Computational Logic

Automated Theorem Proving

Lecture 5

Cezary Kaliszyk

June 16, 2020

Outline

Previous Lecture

- Resolution in practice
- Issues in resolution
- Orderings

Today

- Equality
- Paramodulation
- Superposition

Administration

The last usual practical assignment is given today and due next week

Then we will have the more exam-like assignment

Note however, that it is also given online and there is more time for it

Some hints about the tasks in the exam given at the end today

Last Lecture Add-Ons

The given-clause algorithm was slightly confusing.

Resolution was given a set of original clauses and was repeatedly doing all inferences possible with this set

The idea of the given clause algorithm is to give concrete data structures and order of operations

A really simplified example follows

Given Clause Algorithm Example

In every time step the first in the quere unprocessed clause u_n is renamed as p_n when processed and a number of new clauses are generated added at the end of unprocessed. For example:

Time	Unprocessed	Processed
0	u1,u2,u3	
1	u2, u3, u4, u5	p1
2	u3, u4, u5, u6, u7	p1, p2
3	u4, u5, u6, u7, u8	p1, p2, p3

Orderings in practice

Last time we've seen some practical issues in resolution

In particular, adding an ordering can forbid a number of inferences

While keeping the calculus sound and complete

This means that if you guess an ordering well, we can make the proofs found by the given-clause algorithm much more efficiently

Note however, that that if you choose an ordering that is not very good for a problem (this also means selecting parameters, such as the precedence not very well), still all the boring things will be derived and additionally checking the ordering will slow down the prover

In practice, provers try a problem with multiple orderings subsequently

Importance of equality

Equality in Math

In order to define addition, we need:

$$\begin{aligned} 0 + x &= x \\ s(x) + y &= s(x + y) \end{aligned}$$

Without equality, a lot of basic math is not doable

Equality in Programs

Functional programs are also expressed as equalities, and in fact ATP use in program verification is usually equality between the result of a program and its specification

Proofs without Equality are "easy"

Second order monadic logic is decidable. Read: if we do not have binary predicates (like equality) even second order is fine.

Traditional way of dealing with Equality

Add a new predicate symbol $eq(X, Y)$ and axiomatize it.

Equality axioms: reflexivity, symmetry, transitivity, congruence

$$\begin{aligned} x &= x \\ x = y &\Leftrightarrow y = x \\ x = y \wedge y = z &\Rightarrow x = z \end{aligned}$$

And for every function symbol f and predicate symbol p:

$$\begin{aligned} x = y &\Leftrightarrow f(x) = f(y) \\ x = y \wedge p(x) &\Rightarrow p(y) \end{aligned}$$

Completeness

Adding these equality axioms to an uninterpreted symbol allows a prover to properly handle equality: all basic things can be derived

Done for example by leanCoP

Issues with traditional equality handling

Consider a bigger term $f(g(a), c)$ and an equality $a = b$.

In order to use the equality axioms to replace the a by a b we we have to step-by-step put the equality in a context and use transitivity many times

It would be much more efficient and would not create many irrelevant more complex congruences if we were to add rewriting to the calculus.

This is done in the paramodulation calculus. Note that the optimization that we did with ordered resolution (selecting equalities to process in the ordering) can be immediately done here as well

Paramodulation Calculus

Resolution

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

Paramodulation Calculus

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 $D\sigma$.

Paramodulation Calculus

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 $D\sigma$.

Instead of equality axioms:

Paramodulation

$$\frac{C \vee s \neq s'}{C\sigma'} \qquad \frac{C \vee s = t \quad D \vee L[s']}{(C \vee D \vee L[t])\sigma'}$$

Paramodulation Calculus

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 $D\sigma$.

Instead of equality axioms:

Ordered Paramodulation

$$\frac{C \vee s \neq s'}{C\sigma'} \qquad \frac{C \vee s = t \quad D \vee L[s']}{(C \vee D \vee L[t])\sigma'}$$

$(s = t)\sigma$ and $L[s']\sigma'$ maximal in their clauses.

More observations about equality (1/2)

(Ordered) paramodulation is again an improvement over traditional handling of equality.

However not all issues have been solved. Consider an equality

$$a = b$$

Then in every single clause in the problem, all occurrences of a can be replaced by b and the other way around. So with any clause with 10 occurrences of a and b we still get 2^n variants of the clause. It would be much better to decide that we replace all a by a b everywhere in the proof and never get back to the a.

