

# **Automated Reasoning II**

**Uwe Waldmann**

**Summer Term 2026**

# Topics of the Course

---

Term rewriting systems:

proving termination using dependency pairs.

Decision procedures:

equality (congruence closure),  
algebraic theories,  
combinations.

Satisfiability modulo theories (SMT):

CDCL(T),  
dealing with universal quantification.

# Topics of the Course

---

Superposition:

combining ordered resolution and completion,  
optimizations,  
integrating theories.

# Part 1: Termination Revisited

---

So far: Termination as a subordinate task for entailment checking.

TRS is generated by some saturation process; ordering must be chosen before the saturation starts.

Now: Termination as a main task (e.g., for program analysis).

TRS is fixed and known in advance.

# Termination Revisited

---

Literature:

Nao Hirokawa and Aart Middeldorp: Dependency Pairs Revisited, RTA 2004, pp. 249-268 (in particular Sect. 1–4).

Thomas Arts and Jürgen Giesl: Termination of Term Rewriting Using Dependency Pairs, Theoretical Computer Science, 236:133-178, 2000.

# 1.1 Dependency Pairs

---

Invented by T. Arts and J. Giesl in 1996,  
many refinements since then.

Given: finite TRS  $R$  over  $\Sigma = (\Omega, \emptyset)$ .

$T_0 := \{ t \in T_\Sigma(X) \mid \exists \text{ infinite deriv. } t \rightarrow_R t_1 \rightarrow_R t_2 \rightarrow_R \dots \}$ .

$T_\infty := \{ t \in T_0 \mid \forall p > \varepsilon : t|_p \notin T_0 \}$   
= minimal elements of  $T_0$  w.r.t.  $\triangleright$ .

$t \in T_0 \Rightarrow$  there exists a  $t' \in T_\infty$  such that  $t \triangleright t'$ .

$R$  is non-terminating if and only if  $T_0 \neq \emptyset$  if and only if  $T_\infty \neq \emptyset$ .

# Dependency Pairs

---

Assume that  $T_\infty \neq \emptyset$  and consider some non-terminating derivation starting from  $t \in T_\infty$ .

Since all subterms of  $t$  allow only finite derivations, at some point a rule  $l \rightarrow r \in R$  must be applied at the root of  $t$  (possibly preceded by rewrite steps below the root):

$$t = f(t_1, \dots, t_n) \xrightarrow{>\varepsilon}_R^* f(s_1, \dots, s_n) = l\sigma \xrightarrow{\varepsilon}_R r\sigma.$$

In particular,  $\text{root}(t) = \text{root}(l)$ , so we see that the root symbol of any term in  $T_\infty$  must be contained in

$$D := \{ \text{root}(l) \mid l \rightarrow r \in R \}.$$

$D$  is called the set of **defined symbols** of  $R$ ;

$C := \Omega \setminus D$  is called the set of **constructor symbols** of  $R$ .

## Dependency Pairs

---

The term  $r\sigma$  is contained in  $T_0$ , so there exists a  $v \in T_\infty$  such that  $r\sigma \triangleright v$ .

If  $v$  occurred in  $r\sigma$  at or below a variable position of  $r$ , then  $x\sigma|_p = v$  for some  $x \in \text{var}(r) \subseteq \text{var}(l)$ , hence  $s_i \triangleright x\sigma$  and there would be an infinite derivation starting from some  $t_i$ .

This contradicts  $t \in T_\infty$ , though.

Therefore,  $v = u\sigma$  for some non-variable subterm  $u$  of  $r$ .

As  $v \in T_\infty$ , we see that  $\text{root}(u) = \text{root}(v) \in D$ .

Moreover,  $u$  cannot be a proper subterm of  $l$ , since otherwise again there would be an infinite derivation starting from some  $t_i$ .

# Dependency Pairs

---

Putting everything together, we obtain

$$t = f(t_1, \dots, t_n) \xrightarrow{>\varepsilon}_R^* f(s_1, \dots, s_n) = l\sigma \xrightarrow{\varepsilon}_R r\sigma \triangleright u\sigma$$

where  $r \triangleright u$ ,  $u$  is not a variable,  $\text{root}(u) \in D$ ,  $l \not\triangleright u$ .

Since  $u\sigma \in T_\infty$ , we can continue this process and obtain an infinite sequence.

# Dependency Pairs

---

If we define

$$S := \{ l \rightarrow u \mid l \rightarrow r \in R, r \triangleright u, u \notin X, \text{root}(u) \in D, l \not\triangleright u \},$$

we can combine the rewrite step at the root and the subterm step and obtain

$$t \xrightarrow{>\varepsilon}_R^* l\sigma \xrightarrow{\varepsilon}_S u\sigma.$$

# Dependency Pairs

---

To get rid of the superscripts  $\varepsilon$  and  $>\varepsilon$ , it turns out to be useful to introduce a new set of function symbols  $f^\#$  that are only used for the root symbols of this derivation:

$$\Omega^\# := \{ f^\# / n \mid f / n \in \Omega \}.$$

For a term  $t = f(t_1, \dots, t_n)$  we define  $t^\# := f^\#(t_1, \dots, t_n)$ ;  
for a set of terms  $T$  we define  $T^\# := \{ t^\# \mid t \in T \}$ .

The set of **dependency pairs** of a TRS  $R$  is then defined by

$$\text{DP}(R) := \{ l^\# \rightarrow u^\# \mid l \rightarrow r \in R, r \triangleright u, u \notin X, \text{root}(u) \in D, l \not\triangleright u \}.$$

# Dependency Pairs

---

For  $t \in T_\infty$ , the sequence using the  $S$ -rule corresponds now to

$$t^\# \rightarrow_R^* l^\# \sigma \rightarrow_{\text{DP}(R)} u^\# \sigma$$

where  $t^\# \in T_\infty^\#$  and  $u^\# \sigma \in T_\infty^\#$ .

(Note that rules in  $R$  do not contain symbols from  $\Omega^\#$ , whereas all roots of terms in  $\text{DP}(R)$  come from  $\Omega^\#$ , so rules from  $R$  can only be applied below the root and rules from  $\text{DP}(R)$  can only be applied at the root.)

## Dependency Pairs

---

Since  $u^\# \sigma$  is again in  $T_\infty^\#$ , we can continue the process in the same way. We obtain:  $R$  is non-terminating if and only if there is an infinite sequence

$$t_1 \rightarrow_R^* t_2 \rightarrow_{\text{DP}(R)} t_3 \rightarrow_R^* t_4 \rightarrow_{\text{DP}(R)} \dots$$

with  $t_i \in T_\infty^\#$  for all  $i$ .

Moreover, if there exists such an infinite sequence, then there exists an infinite sequence in which all DPs that are used are used infinitely often. (If some DP is used only finitely often, we can cut off the initial part of the sequence up to the last occurrence of that DP; the remainder is still an infinite sequence.)

# Dependency Graphs

---

Such infinite sequences correspond to “cycles” in the “dependency graph”:

Dependency graph  $DG(R)$  of a TRS  $R$ :

directed graph

nodes: dependency pairs  $s \rightarrow t \in DP(R)$

edges: from  $s \rightarrow t$  to  $u \rightarrow v$  if there are  $\sigma, \tau$   
such that  $t\sigma \rightarrow_R^* u\tau$ .

# Dependency Graphs

---

Intuitively, we draw an edge between two dependency pairs, if these two dependency pairs can be used after another in an infinite sequence (with some  $R$ -steps in between). While this relation is undecidable in general, there are reasonable overapproximations:

# Dependency Graphs

---

The functions  $\text{cap}$  and  $\text{ren}$  are defined by:

$$\begin{aligned}\text{cap}(x) &= x \\ \text{cap}(f(t_1, \dots, t_n)) &= \begin{cases} y & \text{if } f \in D \\ f(\text{cap}(t_1), \dots, \text{cap}(t_n)) & \text{if } f \in C \cup D^\# \end{cases}\end{aligned}$$

$$\text{ren}(x) = y, \quad y \text{ fresh}$$

$$\text{ren}(f(t_1, \dots, t_n)) = f(\text{ren}(t_1), \dots, \text{ren}(t_n))$$

The overapproximated dependency graph contains an edge from  $s \rightarrow t$  to  $u \rightarrow v$  if  $\text{ren}(\text{cap}(t))$  and  $u$  are unifiable.

# Dependency Graphs

---

A **cycle** in the dependency graph is a non-empty subset  $K \subseteq DP(R)$  such that there is a non-empty path in  $K$  from every DP in  $K$  to every DP in  $K$  (the two DPs may be identical).

Let  $K \subseteq DP(R)$ . An infinite rewrite sequence in  $R \cup K$  of the form

$$t_1 \rightarrow_R^* t_2 \rightarrow_K t_3 \rightarrow_R^* t_4 \rightarrow_K \dots$$

with  $t_i \in T_\infty^\#$  is called  $K$ -minimal, if all rules in  $K$  are used infinitely often.

$R$  is non-terminating if and only if there is a cycle  $K \subseteq DP(R)$  and a  $K$ -minimal infinite rewrite sequence.

## 1.2 Subterm Criterion

---

Our task is to show that there are no  $K$ -minimal infinite rewrite sequences.

Suppose that every dependency pair symbol  $f^\sharp$  in  $K$  has positive arity (i.e., no constants). A **simple projection**  $\pi$  is a mapping  $\pi : \Omega^\sharp \rightarrow \mathbb{N}$  such that  $\pi(f^\sharp) = i \in \{1, \dots, \text{arity}(f^\sharp)\}$ .

We define  $\pi(f^\sharp(t_1, \dots, t_n)) = t_{\pi(f^\sharp)}$ .

# Subterm Criterion

---

Theorem 1.1 (Hirokawa and Middeldorp):

Let  $K$  be a cycle in  $DG(R)$ . If there is a simple projection  $\pi$  for  $K$  such that  $\pi(l) \succeq \pi(r)$  for every  $l \rightarrow r \in K$  and  $\pi(l) \triangleright \pi(r)$  for some  $l \rightarrow r \in K$ , then there are no  $K$ -minimal sequences.

