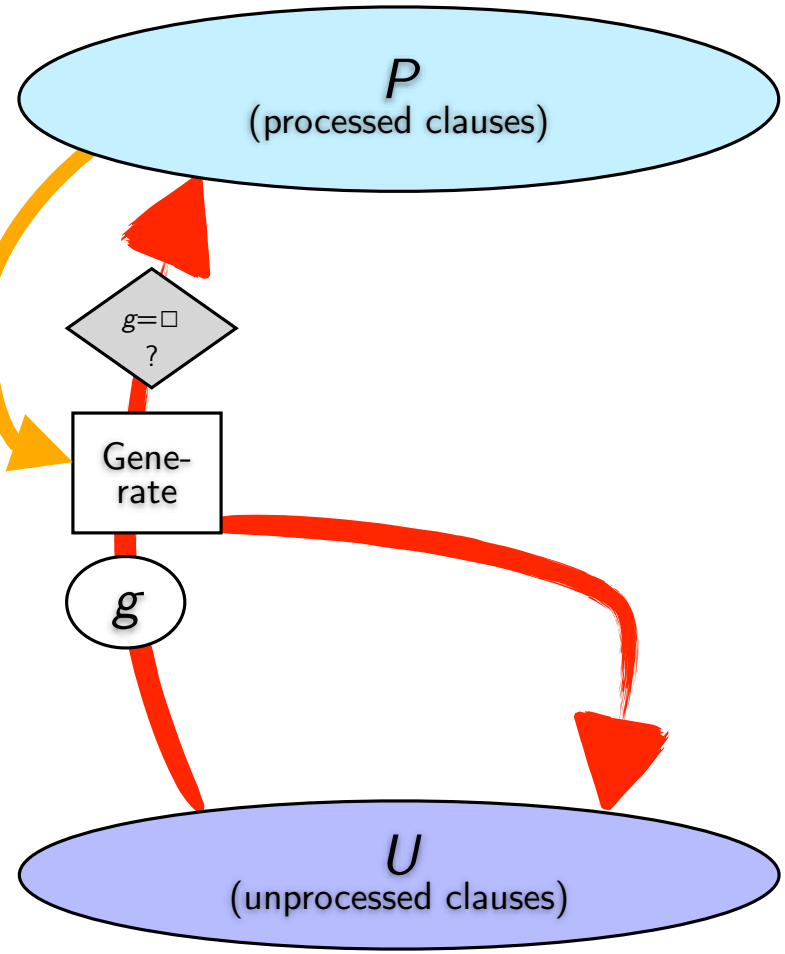
For this most provers use the so-called given-clause algorithm