More observations about equality (2/2)

Even worse, consider an equation

$$x + 0 = x$$

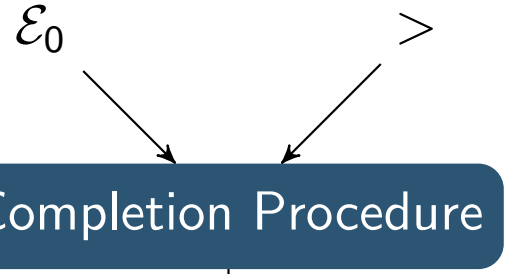
This lets us drop "+0" anywhere. But it can also be used the other way around and change any expression to an expression with arbitrarly many "+0".

It would be nice to realize that there is no need to add such +0 terms everywhere. But maybe by not using the rule we can lose something that we could prove by adding a +0 somewhere?

The completion procedure, known from term rewriting, tries to do just this: Transform a set of equalities into a set of rewrite rules, that is equalities that will be used only one direction, but where the consequences are the same.

Completion

Input: Initial set of equations and a reduction order. In every step the set of equations and an initially empty rewrite system are transformed by the following rules:



$$(\mathcal{E}_0, \varnothing) \vdash (\mathcal{E}_1, \mathcal{R}_1) \vdash (\mathcal{E}_2, \mathcal{R}_2) \vdash (\mathcal{E}_3, \mathcal{R}_3) \vdash \cdots$$

deduce $\dfrac{(\mathcal{E}, \mathcal{R})}{(\mathcal{E} \cup \{s \approx t\}, \mathcal{R})}$ if $s \;{}_{\mathcal{R}}\!\leftarrow u \rightarrow_{\mathcal{R}} t$

delete $\dfrac{(\mathcal{E} \cup \{s \approx s\}, R)}{(\mathcal{E}, \mathcal{R})}$

orient $\dfrac{(\mathcal{E} \cup \{s \mathrel{\dot{\approx}} t\}, \mathcal{R})}{(\mathcal{E}, \mathcal{R} \cup \{s \rightarrow t\})}$ if $s > t$

compose $\dfrac{(\mathcal{E}, \mathcal{R} \cup \{s \rightarrow t\})}{(\mathcal{E}, \mathcal{R} \cup \{s \rightarrow u\})}$ if $t \rightarrow_{\mathcal{R}} u$

simplify $\dfrac{(\mathcal{E} \cup \{s \mathrel{\dot{\approx}} t\}, \mathcal{R})}{(\mathcal{E} \cup \{u \mathrel{\dot{\approx}} t\}, \mathcal{R})}$ if $s \rightarrow_{\mathcal{R}} u$

collapse $\dfrac{(\mathcal{E}, \mathcal{R} \cup \{s \rightarrow t\})}{(\mathcal{E} \cup \{u \approx t\}, \mathcal{R})}$ if $s \overset{\exists}{\rightarrow}_{\mathcal{R}} u$

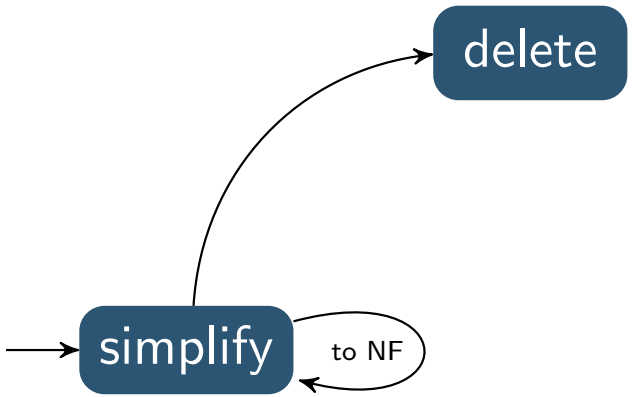
If the equation system is empty in some step, we are done. Otherwise the intermediate system + equations are still useful (unfailing completion)

Completion in Practice

- Given a tool that can find a termination relation ($T_T T_2$ here)
- The following procedure either finds a confluent and terminating system equivalent to the equation problem (or loops)