## Subterm Criterion

---

Problem: The number of cycles in  $DG(R)$  can be exponential.

Better method: Analyze strongly connected components (SCCs).

SCC of a graph: maximal subgraph in which there is a non-empty path from every node to every node. (The two nodes can be identical.)<sup>a</sup>

Important property: Every cycle is contained in some SCC.

---

<sup>a</sup>There are several definitions of SCCs that differ in the treatment of edges from a node to itself.

## Subterm Criterion

---

Idea: Search for a simple projection  $\pi$  such that  $\pi(l) \succeq \pi(r)$  for all DPs  $l \rightarrow r$  in the SCC. Delete all DPs in the SCC for which  $\pi(l) \triangleright \pi(r)$  (by the previous theorem, there cannot be any  $K$ -minimal infinite rewrite sequences using these DPs). Then re-compute SCCs for the remaining graph and re-start.

No SCCs left  $\Rightarrow$  no cycles left  $\Rightarrow R$  is terminating.

Example: See Ex. 13 from Hirokawa and Middeldorp.

## 1.3 Reduction Pairs and Argument Filterings

---

Goal: Show the non-existence of  $K$ -minimal infinite rewrite sequences

$$t_1 \rightarrow_R^* u_1 \rightarrow_K t_2 \rightarrow_R^* u_2 \rightarrow_K \dots$$

using well-founded orderings.

We observe that the requirements for the orderings used here are less restrictive than for reduction orderings:

$K$ -rules are only used at the top, so we need stability under substitutions, but compatibility with contexts is unnecessary.

While  $\rightarrow_K$ -steps should be decreasing, for  $\rightarrow_R$ -steps it would be sufficient to show that they are not increasing.

# Reduction Pairs and Argument Filterings

---

This motivates the following definitions:

Rewrite quasi-ordering  $\succsim$ :

reflexive and transitive binary relation, stable under substitutions, compatible with contexts.

Reduction pair  $(\succsim, \succ)$ :

$\succsim$  is a rewrite quasi-ordering.

$\succ$  is a well-founded ordering that is stable under substitutions.

$\succsim$  and  $\succ$  are compatible:  $\succsim \circ \succ \subseteq \succ$  or  $\succ \circ \succsim \subseteq \succ$ .

(In practice,  $\succ$  is almost always the strict part of the quasi-ordering  $\succsim$ .)

# Reduction Pairs and Argument Filterings

---

Clearly, for any reduction ordering  $\succ$ ,  $(\succeq, \succ)$  is a reduction pair. More general reduction pairs can be obtained using argument filterings:

Argument filtering  $\pi$ :

$$\pi : \Omega \cup \Omega^\# \rightarrow \mathbb{N} \cup \text{list of } \mathbb{N}$$

$$\pi(f) = \begin{cases} i \in \{1, \dots, \text{arity}(f)\}, \text{ or} \\ [i_1, \dots, i_k], \text{ where } 1 \leq i_1 < \dots < i_k \leq \text{arity}(f), \\ 0 \leq k \leq \text{arity}(f) \end{cases}$$

# Reduction Pairs and Argument Filterings

---

Extension to terms:

$$\pi(x) = x$$

$$\pi(f(t_1, \dots, t_n)) = \pi(t_i), \text{ if } \pi(f) = i$$

$$\pi(f(t_1, \dots, t_n)) = f'(\pi(t_{i_1}), \dots, \pi(t_{i_k})), \text{ if } \pi(f) = [i_1, \dots, i_k],$$

where  $f'/k$  is a new function symbol.

# Reduction Pairs and Argument Filterings

---

Let  $\succ$  be a reduction ordering, let  $\pi$  be an argument filtering. Define  $s \succ_{\pi} t$  if and only if  $\pi(s) \succ \pi(t)$  and  $s \sim_{\pi} t$  if and only if  $\pi(s) \succeq \pi(t)$ .

Lemma 1.2:

$(\sim_{\pi}, \succ_{\pi})$  is a reduction pair.

# Reduction Pairs and Argument Filterings

---

For interpretation-based orderings (such as polynomial orderings) the idea of “cutting out” certain subterms can be included directly in the definition of the ordering:

# Reduction Pairs and Argument Filterings

---

## Reduction pairs by interpretation:

Let  $\mathcal{A}$  be a  $\Sigma$ -algebra;

let  $\succ$  be a well-founded strict partial ordering on its universe.

Assume that all interpretations  $f_{\mathcal{A}}$  of function symbols are **weakly monotone**, i.e.,  $a_i \succeq b_i$  implies  $f(a_1, \dots, a_n) \succeq f(b_1, \dots, b_n)$  for all  $a_i, b_i \in U_{\mathcal{A}}$ .

Define  $s \succsim_{\mathcal{A}} t$  if and only if  $\mathcal{A}(\beta)(s) \succeq \mathcal{A}(\beta)(t)$  for all assignments  $\beta : X \rightarrow U_{\mathcal{A}}$ ; define  $s \succ_{\mathcal{A}} t$  if and only if  $\mathcal{A}(\beta)(s) \succ \mathcal{A}(\beta)(t)$  for all assignments  $\beta : X \rightarrow U_{\mathcal{A}}$ .

Then  $(\succsim_{\mathcal{A}}, \succ_{\mathcal{A}})$  is a reduction pair.

## Reduction Pairs and Argument Filterings

---

For polynomial orderings, this definition permits interpretations of function symbols where some variable does not occur at all (e.g.,  $P_f(X_1, X_2) = 2X_1 + 1$  for a *binary* function symbol).

It is no longer required that *every* variable must occur with some positive coefficient.

# Reduction Pairs and Argument Filterings

---

Theorem 1.3 (Arts and Giesl):

Let  $K$  be a cycle in the dependency graph of the TRS  $R$ . If there is a reduction pair  $(\succsim, \succ)$  such that

- $l \succsim r$  for all  $l \rightarrow r \in R$ ,
- $l \succsim r$  or  $l \succ r$  for all  $l \rightarrow r \in K$ ,
- $l \succ r$  for at least one  $l \rightarrow r \in K$ ,

then there is no  $K$ -minimal infinite sequence.

## Reduction Pairs and Argument Filterings

---

The idea can be extended to SCCs in the same way as for the subterm criterion:

Search for a reduction pair  $(\succsim, \succ)$  such that  $l \succsim r$  for all  $l \rightarrow r \in R$  and  $l \succ r$  or  $l \succsim r$  for all DPs  $l \rightarrow r$  in the SCC. Delete all DPs in the SCC for which  $l \succ r$ .

Then re-compute SCCs for the remaining graph and re-start.

# Reduction Pairs and Argument Filterings

---

Example: Consider the following TRS  $R$  from [Arts and Giesl]:

$$\mathit{minus}(x, 0) \rightarrow x \quad (1)$$

$$\mathit{minus}(s(x), s(y)) \rightarrow \mathit{minus}(x, y) \quad (2)$$

$$\mathit{quot}(0, s(y)) \rightarrow 0 \quad (3)$$

$$\mathit{quot}(s(x), s(y)) \rightarrow s(\mathit{quot}(\mathit{minus}(x, y), s(y))) \quad (4)$$

( $R$  is not contained in any simplification ordering, since the left-hand side of rule (4) is embedded in the right-hand side after instantiating  $y$  by  $s(x)$ .)

# Reduction Pairs and Argument Filterings

---

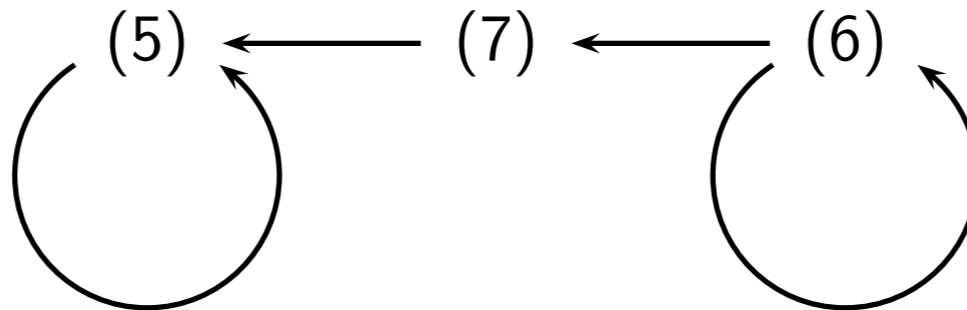
$R$  has three dependency pairs:

$$\mathit{minus}^\sharp(s(x), s(y)) \rightarrow \mathit{minus}^\sharp(x, y) \quad (5)$$

$$\mathit{quot}^\sharp(s(x), s(y)) \rightarrow \mathit{quot}^\sharp(\mathit{minus}(x, y), s(y)) \quad (6)$$

$$\mathit{quot}^\sharp(s(x), s(y)) \rightarrow \mathit{minus}^\sharp(x, y) \quad (7)$$

The dependency graph of  $R$  is



## Reduction Pairs and Argument Filterings

---

There are exactly two SCCs (and also two cycles).

The cycle at (5) can be handled using the subterm criterion with  $\pi(\mathit{minus}^\sharp) = 1$ .

For the cycle at (6) we can use an argument filtering  $\pi$  that maps  $\mathit{minus}$  to 1 and leaves all other function symbols unchanged (that is,  $\pi(g) = [1, \dots, \text{arity}(g)]$  for every  $g$  different from  $\mathit{minus}$ .) After applying the argument filtering, we compare left and right-hand sides using an LPO with precedence  $\mathit{quot} > s$  (the precedence of other symbols is irrelevant). We obtain  $l \succ r$  for (6) and  $l \approx r$  for (1), (2), (3), (4), so the previous theorem can be applied.