Given-clause algorithm



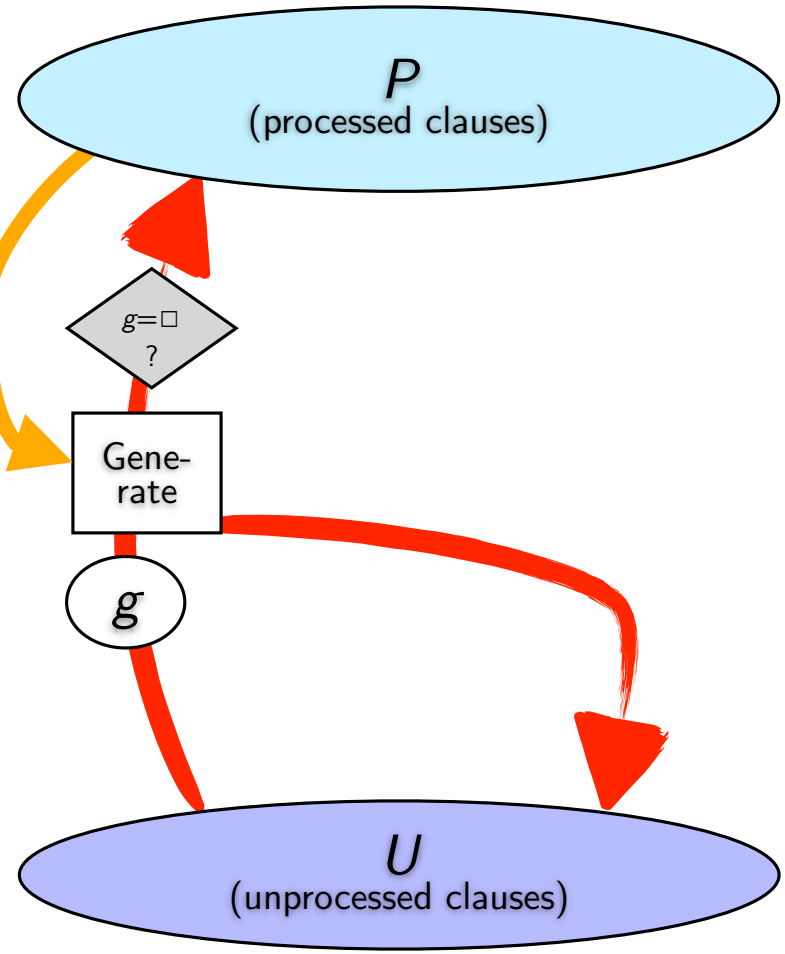
- Initially all clauses are not processed
- Goal: Move all clauses from U to P
- Select one clause u from U

Given-clause algorithm



- Initially all clauses are not processed
- Goal: Move all clauses from U to P
- Select one clause u from U
- Generate all clauses by inferences between u and P and add them to U and insert u into P

Given-clause algorithm



- Initially all clauses are not processed
- Goal: Move all clauses from U to P
- Select one clause u from U
- Generate all clauses by inferences between u and P and add them to U and insert u into P
- Invariant: all clauses in P are interreduced

Given-clause properties

In some provers the sets are referred to as the set of "active clauses" and of "passive clauses"

Note that the set of unprocessed clauses U still grows fast. But the set P grows by at most one clause per iteration of the algorithm.

Now the advantage is that instead of processing everything with everything in the large set U, we process the single selected unprocessed clause u with the clauses in the more manageable set P.

Is the algorithm still complete? If we choose a fair strategy of selecting the clause u (i.e. eventually every clause will be selected) then the procedure is still complete

Now what comes as most important is the selection of the clause u. For example, the empty clause is empty, so focusing on the shortest clause could be a good strategy. Question: is it a fair one? (See exercises)

Basic Approaches to Clause Selection

Symbol counting

- Pick smallest clause in U
- (Size of a clause is the number of symbols / variables that appear in all the literals)

FIFO

- Pick the oldest clause (the one added earliest)

Flexible weighting

- Symbol counting giving different weights to different symbols
- For exampe lower weight to symbols from the goal

Combined

- Interleave the selection of clauses from the above schemes
- Most popular: mostly pick small clauses, but every 5 times select an oldest clause to keep the selection fair [*DISCOUNT*]

Observations

So far, we have only reordered the inferences that would be made either way

This is good and can lead to finding proofs in more reasonable time, but does not change any inferences: all would still be made

But maybe not all are needed?

Idea 1

Consider the problem:

$$a \vee b \vee c \qquad \neg a \qquad \neg b \qquad \neg c$$

from the first clause we can remove either the a, the b, or the c. This way we produce 3 new clauses.

$$a \vee b \qquad a \vee c \qquad b \vee c$$

From each of them we can proceed in two different ways, getting 3 distinct clauses

Were all these opeations necessary?

Well, we could decide that we get always resolve a before b before c. This way there is only one way we can proceed.

Idea 2

Consider a problem, where all the clauses have both postive (non-negated) and negative (negated literals. Can we ever get a proof (empty clause)?