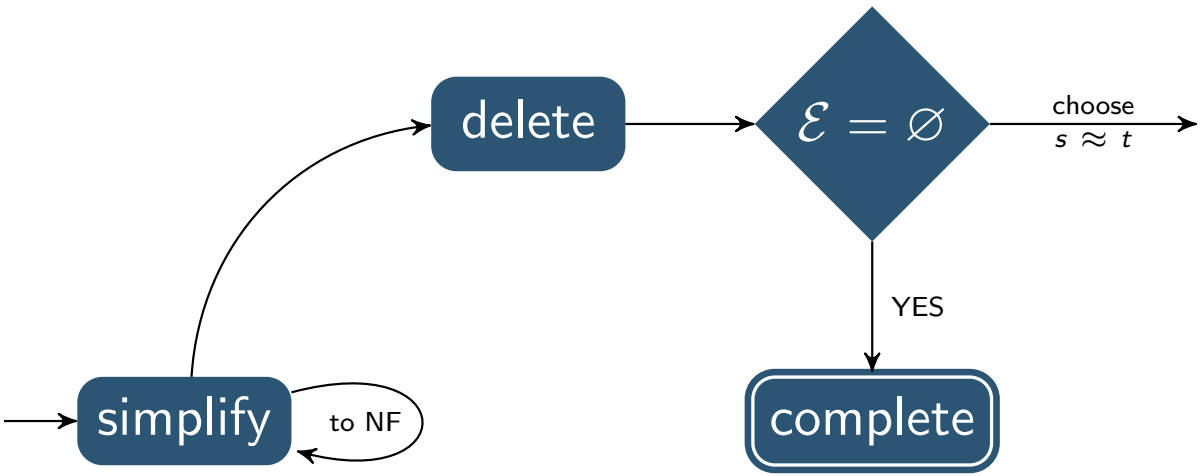
Completion in Practice

- Given a tool that can find a termination relation ($T_T T_2$ here)
- The following procedure either finds a confluent and terminating system equivalent to the equation problem (or loops)



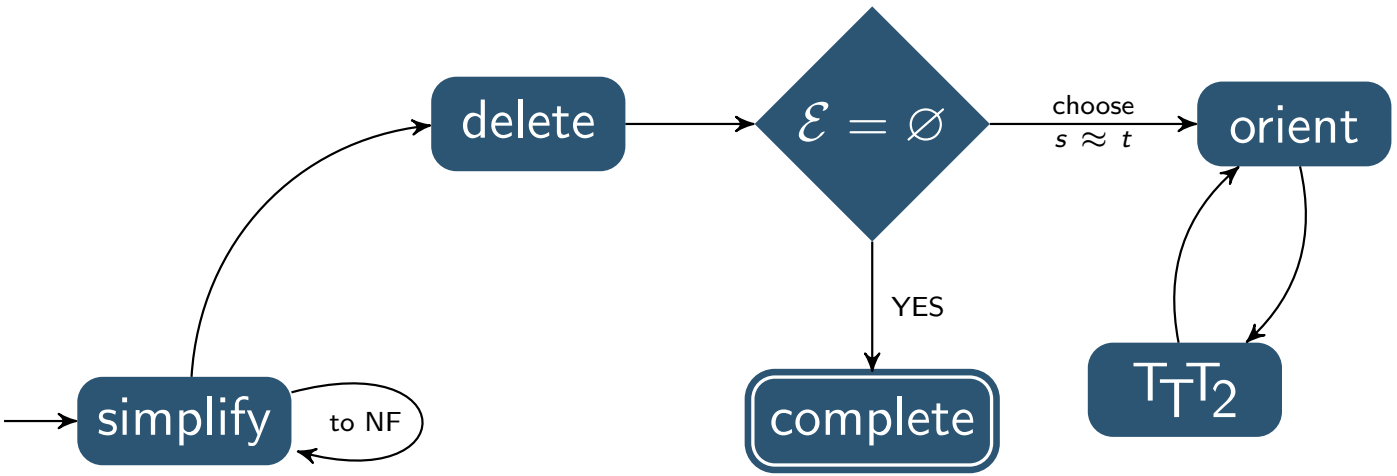
Completion in Practice

- Given a tool that can find a termination relation ($T_T T_2$ here)
- The following procedure either finds a confluent and terminating system equivalent to the equation problem (or loops)