## Reduction Pairs and Argument Filterings

---

Alternatively, we can handle the cycle at (5) with a polynomial interpretation with  $P_{minus\#}(X_1, X_2) = X_1$ ,  $P_s(X_1) = X_1 + 1$ ,  $P_{minus}(X_1, X_2) = X_1$ ,  $P_{quot}(X_1, X_2) = X_1$ ,  $P_0 = 1$ . We obtain  $l \succ r$  for (5) and  $l \succsim r$  for (1), (2), (3), (4), so the previous theorem can be applied.

It remains to handle the cycle at (6). We choose a polynomial interpretation with  $P_{quot\#}(X_1, X_2) = X_1$ ,  $P_s(X_1) = X_1 + 1$ ,  $P_{minus}(X_1, X_2) = X_1$ ,  $P_{quot}(X_1, X_2) = X_1$ ,  $P_0 = 1$ . We obtain  $l \succ r$  for (6) and  $l \succsim r$  for (1), (2), (3), (4), so the previous theorem can be applied again.

# DP Processors

---

The methods described so far are particular cases of **DP processors**:

A DP processor

$$\frac{(G, R)}{(G_1, R_1), \dots, (G_n, R_n)}$$

takes a graph  $G$  and a TRS  $R$  as input and produces a set of pairs consisting of a graph and a TRS.

It is sound and complete if there are  $K$ -minimal infinite sequences for  $G$  and  $R$  if and only if there are  $K$ -minimal infinite sequences for at least one of the pairs  $(G_i, R_i)$ .

# DP Processors

---

Examples:

$$\frac{(G, R)}{(SCC_1, R), \dots, (SCC_n, R)}$$

where  $SCC_1, \dots, SCC_n$  are the strongly conn. components of  $G$ .

$$\frac{(G, R)}{(G \setminus N, R)}$$

if there is an SCC of  $G$  and a simple projection  $\pi$  such that  $\pi(l) \succeq \pi(r)$  for all DPs  $l \rightarrow r$  in the SCC, and  $N$  is the set of DPs of the SCC for which  $\pi(l) \triangleright \pi(r)$ .

(and analogously for reduction pairs)

# Innermost Termination

---

The dependency method can also be used for proving termination of **innermost rewriting**:  $s \xrightarrow{i} t$  if  $s \rightarrow_R t$  at position  $p$  and no rule of  $R$  can be applied at a position strictly below  $p$ .

(DP processors for innermost termination are more powerful than for ordinary termination, and for program analysis, innermost termination is usually sufficient.)

## Part 2: Decision Procedures

---

In general, validity (or unsatisfiability) of first-order formulas is undecidable.

To get decidability results, we have to impose restrictions on

- signatures,
- formulas,
- and/or algebras.

## 2.1 Theories and Fragments

---

So far, we have considered the validity or satisfiability of “unstructured” sets of formulas.

We will now split these sets of formulas into two parts:  
a theory (which we keep fixed) and a set of formulas that we consider relative to the theory.

# Theories and Fragments

---

A **first-order theory**  $\mathcal{T}$  is defined by

its signature  $\Sigma = (\Omega, \Pi)$

its axioms, that is, a set of closed  $\Sigma$ -formulas.

(We often use the same symbol  $\mathcal{T}$  for a theory and its set of axioms.)

Note: This is the *syntactic view* of theories. There is also a *semantic view*, where one specifies a class of  $\Sigma$ -algebras  $\mathcal{M}$  and considers  $Th(\mathcal{M})$ , that is, all closed  $\Sigma$ -formulas that hold in the algebras of  $\mathcal{M}$ .

# Theories and Fragments

---

A  $\Sigma$ -algebra that satisfies all axioms of  $\mathcal{T}$  is called a  $\mathcal{T}$ -algebra (or  $\mathcal{T}$ -interpretation).

$\mathcal{T}$  is called **consistent** if there is at least one  $\mathcal{T}$ -algebra.  
(We will only consider consistent theories.)

# Theories and Fragments

---

We can define models, validity, satisfiability, entailment, equivalence, etc., relative to a theory  $\mathcal{T}$ :

A  $\mathcal{T}$ -algebra that is a model of a  $\Sigma$ -formula  $F$  is also called a  $\mathcal{T}$ -model of  $F$ .

A  $\Sigma$ -formula  $F$  is called  $\mathcal{T}$ -valid,  
if  $\mathcal{A}, \beta \models F$  for all  $\mathcal{T}$ -algebras  $\mathcal{A}$  and assignments  $\beta$ .

A  $\Sigma$ -formula  $F$  is called  $\mathcal{T}$ -satisfiable,  
if  $\mathcal{A}, \beta \models F$  for some  $\mathcal{T}$ -algebra and assignment  $\beta$   
(and otherwise  $\mathcal{T}$ -unsatisfiable).

( $\mathcal{T}$ -satisfiability of sets of formulas,  $\mathcal{T}$ -entailment,  $\mathcal{T}$ -equivalence:  
analogously.)

# Theories and Fragments

---

A *fragment* is some syntactically restricted class of  $\Sigma$ -formulas.

Typical restriction: only certain quantifier prefixes are permitted.

## 2.2 Equality

---

Theory of equality:

Signature: arbitrary

Axioms: none

(but the equality predicate  $\approx$  has a fixed interpretation)

Alternatively:

Signature contains a binary predicate symbol  $\sim$  instead of the built-in  $\approx$

Axioms: reflexivity, symmetry, transitivity, congruence for  $\sim$

# Equality

---

In general, satisfiability of first-order formulas w.r.t. equality is undecidable.

However, we will show that it is decidable for *ground* first-order formulas.

Note: It suffices to consider conjunctions of literals.

Arbitrary ground formulas can be converted into DNF;

a formula in DNF is satisfiable if and only if one of its conjunctions is satisfiable.

# Equality

---

Note that our problem can be written in several ways:

An equational clause

$\forall \vec{x} (A_1 \vee \dots \vee A_n \vee \neg B_1 \vee \dots \vee \neg B_k)$  is  $\mathcal{T}$ -valid

iff

$\exists \vec{x} (\neg A_1 \wedge \dots \wedge \neg A_n \wedge B_1 \wedge \dots \wedge B_k)$  is  $\mathcal{T}$ -unsatisfiable

iff

the Skolemized (ground!) formula

$(\neg A_1 \wedge \dots \wedge \neg A_n \wedge B_1 \wedge \dots \wedge B_k)\{\vec{x} \mapsto \vec{c}\}$  is  $\mathcal{T}$ -unsatisfiable

iff

$(A_1 \vee \dots \vee A_n \vee \neg B_1 \vee \dots \vee \neg B_k)\{\vec{x} \mapsto \vec{c}\}$  is  $\mathcal{T}$ -valid

# Equality

---

Other names:

The theory is also known as **EU**F (equality with uninterpreted function symbols).

The decision procedures for the ground fragment are called **congruence closure** algorithms.

# Congruence Closure

---

Goal: check (un-)satisfiability of a ground conjunction

$$u_1 \approx v_1 \wedge \dots \wedge u_n \approx v_n \wedge \neg s_1 \approx t_1 \wedge \dots \wedge \neg s_k \approx t_k$$

Idea:

transform  $E = \{u_1 \approx v_1, \dots, u_n \approx v_n\}$  into an equivalent convergent TRS  $R$  and check whether  $s_i \downarrow_R = t_i \downarrow_R$ .

if  $s_i \downarrow_R = t_i \downarrow_R$  for some  $i$ :

$$s_i \downarrow_R = t_i \downarrow_R \Leftrightarrow s_i \leftrightarrow_E^* t_i \Leftrightarrow E \models s_i \approx t_i \Rightarrow \text{unsat.}$$

if  $s_i \downarrow_R = t_i \downarrow_R$  for no  $i$ :

$T_\Sigma(X)/R = T_\Sigma(X)/E$  is a model of the conjunction  $\Rightarrow \text{sat.}$

# Congruence Closure

---

In principle, one could use Knuth-Bendix completion to convert  $E$  into an equivalent convergent TRS  $R$ .

If done properly (see exercises), Knuth-Bendix completion terminates for ground inputs.

However, for the ground case, one can optimize the general procedure.

# Congruence Closure

---

First step:

Flatten terms:

Introduce new constant symbols  $c_1, c_2, \dots$  for all subterms:

$$g(a, h(h(b))) \approx h(a)$$

is replaced by

$$a \approx c_1 \wedge b \approx c_2 \wedge h(c_2) \approx c_3 \wedge h(c_3) \approx c_4$$

$$\wedge g(c_1, c_4) \approx c_5 \wedge h(c_1) \approx c_6 \wedge c_5 \approx c_6$$

# Congruence Closure

---

Result: only two kinds of equations left.

D-equations:  $f(c_{i_1}, \dots, c_{i_n}) \approx c_{i_0}$  for  $f/n \in \Omega$ ,  $n \geq 0$ .

C-equations:  $c_i \approx c_j$ .

$\Rightarrow$  efficient indexing (e.g., using hash tables),  
obvious termination for D-equations.

# Inference Rules

---

The congruence closure algorithm is presented as a set of inference rules working on a set of equations  $E$  and a set of rules  $R$ :

$$E_0, R_0 \vdash E_1, R_1 \vdash E_2, R_2 \vdash \dots$$

At the beginning,  $E = E_0$  is the set of C-equations and  $R = R_0$  is the set of D-equations oriented left-to-right. At the end,  $E$  should be empty; then  $R$  is the result.

Notation: The formula  $s \overset{\cdot}{\approx} t$  denotes either  $s \approx t$  or  $t \approx s$ .

# Inference Rules

---

*Simplify:*

$$\frac{E \cup \{c \dot{\approx} c'\}, R \cup \{c \rightarrow c''\}}{E \cup \{c'' \dot{\approx} c'\}, R \cup \{c \rightarrow c''\}}$$

*Delete:*

$$\frac{E \cup \{c \approx c\}, R}{E, R}$$

*Orient:*

$$\frac{E \cup \{c \dot{\approx} c'\}, R}{E, R \cup \{c \rightarrow c'\}} \quad \text{if } c \succ c'$$

# Inference Rules

---

*Collapse:*

$$\frac{E, R \cup \{t[c]_p \rightarrow c', c \rightarrow c''\}}{E, R \cup \{t[c'']_p \rightarrow c', c \rightarrow c''\}} \quad \text{if } p \neq \varepsilon$$

*Deduce:*

$$\frac{E, R \cup \{t \rightarrow c, t \rightarrow c'\}}{E \cup \{c \approx c'\}, R \cup \{t \rightarrow c\}}$$

Note: for ground rewrite rules, critical pair computation does not involve substitution. Therefore, every critical pair computation can be replaced by a simplification, either using Deduce or Collapse.