No. Resolving a clause that contains positive and negative literals with a clause that contains positive and negative literals only ever leads to clauses with positive and negative literals. Factoring a mixed clause also gives a mixed clause.

So every proof must always contain some inferences with only positive or only negative clauses.

Implications of the Ideas

We consider the problem:

$$a \vee b \vee c \qquad \neg a \qquad \neg b \qquad \neg c$$

And only allow to resolve a clause with another one on a literal if that literal is maximal in the order

$$\neg a > \neg b > \neg c > a > b > c$$

Is this still complete?

More general: Is imposing an ordering on literals in resolution still complete?

Proof of completeness (informal). The original calculus was complete. We will show that one where an ordering is imposed is still complete. For this it suffices to show that if there exists a proof which does not satisfy the ordering, there still exists a proof that satisfies the ordering. Indeed in the proof where the ordering was not satisfied, there was a point where a resolution inference was done that does not satisfy the ordering. So a literal resolved was not maximal. This means that the maximal literal from that clause was resolved later on in the proof. We reorder the proof moving the later inference to this point. The fact that we can do this and by what steps is left to the reader.

Ordered Resolution

Resolution

$$\frac{C \vee A \quad D \vee \neg B}{(C \vee D)\sigma} \qquad \frac{C \vee A \vee B}{(C \vee A)\sigma}$$

$A\sigma$ strictly maximal wrt $C\sigma$ and B maximal wrt $D\sigma$.

The calculus is sound and complete for orderings that are well behaved.

Needed properties for the orderings

- Closed under context
- Closed under substitution
- Ground complete

Ordered Resolution

Ordered Resolution

$$\frac{C \vee A \quad D \vee \neg B}{(C \vee D)\sigma} \qquad \frac{C \vee A \vee B}{(C \vee A)\sigma}$$

$A\sigma$ strictly maximal wrt $C\sigma$ and B maximal wrt $D\sigma$.

The calculus is sound and complete for orderings that are well behaved.

Needed properties for the orderings

- Closed under context
- Closed under substitution
- Ground complete

Useful Orders

Reminder from the Term Rewriting course:

Definition (LPO [Middeldorp])

Given a precedence on symbols $>_\Sigma$, we define the lexicographic path order (LPO) $>_{\text{lpo}}$ as follows: $s = f(s_1, \ldots, s_n) >_{\text{lpo}} t$ iff one of the following conditions holds:

1. $t = f(t_1, \ldots, t_n)$ and $\exists i \in \{1, \ldots, n\}$ such that
   - (i) $s_j = t_j$ for all j such that $1 \leqslant j < i$,
   - (ii) $s_i >_{\text{lpo}} t_i$, and
   - (iii) $s >_{\text{lpo}} t_j$ for all j such that $i < j \leqslant n$,
2. $t = g(t_1, \ldots, t_m)$, $f >_\Sigma g$, and $s >_{\text{lpo}} t_i$ for all i such that $1 \leqslant i \leqslant m$, or
3. $s_i \geq_{\text{lpo}} t$ for all i such that $1 \leqslant i \leqslant n$.

Where $\geq_{\text{lpo}}$ is the reflexive closure of $>_{\text{lpo}}$.

Other useful orders in first-order theorem proving are KBO and RPO and are implemented in many modern provers. They will be also useful in more efficient handling of equality.

Summary

This Lecture

- Resolution in practice: Given clause algorithm
- Using an ordering to restrict the inferences in resolution

Next

- Equality in theorem proving
- Paramodulation and superposition

Exercises: To be submitted before June 16

- Consider the problem

$$p(o) \wedge (\forall X.(p(X) \rightarrow p(s(X)))) \rightarrow r$$

  Convert the problem to CNF manually, and try a few steps of the given-clause algorithm
- Can you propose a problem where a proof exists, but the strategy to select the shortest clause does not lead to a solution?
  Hint: add some clause to the first exercise.
- Extend the unification exercise from last week to now perform the resolution of two clauses.

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