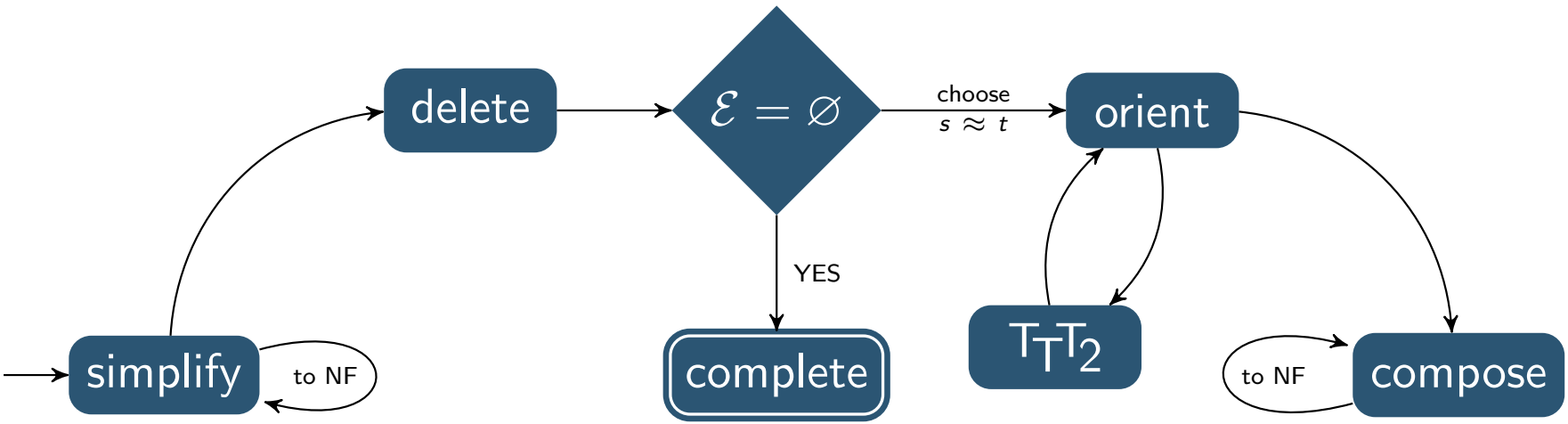
Completion in Practice

- Given a tool that can find a termination relation ($\mathsf{T}_\mathsf{T}\mathsf{T}_2$ here)
- The following procedure either finds a confluent and terminating system equivalent to the equation problem (or loops)