# Inference Rules

---

Theorem 2.1:

Let  $E_0$  be a finite set of C-equations, let  $R_0$  be a finite set of D-equations oriented left-to-right w.r.t.  $\succ$ , and let  $\succ$  be a total ordering on constants. Then the inference system terminates with a final state  $(E_n, R_n)$  where  $E_n = \emptyset$ ,  $R_n$  is terminating and confluent, and  $\approx_{E_0 \cup R_0}$  equals  $\approx_{R_n}$ .

# Strategy

---

The inference rules are applied according to the following strategy:

- (1) If there is an equation in  $E$ , use Simplify as long as possible for this equation, then use either Delete or Orient. Repeat until  $E$  is empty.
- (2) If Collapse is applicable, apply it, if now Deduce is applicable, apply it as well. Repeat until Collapse is no longer applicable.
- (3) If  $E$  is non-empty, go to (1), otherwise return  $R$ .

# Implementation

---

Instead of fixing the ordering  $\succ$  in advance, it is preferable to define it on the fly during the algorithm:

If we orient an equation  $c \approx c'$  between two constant symbols, we try to make that constant symbol larger that occurs less often in  $R$   
 $\Rightarrow$  fewer Collapse steps.

Additionally:

Use various index data structures so that all the required operations can be performed efficiently.

Use a union-find data structure to represent the equivalence classes encoded by the C-rules.

# Implementation

---

Average runtime for an implementation using hash tables:

$O(m \log m)$ , where  $m$  is the number of edges in the graph representation of the initial C and D-equations.

## One Small Problem

---

The inference rules are sound in the usual sense: The conclusions are entailed by the premises, so every  $\mathcal{T}$ -model of the premises is a  $\mathcal{T}$ -model of the conclusions.

For the initial flattening, however, we get a weaker result: We have to *extend* the  $\mathcal{T}$ -models of the original equations to obtain models of the flattened equations.

That is, we get a new algebra with the same universe as the old one, with the same interpretations for old functions and predicate symbols, but with appropriately chosen interpretations for the new constants.

## One Small Problem

---

Consequently, the relations  $\approx_E$  and  $\approx_R$  for the original  $E$  and the final  $R$  are not the same. For instance,  $c_3 \approx_E c_7$  does not hold, but  $c_3 \approx_R c_7$  may hold.

On the other hand, the model extension preserves the universe and the interpretations for old symbols. Therefore, if  $s$  and  $t$  are terms over the old symbols, we have  $s \approx_E t$  iff  $s \approx_R t$ .

This is sufficient for our purposes: The terms  $s_i$  and  $t_i$  that we want to normalize using  $R$  do not contain new symbols.

## Other Predicate Symbols

---

If the initial ground conjunction contains also non-equational literals  $[\neg] P(t_1, \dots, t_n)$ , treat these like equational literals  $[\neg] P(t_1, \dots, t_n) \approx true$ .  
Then use the same algorithm as before.

# History

---

Congruence closure algorithms have been published, among others, by Shostak (1978). by Nelson and Oppen (1980), and by Downey, Sethi and Tarjan (1980).

Kapur (1997) showed that Shostak's algorithm can be described as a completion procedure.

Bachmair and Tiwari (2000) did this also for the Nelson/Oppen and the Downey/Sethi/Tarjan algorithm.

The algorithm presented here is the Downey/Sethi/Tarjan algorithm in the presentation of Bachmair and Tiwari.

## 2.3 Linear Rational Arithmetic

---

There are several ways to define **linear rational arithmetic**.

We need at least the following signature:

$$\Sigma = (\{0/0, 1/0, +/2\}, \{</2\})$$

and the pre-defined binary predicate  $\approx$ .

# Linear Rational Arithmetic

---

The equational part of linear rational arithmetic is described by the theory of **divisible torsion-free abelian groups**:

$$\forall x, y, z (x + (y + z) \approx (x + y) + z) \quad (\text{associativity})$$

$$\forall x, y (x + y \approx y + x) \quad (\text{commutativity})$$

$$\forall x (x + 0 \approx x) \quad (\text{identity})$$

$$\forall x \exists y (x + y \approx 0) \quad (\text{inverse})$$

$$\text{For all } n \geq 1: \forall x (\underbrace{x + \dots + x}_{n \text{ times}} \approx 0 \rightarrow x \approx 0) \quad (\text{torsion-freeness})$$

$$\text{For all } n \geq 1: \forall x \exists y (\underbrace{y + \dots + y}_{n \text{ times}} \approx x) \quad (\text{divisibility})$$

$$\neg 1 \approx 0 \quad (\text{non-triviality})$$

# Linear Rational Arithmetic

---

Note: Quantification over natural numbers is not part of our language. We really need infinitely many axioms for torsion-freeness and divisibility.

# Linear Rational Arithmetic

---

By adding the axioms of a compatible strict total ordering, we define **ordered divisible abelian groups**:

$$\forall x (\neg x < x) \quad (\text{irreflexivity})$$

$$\forall x, y, z (x < y \wedge y < z \rightarrow x < z) \quad (\text{transitivity})$$

$$\forall x, y (x < y \vee y < x \vee x \approx y) \quad (\text{totality})$$

$$\forall x, y, z (x < y \rightarrow x + z < y + z) \quad (\text{compatibility})$$

$$0 < 1 \quad (\text{non-triviality})$$

# Linear Rational Arithmetic

---

Note: The second non-triviality axiom renders the first one superfluous. Moreover, as soon as we add the axioms of compatible strict total orderings, torsion-freeness can be omitted. Every ordered divisible abelian group is obviously torsion-free.

In fact the converse holds: Every torsion-free abelian group can be ordered (F.-W. Levi 1913).

Examples:  $\mathbb{Q}$ ,  $\mathbb{R}$ ,  $\mathbb{Q}^n$ ,  $\mathbb{R}^n$ , ...

# Linear Rational Arithmetic

---

The signature can be extended by further symbols:

$\leq/2, >/2, \geq/2, \neq/2$ : defined using  $<$  and  $\approx$

$-/1$ : Skolem function for inverse axiom

$-/2$ : defined using  $+/2$  and  $-/1$

$\text{div}_n/1$ : Skolem functions for divisibility axiom for all  $n \geq 1$ .

$\text{mult}_n/1$ : defined by  $\forall x (\text{mult}_n(x) \approx \underbrace{x + \cdots + x}_{n \text{ times}})$  for all  $n \geq 1$ .

$\text{mult}_q/1$ : defined using  $\text{mult}_n, \text{div}_n, -$  for all  $q \in \mathbb{Q}$ .

(We usually write  $q \cdot t$  or  $qt$  instead of  $\text{mult}_q(t)$ .)

$q/0$  (for  $q \in \mathbb{Q}$ ): defined by  $q \approx q \cdot 1$ .

# Linear Rational Arithmetic

---

When  $\cdot$  is considered as a binary operator, (ordered) divisible torsion-free abelian groups correspond to (ordered) rational vector spaces.

# Fourier-Motzkin Quantifier Elimination

---

Linear rational arithmetic permits **quantifier elimination**: every formula  $\exists x F$  or  $\forall x F$  in linear rational arithmetic can be converted into an equivalent formula without the variable  $x$ .

The method was discovered in 1826 by J. Fourier and re-discovered by T. Motzkin in 1936.

# Fourier-Motzkin Quantifier Elimination

---

Observation: Every literal over the variables  $x, y_1, \dots, y_n$  can be converted into an ODAG-equivalent literal  $x \sim t[\vec{y}]$  or  $0 \sim t[\vec{y}]$ , where  $\sim \in \{<, >, \leq, \geq, \approx, \neq\}$  and  $t[\vec{y}]$  has the form  $\sum_i q_i \cdot y_i + q_0$ .

In other words, we can either eliminate  $x$  completely or isolate it on one side of the literal, and we can replace every negative ordering literal by a positive one.

Moreover, we can convert every  $\neq$ -literal into an ODAG-equivalent disjunction of two  $<$ -literals.

# Fourier-Motzkin Quantifier Elimination

---

We first consider existentially quantified conjunctions of atoms.

If the conjunction contains an equation  $x \approx t[\vec{y}]$ , we can eliminate the quantifier  $\exists x$  by substitution:

$$\exists x (x \approx t[\vec{y}] \wedge F)$$

is equivalent to

$$F \{x \mapsto t[\vec{y}]\}$$

# Fourier-Motzkin Quantifier Elimination

---

If  $x$  occurs only in inequations, then

$$\begin{aligned} \exists x \left( \bigwedge_i x < s_i(\vec{y}) \wedge \bigwedge_j x \leq t_j(\vec{y}) \right. \\ \left. \wedge \bigwedge_k x > u_k(\vec{y}) \wedge \bigwedge_l x \geq v_l(\vec{y}) \wedge \bigwedge_m 0 \sim_m w_m(\vec{y}) \right) \end{aligned}$$

is equivalent to

$$\begin{aligned} \bigwedge_i \bigwedge_k s_i(\vec{y}) > u_k(\vec{y}) \wedge \bigwedge_j \bigwedge_k t_j(\vec{y}) > u_k(\vec{y}) \\ \wedge \bigwedge_i \bigwedge_l s_i(\vec{y}) > v_l(\vec{y}) \wedge \bigwedge_j \bigwedge_l t_j(\vec{y}) \geq v_l(\vec{y}) \\ \wedge \bigwedge_m 0 \sim_m w_m(\vec{y}) \end{aligned}$$

Proof: ( $\Rightarrow$ ) by transitivity;

( $\Leftarrow$ ) take  $\frac{1}{2}(\min\{s_i, t_j\} + \max\{u_k, v_l\})$  as a witness.

# Fourier-Motzkin Quantifier Elimination

---

Extension to arbitrary formulas:

Transform into prenex formula;

if innermost quantifier is  $\exists$ : transform matrix into DNF and move  $\exists$  into disjunction;

if innermost quantifier is  $\forall$ : replace  $\forall x F$  by  $\neg \exists x \neg F$ , then eliminate  $\exists$ .

# Fourier-Motzkin Quantifier Elimination

---

Consequence: every closed formula over the signature of ODAGs is ODAG-equivalent to either  $\top$  or  $\perp$ .

Consequence: ODAGs are a **complete** theory, i.e., every closed formula over the signature of ODAGs is either valid or unsatisfiable w.r.t. ODAGs.

# Fourier-Motzkin Quantifier Elimination

---

Consequence: every closed formula over the signature of ODAGs holds either in all ODAGs or in no ODAG.

ODAGs are indistinguishable by first-order formulas over the signature of ODAGs.

(These properties do not hold for extended signatures!)

# Fourier-Motzkin: Complexity

---

One FM-step for  $\exists$ :

formula size grows quadratically, therefore  $O(n^2)$  runtime.

$m$  quantifiers  $\exists \dots \exists$ :

naive implementation produces a doubly exponential number of inequations, therefore needs  $O(n^{2^m})$  runtime  
(the number of *necessary* inequations grows only exponentially, though).

$m$  quantifiers  $\exists \forall \exists \forall \dots \exists$ :

CNF/DNF conversion (exponential!) required after each step;  
therefore non-elementary runtime.

# Loos-Weispfenning Quantifier Elimination

---

A more efficient way to eliminate quantifiers in linear rational arithmetic was developed by R. Loos and V. Weispfenning (1993).

The method is also known as “test point method” or “virtual substitution method”.

# Loos-Weispfenning Quantifier Elimination

---

For simplicity, we consider only one particular ODAG, namely  $\mathbb{Q}$  (as we have seen above, the results are the same for all ODAGs).

# Loos-Weispfenning Quantifier Elimination

---

Let  $F(x, \vec{y})$  be a *positive* boolean combination of linear (in-)equations  $x \sim_i s_i(\vec{y})$  and  $0 \sim_j s'_j(\vec{y})$  with  $\sim_i, \sim_j \in \{\approx, \neq, <, \leq, >, \geq\}$ , that is, a formula built from linear (in-)equations,  $\wedge$  and  $\vee$  (but without  $\neg$ ).

Goal: Find a *finite* set  $T$  of “test points” so that

$$\exists x F(x, \vec{y}) \quad \models \quad \bigvee_{t \in T} F(x, \vec{y}) \{x \mapsto t\}$$

In other words: We want to replace the infinite disjunction  $\exists x$  by a finite disjunction.

# Loos-Weispfenning Quantifier Elimination

---

If we keep the values of the variables  $\vec{y}$  fixed, then we can consider  $F$  as a function  $F : x \mapsto F(x, \vec{y})$  from  $\mathbb{Q}$  to  $\{0, 1\}$ .

The value of each of the atoms  $s_i(\vec{y}) \sim_i x$  changes only at  $s_i(\vec{y})$ , and the value of  $F$  can only change if the value of one of its atoms changes.

# Loos-Weispfenning Quantifier Elimination

---

Let  $\delta(\vec{y}) = \min\{ |s_i(\vec{y}) - s_j(\vec{y})| \mid s_i(\vec{y}) \neq s_j(\vec{y}) \}$

$F$  is a piecewise constant function; more precisely, the set of all  $x$  with  $F(x, \vec{y}) = 1$  is a finite union of intervals. (The union may be empty, the individual intervals may be finite or infinite and open or closed.)

Moreover, each of the intervals has either length 0 (i.e., it consists of one point), or its length is at least  $\delta(\vec{y})$ .

# Loos-Weispfenning Quantifier Elimination

---

If the set of all  $x$  for which  $F(x, \vec{y})$  is 1 is non-empty, then

- (i)  $F(x, \vec{y}) = 1$  for all  $x \leq r(\vec{y})$  for some  $r(\vec{y}) \in \mathbb{Q}$
- (ii) or there is some point where the value of  $F(x, \vec{y})$  switches from 0 to 1 when we traverse the real axis from  $-\infty$  to  $+\infty$ .

We use this observation to construct a set of test points.

We start with some “sufficiently small” test point  $r(\vec{y})$  to take care of case (i).

# Loos-Weispfenning Quantifier Elimination

---

For case (ii), we observe that  $F(x, \vec{y})$  can only switch from 0 to 1 if one of the atoms switches from 0 to 1. (We consider only *positive* boolean combinations of atoms, and  $\wedge$  and  $\vee$  are monotonic w.r.t. truth values.)

$x \leq s_i(\vec{y})$  and  $x < s_i(\vec{y})$  do not switch from 0 to 1 when  $x$  grows.

$x \geq s_i(\vec{y})$  and  $x \approx s_i(\vec{y})$  switch from 0 to 1 at  $s_i(\vec{y})$

$\Rightarrow s_i(\vec{y})$  is a test point.

$x > s_i(\vec{y})$  and  $x \not\approx s_i(\vec{y})$  switch from 0 to 1 “right after”  $s_i(\vec{y})$

$\Rightarrow s_i(\vec{y}) + \varepsilon$  (for some  $0 < \varepsilon < \delta(\vec{y})$ ) is a test point.

# Loos-Weispfenning Quantifier Elimination

---

If  $r(\vec{y})$  is sufficiently small and  $0 < \varepsilon < \delta(\vec{y})$ , then

$$T := \{r(\vec{y})\} \cup \{s_i(\vec{y}) \mid \sim_i \in \{\geq, \approx\}\} \\ \cup \{s_i(\vec{y}) + \varepsilon \mid \sim_i \in \{>, \neq\}\}.$$

is a set of test points.

Problem:

We don't know how small  $r(\vec{y})$  has to be for case (i), and we don't know  $\delta(\vec{y})$  for case (ii).

# Loos-Weispfenning Quantifier Elimination

---

Idea:

We consider the limits for  $r \rightarrow -\infty$  and for  $\varepsilon \searrow 0$ , that is, we redefine

$$\begin{aligned} T := & \{-\infty\} \cup \{s_i(\vec{y}) \mid \sim_i \in \{\geq, \approx\}\} \\ & \cup \{s_i(\vec{y}) + \varepsilon \mid \sim_i \in \{>, \neq\}\}. \end{aligned}$$

How can we eliminate the infinitesimals  $\infty$  and  $\varepsilon$  when we substitute elements of  $T$  for  $x$ ?

# Loos-Weispfenning Quantifier Elimination

---

Virtual substitution:

$$(x < s(\vec{y})) \{x \mapsto -\infty\} := \lim_{r \rightarrow -\infty} (r < s(\vec{y})) = \top$$

$$(x \leq s(\vec{y})) \{x \mapsto -\infty\} := \lim_{r \rightarrow -\infty} (r \leq s(\vec{y})) = \top$$

$$(x > s(\vec{y})) \{x \mapsto -\infty\} := \lim_{r \rightarrow -\infty} (r > s(\vec{y})) = \perp$$

$$(x \geq s(\vec{y})) \{x \mapsto -\infty\} := \lim_{r \rightarrow -\infty} (r \geq s(\vec{y})) = \perp$$

$$(x \approx s(\vec{y})) \{x \mapsto -\infty\} := \lim_{r \rightarrow -\infty} (r \approx s(\vec{y})) = \perp$$

$$(x \not\approx s(\vec{y})) \{x \mapsto -\infty\} := \lim_{r \rightarrow -\infty} (r \not\approx s(\vec{y})) = \top$$

# Loos-Weispfenning Quantifier Elimination

---

Virtual substitution:

$$(x < s(\vec{y})) \{x \mapsto u + \varepsilon\} := \lim_{\varepsilon \searrow 0} (u + \varepsilon < s(\vec{y})) = (u < s(\vec{y}))$$

$$(x \leq s(\vec{y})) \{x \mapsto u + \varepsilon\} := \lim_{\varepsilon \searrow 0} (u + \varepsilon \leq s(\vec{y})) = (u < s(\vec{y}))$$

$$(x > s(\vec{y})) \{x \mapsto u + \varepsilon\} := \lim_{\varepsilon \searrow 0} (u + \varepsilon > s(\vec{y})) = (u \geq s(\vec{y}))$$

$$(x \geq s(\vec{y})) \{x \mapsto u + \varepsilon\} := \lim_{\varepsilon \searrow 0} (u + \varepsilon \geq s(\vec{y})) = (u \geq s(\vec{y}))$$

$$(x \approx s(\vec{y})) \{x \mapsto u + \varepsilon\} := \lim_{\varepsilon \searrow 0} (u + \varepsilon \approx s(\vec{y})) = \perp$$

$$(x \not\approx s(\vec{y})) \{x \mapsto u + \varepsilon\} := \lim_{\varepsilon \searrow 0} (u + \varepsilon \not\approx s(\vec{y})) = \top$$

# Loos-Weispfenning Quantifier Elimination

---

We have traversed the real axis from  $-\infty$  to  $+\infty$ . Alternatively, we can traverse it from  $+\infty$  to  $-\infty$ . In this case, the test points are

$$\begin{aligned} T' := & \{+\infty\} \cup \{s_i(\vec{y}) \mid \sim_i \in \{\leq, \approx\}\} \\ & \cup \{s_i(\vec{y}) - \varepsilon \mid \sim_i \in \{<, \neq\}\}. \end{aligned}$$

Infinitesimals are eliminated in a similar way as before.

In practice: Compute both  $T$  and  $T'$  and take the smaller set.

## Loos-Weispfenning Quantifier Elimination

---

For a universally quantified formulas  $\forall x F$ , we replace it by  $\neg \exists x \neg F$ , push inner negation downwards, and then continue as before.