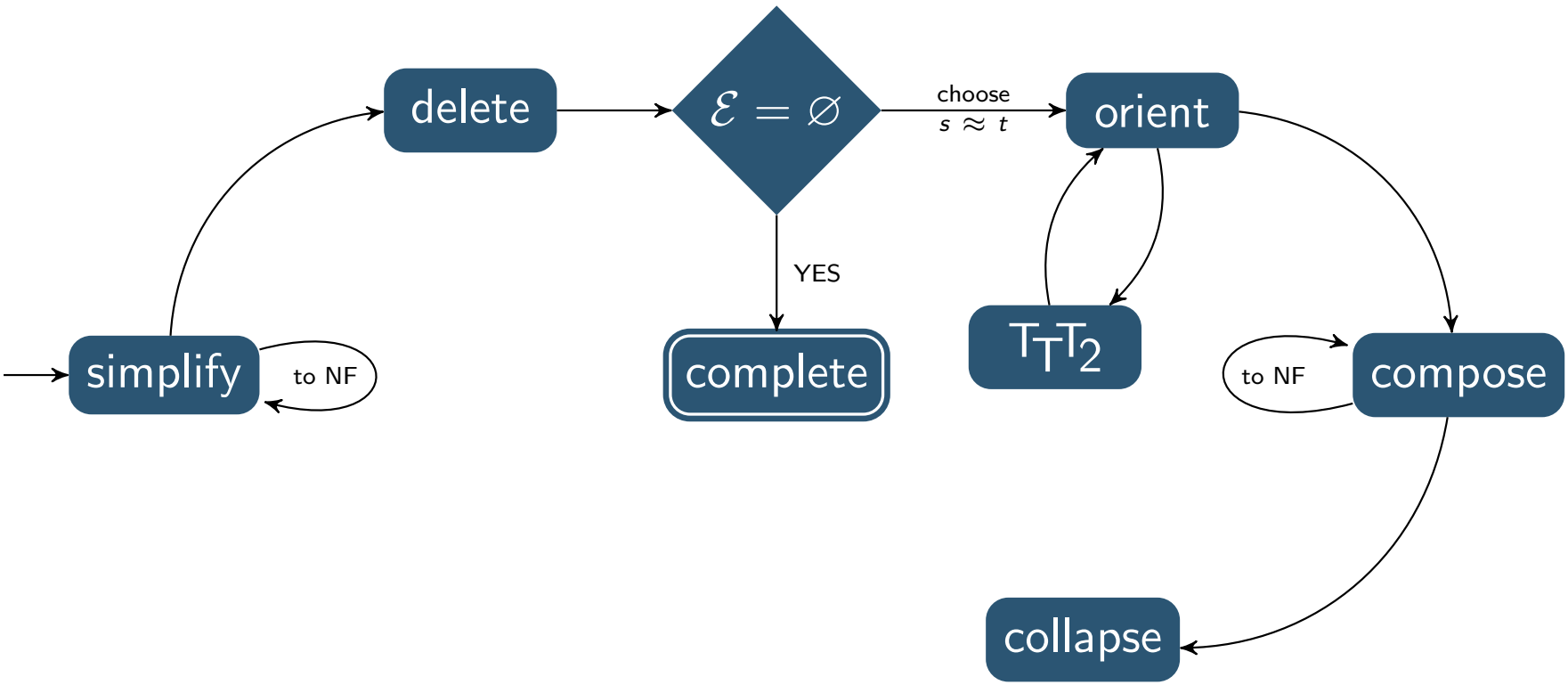
Completion in Practice

- Given a tool that can find a termination relation ($\mathsf{T}_\mathsf{T}\mathsf{T}_2$ here)
- The following procedure either finds a confluent and terminating system equivalent to the equation problem (or loops)