Note that there is no CNF/DNF transformation required. Loos-Weispfenning quantifier elimination works on arbitrary positive formulas.

# Loos-Weispfenning: Complexity

---

One LW-step for  $\exists$  or  $\forall$ :

as the number of test points is at most one plus the number of atoms (one plus half of the number of atoms, if there are only ordering literals), the formula size grows quadratically; therefore  $O(n^2)$  runtime.

# Loos-Weispfenning: Complexity

---

Multiple quantifiers of the same kind:

$$\exists x_2 \exists x_1. F(x_1, x_2, \vec{y})$$

$$\rightsquigarrow \exists x_2. \left( \bigvee_{t_1 \in T_1} F(x_1, x_2, \vec{y}) \{x_1 \mapsto t_1\} \right)$$

$$\rightsquigarrow \bigvee_{t_1 \in T_1} \left( \exists x_2. F(x_1, x_2, \vec{y}) \{x_1 \mapsto t_1\} \right)$$

$$\rightsquigarrow \bigvee_{t_1 \in T_1} \bigvee_{t_2 \in T_2} \left( F(x_1, x_2, \vec{y}) \{x_1 \mapsto t_1\} \{x_2 \mapsto t_2\} \right)$$

# Loos-Weispfenning: Complexity

---

$m$  quantifiers  $\exists \dots \exists$  or  $\forall \dots \forall$ :

formula size is multiplied by  $n$  in each step, therefore  $O(n^{m+1})$  runtime.

$m$  quantifiers  $\exists \forall \exists \forall \dots \exists$ :

doubly exponential runtime.

Note: The formula resulting from a LW-step is usually highly redundant; so an efficient implementation must make heavy use of simplification techniques.

## 2.4 Existentially-quantified LRA

---

So far, we have considered formulas that may contain free, existentially quantified, and universally quantified variables.

For the special case of conjunction of linear inequations in which *all* variables are existentially quantified, there are more efficient methods available.

Main idea: reduce satisfiability problem to optimization problem.

# Linear Optimization

---

Goal:

Solve a linear optimization (also called: linear programming) problem for given numbers  $a_{ij}, b_i, c_j \in \mathbb{R}$ :

$$\begin{aligned} &\text{maximize } \sum_{1 \leq j \leq n} c_j x_j \\ &\text{for } \bigwedge_{1 \leq i \leq m} \sum_{1 \leq j \leq n} a_{ij} x_j \leq b_i \end{aligned}$$

or in vectorial notation:

$$\begin{aligned} &\text{maximize } \vec{c}^\top \vec{x} \\ &\text{for } A\vec{x} \leq \vec{b} \end{aligned}$$

# Linear Optimization

---

Simplex algorithm:

Developed independently by Kantorovich (1939), Dantzig (1948).

Polynomial-time average-case complexity; worst-case time complexity is exponential, though.

Interior point methods:

First algorithm by Karmarkar (1984).

Polynomial-time worst-case complexity (but large constants).

In practice: no clear winner.

# Linear Optimization

---

Implementations:

GLPK (GNU Linear Programming Kit),

Gurobi.

# Linear Optimization

---

Main idea of Simplex:

$A\vec{x} \leq \vec{b}$  describes a convex polyhedron.

Pick one vertex of the polyhedron,  
then follow the edges of the polyhedron towards an optimal solution.

By convexity, the local optimum found in this way is also a global optimum.

Details: see special lecture on optimization.

# Linear Optimization

---

A variant of the Simplex algorithm can be used to check the satisfiability of conjunctions of linear inequations.

See Martin Bromberger's slides on the Simplex algorithm.

## 2.5 Non-linear Real Arithmetic

---

Tarski (1951): Quantifier elimination is possible for *non-linear* real arithmetic (or more generally, for real-closed fields).

His algorithm had non-elementary complexity, however.

An improved algorithm by Collins (1975) (with further improvements by Hong) has doubly exponential complexity: Cylindrical algebraic decomposition (CAD).

Implementation: QEPCAD.

# Cylindrical Algebraic Decomposition

---

Given: First-order formula over atoms of the form  $f_i(\vec{x}) \sim 0$ , where the  $f_i$  are polynomials over variables  $\vec{x}$ .

Goal: Decompose  $\mathbb{R}^n$  into a finite number of regions such that all polynomials have invariant sign on every region  $X$ :

$$\begin{aligned} \forall i \ ( \forall \vec{x} \in X. f_i(\vec{x}) < 0 \\ \vee \forall \vec{x} \in X. f_i(\vec{x}) = 0 \\ \vee \forall \vec{x} \in X. f_i(\vec{x}) > 0 ) \end{aligned}$$

Note: Implementation needs exact arithmetic using algebraic numbers (i.e., zeroes of univariate polynomials with integer coefficients).

## 2.6 Real Arithmetic incl. Transcendental Fctns.

---

Real arithmetic with exp/log: decidability unknown.

Real arithmetic with trigonometric functions: undecidable

The following formula holds exactly if  $x \in \mathbb{Z}$ :

$$\exists y (\sin(y) = 0 \wedge 3 < y \wedge y < 4 \wedge \sin(x \cdot y) = 0)$$

(note that necessarily  $y = \pi$ ).

Consequence: Peano arithmetic (which is undecidable) can be encoded in real arithmetic with trigonometric functions.

## Real Arithmetic incl. Transcendental Fctns.

---

However, real arithmetic with transcendental functions is decidable for formulas that are *stable under perturbations*, i.e., whose truth value does not change if numeric constants are modified by some sufficiently small  $\varepsilon$ .

Example:

Stable under perturbations:  $\exists x x^2 \leq 5$

Not stable under perturbations:  $\exists x x^2 \leq 0$

(Formula is true, but if we subtract an arbitrarily small  $\varepsilon > 0$  from the right-hand side, it becomes false.)

## Real Arithmetic incl. Transcendental Fctns.

---

Unsatisfactory from a mathematical point of view, but sufficient for engineering applications (where stability under perturbations is necessary anyhow).

Approach:

Interval arithmetic + interval bisection if necessary (Ratschan).

Sound for general formulas; complete for formulas that are stable under perturbations; may loop forever if the formula is not stable under perturbations.

## 2.7 Linear Integer Arithmetic

---

Linear integer arithmetic = Presburger arithmetic.

Decidable (Presburger, 1929), but quantifier elimination is only possible if additional divisibility operators are present:

$\exists x (y = 2x)$  is equivalent to  $\text{divides}(2, y)$  but not to any quantifier-free formula over the base signature.

Cooper (1972): Quantifier elimination procedure.

Oppen (1978): Cooper's algorithm runs in triple exponential for arbitrarily quantified formulas.

# The Omega Test

---

Omega test (Pugh, 1991): variant of Fourier–Motzkin for conjunctions of (in-)equations in linear integer arithmetic.

Idea:

- Perform easy transformations, e.g.:

$$3x + 6y \leq 8 \mapsto 3x + 6y \leq 6 \mapsto x + 2y \leq 2$$

$$3x + 6y = 8 \mapsto \perp$$

(since  $3x + 6y$  must be divisible by 3).

- Eliminate equations  
(easy, if one coefficient is 1; tricky otherwise).

# The Omega Test

---

- If only inequations are left:
  - no real solutions  $\rightarrow$  unsatisfiable for  $\mathbb{Z}$
  - “sufficiently many” real solutions  $\rightarrow$  satisfiable for  $\mathbb{Z}$
  - otherwise: branch

# The Omega Test

---

What does “sufficiently many” mean?

Consider inequations  $ax \leq s$  and  $bx \geq t$  with  $a, b \in \mathbb{N}^{>0}$  and polynomials  $s, t$ .

If these inequations have real solutions, the interval of solutions ranges from  $\frac{1}{b}t$  to  $\frac{1}{a}s$ .

The longest possible interval of this kind that does not contain any integer number ranges from  $i + \frac{1}{b}$  to  $i + 1 - \frac{1}{a}$  for some  $i \in \mathbb{Z}$ ; it has the length  $1 - \frac{1}{a} - \frac{1}{b}$ .

# The Omega Test

---

Consequence:

If  $\frac{1}{a}s > \frac{1}{b}t + (1 - \frac{1}{a} - \frac{1}{b})$ , or equivalently,  $bs \geq at + ab - a - b + 1$  is satisfiable, then the original problem must have integer solutions.

It remains to consider the case that  $bs \geq at$  is satisfiable (hence there are real solutions) but  $bs \geq at + ab - a - b + 1$  is not (hence the interval of real solutions need not contain an integer).

# The Omega Test

---

In the latter case,  $bs \leq at + ab - a - b$  holds, hence for every solution of the original problem:

$$t \leq bx \leq \frac{b}{a}s \leq t + (b - 1 - \frac{b}{a})$$

and if  $x$  is an integer,  $t \leq bx \leq t + \lfloor b - 1 - \frac{b}{a} \rfloor$

⇒ Branch non-deterministically:

Add one of the equations  $bx = t + i$

for  $i \in \{0, \dots, \lfloor b - 1 - \frac{b}{a} \rfloor\}$ .

Alternatively, if  $b > a$ :

Add one of the equations  $ax = s - i$

for  $i \in \{0, \dots, \lfloor a - 1 - \frac{a}{b} \rfloor\}$ .

# The Omega Test

---

Note: Efficiency depends highly on the size of coefficients.

In applications from program verification, there is almost always some variable with a very small coefficient.

If all coefficients are large, the branching step gets expensive.

# Branch-and-Cut

---

Alternative approach: Reduce satisfiability problem to optimization problem (like Simplex).

ILP, MILP: (mixed) integer linear programming.

# Branch-and-Cut

---

Two basic approaches:

Branching:

If the simplex algorithm finds a solution with  $x = 2.7$ , add the inequation  $x \leq 2$  or the inequation  $x \geq 3$ .

Cutting planes:

Derive an inequation that holds for all real solutions, then round it to obtain an inequation that holds for all integer solutions, but not for the real solution found previously.

# Branch-and-Cut

---

Example:

$$\text{Given: } 2x - 3y \leq 1$$

$$2x + 3y \leq 5$$

$$-5x - 4y \leq -7$$

Simplex finds an extremal solution  $x = \frac{3}{2}$ ,  $y = \frac{2}{3}$ .

From the first two inequations, we see that  $4x \leq 6$ ,  
hence  $x \leq \frac{3}{2}$ . If  $x \in \mathbb{Z}$ , we conclude  $x = \lfloor x \rfloor \leq \lfloor \frac{3}{2} \rfloor = 1$ .

$\Rightarrow$  Add the inequation  $x \leq 1$ , which holds for all integer solutions, but cuts off the solution  $(\frac{3}{2}, \frac{2}{3})$ .

# Branch-and-Cut

---

In practice:

Use both: Alternate between branching and cutting steps.

Better performance than the individual approaches.

## 2.8 Difference Logic

---

Difference Logic (DL):

Fragment of linear rational or integer arithmetic.

Formulas: conjunctions of atoms  $x - y < c$  or  $x - y \leq c$ ,  
 $x, y \in X$ ,  $c \in \mathbb{Q}$  (or  $c \in \mathbb{Z}$ ).

One special variable  $x_0$  whose value is fixed to 0 is permitted;  
this allows to express atoms like  $x < 3$  in the form  $x - x_0 < 3$ .

# Difference Logic

---

Solving difference logic:

Let  $F$  be a conjunction in DL.

For simplicity: only non-strict inequalities.

Define a weighted graph  $G$ :

Vertices  $V$ : Variables in  $F$ .

Edges  $E$ :  $x - y \leq c \rightsquigarrow$  edge  $(x, y)$  with weight  $c$ .

Theorem:  $F$  is unsatisfiable iff  $G$  has a negative cycle.

Can be checked in  $O(|V| \cdot |E|)$  using the Bellman-Ford algorithm.

## 2.9 C-Arithmetic

---

In languages like C: Bounded integer arithmetic (modulo  $2^n$ ),  
in device drivers also combined with bitwise operations.

Bit-Blasting (encode everything as boolean circuits, use CDCL):

Naive encoding: possible, but often too inefficient.

If combined with over-/underapproximation techniques (Bryant, Kroening, et al.): successful.

## 2.10 Decision Procedures for Data Structures

---

There are decision procedures for, e.g.,

Arrays (read, write)

Lists (car, cdr, cons)

Sets or multisets with cardinalities

Bitvectors

Note: There are usually restrictions on quantifications. Unrestricted universal quantification can lead to undecidability.

## 2.11 Combining Decision Procedures

---

Problem:

Let  $\mathcal{T}_1$  and  $\mathcal{T}_2$  be first-order theories over the signatures  $\Sigma_1$  and  $\Sigma_2$ .

Assume that we have decision procedures for the satisfiability of existentially quantified formulas (or the validity of universally quantified formulas) w.r.t.  $\mathcal{T}_1$  and  $\mathcal{T}_2$ .

Can we combine them to get a decision procedure for the satisfiability of existentially quantified formulas w.r.t.  $\mathcal{T}_1 \cup \mathcal{T}_2$ ?

# Combining Decision Procedures

---

General assumption:

$\Sigma_1$  and  $\Sigma_2$  are disjoint.

The only symbol shared by  $\mathcal{T}_1$  and  $\mathcal{T}_2$  is built-in equality.

We consider only conjunctions of literals.

For general formulas, convert to DNF first and consider each conjunction individually.

# Abstraction

---

To be able to use the individual decision procedures, we have to transform the original formula in such a way that each atom contains only symbols of one of the signatures (plus variables).

This process is known as **variable abstraction** or **purification**.

# Abstraction

---

We apply the following rule as long as possible:

$$\frac{\exists \vec{x} (F[t])}{\exists \vec{x}, y (F[y] \wedge t \approx y)}$$

if the top symbol of  $t$  belongs to  $\Sigma_i$  and  $t$  occurs in  $F$  directly below a  $\Sigma_j$ -symbol or in a (positive or negative) equation  $s \approx t$  where the top symbol of  $s$  belongs to  $\Sigma_j$  ( $i \neq j$ ), and if  $y$  is a new variable.

It is easy to see that the original and the purified formula are equivalent.

# Stable Infiniteness

---

Problem:

Even if the  $\Sigma_1$ -formula  $F_1$  and the  $\Sigma_2$ -formula  $F_2$  do not share any symbols (not even variables), and if  $F_1$  is  $\mathcal{T}_1$ -satisfiable and  $F_2$  is  $\mathcal{T}_2$ -satisfiable, we cannot conclude that  $F_1 \wedge F_2$  is  $(\mathcal{T}_1 \cup \mathcal{T}_2)$ -satisfiable.

# Stable Infiniteness

---

Example:

Consider

$$\mathcal{T}_1 = \{\forall x, y, z (x \approx y \vee x \approx z \vee y \approx z)\}$$

and

$$\mathcal{T}_2 = \{\exists x, y, z (x \not\approx y \wedge x \not\approx z \wedge y \not\approx z)\}.$$

All  $\mathcal{T}_1$ -models have at most two elements, and all  $\mathcal{T}_2$ -models have at least three elements.

Since  $\mathcal{T}_1 \cup \mathcal{T}_2$  is contradictory, there are no  $(\mathcal{T}_1 \cup \mathcal{T}_2)$ -satisfiable formulas.

## Stable Infiniteness

---

To ensure that  $\mathcal{T}_1$ -models and  $\mathcal{T}_2$ -models can be combined to  $(\mathcal{T}_1 \cup \mathcal{T}_2)$ -models, we require that both  $\mathcal{T}_1$  and  $\mathcal{T}_2$  are stably infinite.

A first-order theory  $\mathcal{T}$  is called **stably infinite**, if every existentially quantified formula that has a  $\mathcal{T}$ -model has also a  $\mathcal{T}$ -model with a (countably) infinite universe.

Note: By the Löwenheim–Skolem theorem, “countable” is redundant here.

## Shared Variables

---

Even if  $\exists \vec{x} F_1$  is  $\mathcal{T}_1$ -satisfiable and  $\exists \vec{x} F_2$  is  $\mathcal{T}_2$ -satisfiable, it can happen that  $\exists \vec{x} (F_1 \wedge F_2)$  is not  $(\mathcal{T}_1 \cup \mathcal{T}_2)$ -satisfiable, for instance because the shared variables  $x$  and  $y$  must be equal in all  $\mathcal{T}_1$ -models of  $\exists \vec{x} F_1$  and different in all  $\mathcal{T}_2$ -models of  $\exists \vec{x} F_2$ .

# Shared Variables

---

Example:

Consider

$$F_1 = (x + (-y) \approx 0),$$

and

$$F_2 = (f(x) \neq f(y))$$

where  $\mathcal{T}_1$  is linear rational arithmetic and  $\mathcal{T}_2$  is EUF.

We must exchange information about shared variables to detect the contradiction.

## The Nelson–Oppen Algorithm (Non-determ.)

Suppose that  $\exists \vec{x} F$  is a purified conjunction of  $\Sigma_1$  and  $\Sigma_2$ -literals.

Let  $F_1$  be the conjunction of all literals of  $F$  that do not contain  $\Sigma_2$ -symbols; let  $F_2$  be the conjunction of all literals of  $F$  that do not contain  $\Sigma_1$ -symbols. (Equations between variables are in both  $F_1$  and  $F_2$ .)

The Nelson–Oppen algorithm starts with the pair  $F_1, F_2$  and applies the following inference rules.

# The Nelson–Oppen Algorithm (Non-determ.)

---

Unsat:

$$\frac{F_1, F_2}{\perp}$$

if  $\exists \vec{x} F_i$  is unsatisfiable w.r.t.  $\mathcal{T}_i$  for some  $i$ .

Branch:

$$\frac{F_1, F_2}{F_1 \wedge (x \approx y), F_2 \wedge (x \approx y) \quad | \quad F_1 \wedge (x \not\approx y), F_2 \wedge (x \not\approx y)}$$

if  $x$  and  $y$  are two different variables appearing in both  $F_1$  and  $F_2$  such that neither  $x \approx y$  nor  $x \not\approx y$  occurs in both  $F_1$  and  $F_2$

## The Nelson–Oppen Algorithm (Non-determ.)

---

“|” means non-deterministic (backtracking!) branching of the derivation into two subderivations. Derivations are therefore trees. All branches need to be reduced until termination.

Clearly, all derivation paths are finite since there are only finitely many **shared variables** in  $F_1$  and  $F_2$ , therefore the procedure represented by the rules is terminating.

We call a constraint configuration to which no rule applies **irreducible**.

## The Nelson–Oppen Algorithm (Non-determ.)

---

Theorem 2.2 (Soundness):

If “Branch” can be applied to  $F_1, F_2$ , then  $\exists \vec{x} (F_1 \wedge F_2)$  is satisfiable in  $\mathcal{T}_1 \cup \mathcal{T}_2$  if and only if one of the successor configurations of  $F_1, F_2$  is satisfiable in  $\mathcal{T}_1 \cup \mathcal{T}_2$ .

Corollary 2.3:

If all paths in a derivation tree from  $F_1, F_2$  end in  $\perp$ , then  $\exists \vec{x} (F_1 \wedge F_2)$  is unsatisfiable in  $\mathcal{T}_1 \cup \mathcal{T}_2$ .

## The Nelson–Oppen Algorithm (Non-determ.)

---

For completeness we need to show that if one branch in a derivation terminates with an irreducible configuration  $F_1, F_2$  (different from  $\perp$ ), then  $\exists \vec{x} (F_1 \wedge F_2)$  (and, thus, the initial formula of the derivation) is satisfiable in the combined theory.