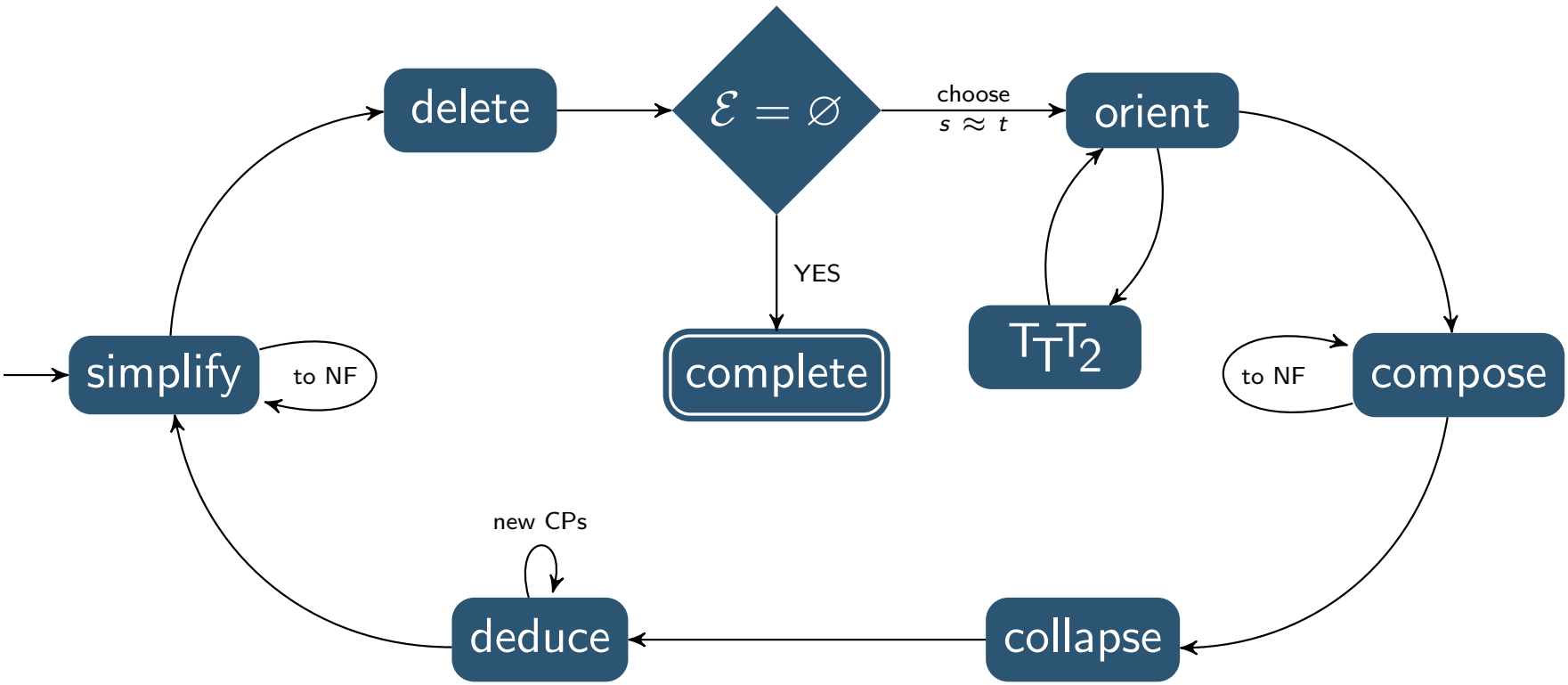
Completion in Practice

- Given a tool that can find a termination relation ($\mathsf{T}_\mathsf{T}\mathsf{T}_2$ here)
- The following procedure either finds a confluent and terminating system equivalent to the equation problem (or loops)



Completion in Practice

- Given a tool that can find a termination relation ($\mathsf{T}_\mathsf{T}\mathsf{T}_2$ here)
- The following procedure either finds a confluent and terminating system equivalent to the equation problem (or loops)



Completion in theorem proving

Resolution loop is also a semi-decidable procedure

- We can interleave a step of the completion procedure for the equations given with resolution steps
- Some equations are parts of clauses: This is not a problem, the other parts of the clause can be kept
- Even further: we can include completion rules in the calculus. This means a system of rules that rewrite clauses and equations with other equations but in such a way to preserve the given ordering
- Note: Usually a single fixed ordering is used

This gives rise to the superposition calculus

Superposition calculus

This is essentially the ordered paramodulation calculus, where additionally we can rewrite equations and inequalities with other equations, but only in certain ways.

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

$$\frac{C \vee s = t \quad D \vee \neg A[s']}{(C \vee D \vee \neg A[t])\sigma} \qquad \frac{C \vee s = t \quad D \vee A[s']}{(C \vee D \vee A[t])\sigma}$$

$$\frac{C \vee s = t \quad D \vee u[s'] \neq v}{(C \vee D \vee u[t] \neq v)\sigma} \qquad \frac{C \vee s = t \quad D \vee u[s'] = v}{(C \vee D \vee u[t] = v)\sigma}$$

$$\frac{C \vee s \neq t}{C\sigma} \qquad \frac{C \vee u = v \vee s = t}{(C \vee v \neq t \vee u = t)\sigma}$$

with usual order-based restrictions on clauses in the inferences but also $t\sigma$ not bigger than $s\sigma$.

We will not discuss it further in the course, but it is sound and complete.

Additional Literature (not required)

A rather hard paper to read, but properly introduces the intricacies of the superposition calculus

Leo Bachmair, Harald Ganzinger, Christopher Lynch, and Wayne Snyder.
Basic paramodulation and superposition.
In Deepak Kapur, editor, *Automated Deduction - CADE-11, 11th International Conference on Automated Deduction, Saratoga Springs, NY, USA, June 15-18, 1992, Proceedings*, volume 607 of *Lecture Notes in Computer Science*, pages 462–476. Springer, 1992.

Summary

This Lecture

- Equality
- Paramodulation
- Superposition

Next

Some current research topics in automated theorem proving

- Automated theorem proving beyond first-order logic
- AI in theorem proving

The virtual exam

What to expect at the "exam"

It was supposed to happen in the last slot of the course, but as I will either way give one week for it, no short two hour slot is given. Example topics you could expect:

A practical assignment to do a proof either in ordered resolution or a paramodulation proof (no superposition)

A question tableaux

A question about the selection of the next clause in the given-clause algorithm and fairness

And something about the understanding of the research topics next week

Exercises: To be submitted before June 23

(1) Equalty

Consider the problem

$$a = b \wedge p(s(a)) \wedge \neg p(s(b))$$

Derive the equality axioms for the problem and find a proof with the axioms

(2) Implementation

Extend the implementation parts that you have to a resolution prover: Given a problem in TPTP, it should read the CNF and follow the given clause algorithm performing resolutions until it finds an empty clause.

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