As  $\exists \vec{x} (F_1 \wedge F_2)$  is irreducible by “Unsat”, the two formulas are satisfiable in their respective component theories, that is, we have  $\mathcal{T}_i$ -models  $\mathcal{A}_i$  of  $\exists \vec{x} F_i$  for  $i \in \{1, 2\}$ . We are left with combining the models into a single one that is both a model of the combined theory and of the combined formula. These constructions are called **amalgamations**.

## The Nelson–Oppen Algorithm (Non-determ.)

---

Let  $F$  be a  $\Sigma_i$ -formula and let  $S$  be a set of variables of  $F$ .

$F$  is called **compatible** with an equivalence  $\sim$  on  $S$  if the formula

$$\exists \vec{z} \left( F \wedge \bigwedge_{x,y \in S, x \sim y} x \approx y \wedge \bigwedge_{x,y \in S, x \not\sim y} x \not\approx y \right) \quad (1)$$

is  $\mathcal{T}_i$ -satisfiable whenever  $F$  is  $\mathcal{T}_i$ -satisfiable.

This expresses that  $F$  does not contradict equalities between the variables in  $S$  as given by  $\sim$ .

The formula  $\bigwedge_{x,y \in S, x \sim y} x \approx y \wedge \bigwedge_{x,y \in S, x \not\sim y} x \not\approx y$  is called an *arrangement* of  $S$ .

## The Nelson–Oppen Algorithm (Non-determ.)

---

Proposition 2.4:

If  $F_1, F_2$  is a pair of conjunctions over  $\mathcal{T}_1$  and  $\mathcal{T}_2$ , respectively, that is irreducible by “Branch”, then both  $F_1$  and  $F_2$  are compatible with some equivalence  $\sim$  on the shared variables  $S$  of  $F_1$  and  $F_2$ .

Proof:

If  $F_1, F_2$  is irreducible by the branching rule, then for each pair of shared variables  $x$  and  $y$ , both  $F_1$  and  $F_2$  contain either  $x \approx y$  or  $x \not\approx y$ .

Choose  $\sim$  to be the equivalence given by all (positive) variable equations between shared variables that are contained in  $F_1$ .

## The Nelson–Oppen Algorithm (Non-determ.)

---

Let  $\Sigma = (\Omega, \Pi)$ ; let  $\Sigma' = (\Omega', \Pi')$  with  $\Omega' \subseteq \Omega$  and  $\Pi' \subseteq \Pi$  be a subsignature of  $\Sigma$ .

Let  $\mathcal{A}$  be a  $\Sigma$ -algebra. Then the **reduct**  $\mathcal{A}|_{\Sigma'}$  is the  $\Sigma'$ -algebra  $\mathcal{A}'$  with

$$U_{\mathcal{A}'} = U_{\mathcal{A}},$$

$$f_{\mathcal{A}'} = f_{\mathcal{A}} \text{ for all } f \in \Omega', \text{ and}$$

$$P_{\mathcal{A}'} = P_{\mathcal{A}} \text{ for all } P \in \Pi'.$$

## The Nelson–Oppen Algorithm (Non-determ.)

---

Lemma 2.5 (Amalgamation Lemma):

Let  $\mathcal{T}_1$  and  $\mathcal{T}_2$  be two stably infinite theories over disjoint signatures  $\Sigma_1$  and  $\Sigma_2$ .

Furthermore let  $F_1, F_2$  be a pair of conjunctions of literals over  $\mathcal{T}_1$  and  $\mathcal{T}_2$ , respectively, both compatible with some equivalence  $\sim$  on the shared variables of  $F_1$  and  $F_2$ .

Then  $F_1 \wedge F_2$  is  $(\mathcal{T}_1 \cup \mathcal{T}_2)$ -satisfiable if and only if each  $F_i$  is  $\mathcal{T}_i$ -satisfiable.

## The Nelson–Oppen Algorithm (Non-determ.)

---

Theorem 2.6:

The non-deterministic Nelson–Oppen algorithm is terminating and complete for deciding satisfiability of pure conjunctions of literals  $F_1$  and  $F_2$  over  $\mathcal{T}_1 \cup \mathcal{T}_2$  for signature-disjoint, stably infinite theories  $\mathcal{T}_1$  and  $\mathcal{T}_2$ .

Proof:

Suppose that  $F_1, F_2$  is irreducible by the inference rules of the Nelson–Oppen algorithm. Applying the amalgamation lemma in combination with Prop. 2.4 we infer that  $F_1, F_2$  is satisfiable w.r.t.  $\mathcal{T}_1 \cup \mathcal{T}_2$ .

# Convexity

---

The number of possible equivalences of shared variables grows superexponentially with the number of shared variables, so enumerating all possible equivalences non-deterministically is going to be inefficient.

A much faster variant of the Nelson–Oppen algorithm exists for convex theories.

# Convexity

---

A first-order theory  $\mathcal{T}$  is called **convex w.r.t. equations**,  
if for every conjunction  $\Gamma$  of  $\Sigma$ -equations and non-equational  $\Sigma$ -literals and  
for all  $\Sigma$ -equations  $A_i$  ( $1 \leq i \leq n$ ),  
whenever  $\mathcal{T} \models \forall \vec{x} (\Gamma \rightarrow A_1 \vee \dots \vee A_n)$ , then there exists  
some index  $j$  such that  $\mathcal{T} \models \forall \vec{x} (\Gamma \rightarrow A_j)$ .

# Convexity

---

Theorem 2.7:

If a first-order theory  $\mathcal{T}$  is convex w.r.t. equations and has no trivial models (i.e., models with only one element), then  $\mathcal{T}$  is stably infinite.

# Convexity

---

Lemma 2.8:

Suppose  $\mathcal{T}$  is convex,  $F$  a conjunction of literals, and  $S$  a subset of its variables.

Let, for any pair of variables  $x_i$  and  $x_j$  in  $S$ ,  
 $x_i \sim x_j$  if and only if  $\mathcal{T} \models \forall \vec{x} (F \rightarrow x_i \approx x_j)$ .

Then  $F$  is compatible with  $\sim$ .

# The Nelson–Oppen Algorithm (Determin./Convex)

---

Unsat:

$$\frac{F_1, F_2}{\perp}$$

if  $\exists \vec{x} F_i$  is unsatisfiable w.r.t.  $\mathcal{T}_i$  for some  $i$ .

Propagate:

$$\frac{F_1, F_2}{F_1 \wedge (x \approx y), F_2 \wedge (x \approx y)}$$

if  $x$  and  $y$  are two different variables appearing in both  $F_1$  and  $F_2$  such that

$\mathcal{T}_1 \models \forall \vec{x} (F_1 \rightarrow x \approx y)$  and  $\mathcal{T}_2 \not\models \forall \vec{x} (F_2 \rightarrow x \approx y)$   
or  $\mathcal{T}_2 \models \forall \vec{x} (F_2 \rightarrow x \approx y)$  and  $\mathcal{T}_1 \not\models \forall \vec{x} (F_1 \rightarrow x \approx y)$ .

## The Nelson–Oppen Algorithm (Determin./Convex)

Theorem 2.9:

If  $\mathcal{T}_1$  and  $\mathcal{T}_2$  are signature-disjoint theories that are convex w.r.t. equations and have no trivial models,

then the deterministic Nelson–Oppen algorithm is terminating, sound and complete for deciding satisfiability of pure conjunctions of literals  $F_1$  and  $F_2$  over  $\mathcal{T}_1 \cup \mathcal{T}_2$ .

## The Nelson–Oppen Algorithm (Determin./Convex)

Corollary 2.10:

The deterministic Nelson–Oppen algorithm for convex theories requires at most  $O(n^3)$  calls to the individual decision procedures for the component theories, where  $n$  is the number of shared variables.

## Iterating Nelson–Oppen

---

The Nelson–Oppen combination procedures can be iterated to work with more than two component theories by virtue of the following observations where signature disjointness is assumed:

Theorem 2.11:

If  $\mathcal{T}_1$  and  $\mathcal{T}_2$  are stably infinite, then so is  $\mathcal{T}_1 \cup \mathcal{T}_2$ .

## Iterating Nelson–Oppen

---

Lemma 2.12:

A first-order theory  $\mathcal{T}$  is convex w.r.t. equations if and only if for every conjunction  $\Gamma$  of  $\Sigma$ -equations and non-equational  $\Sigma$ -literals and for all equations  $x_i \approx x'_i$  ( $1 \leq i \leq n$ ), whenever  $\mathcal{T} \models \forall \vec{x} (\Gamma \rightarrow x_1 \approx x'_1 \vee \dots \vee x_n \approx x'_n)$ , then there exists some index  $j$  such that  $\mathcal{T} \models \forall \vec{x} (\Gamma \rightarrow x_j \approx x'_j)$ .

## Iterating Nelson–Oppen

---

Lemma 2.13:

Let  $\mathcal{T}$  be a first-order theory that is convex w.r.t. equations. Let  $F$  is a conjunction of literals; let  $F^-$  be the conjunction of all negative equational literals in  $F$  and let  $F^+$  be the conjunction of all remaining literals in  $F$ . If  $\mathcal{T} \models \forall \vec{x} (F \rightarrow x \approx y)$ , then  $\exists \vec{x} F$  is  $\mathcal{T}$ -unsatisfiable or  $\mathcal{T} \models \forall \vec{x} (F^+ \rightarrow x \approx y)$ .

Theorem 2.14:

If  $\mathcal{T}_1$  and  $\mathcal{T}_2$  are convex w.r.t. equations and do not have trivial models, then so is  $\mathcal{T}_1 \cup \mathcal{T}_2$ .

# Extensions

---

Many-sorted logics:

*read/2* becomes  $read : array \times int \rightarrow data$ .

*write/3* becomes  $write : array \times int \times data \rightarrow array$ .

Variables:  $x : data$

Only one declaration per function/predicate/variable symbol.

All terms, atoms, substitutions must be well-sorted.

Algebras:

Instead of universe  $U_{\mathcal{A}}$ , one set per sort:  $array_{\mathcal{A}}$ ,  $int_{\mathcal{A}}$ .

Interpretations of function and predicate symbols correspond to their declarations:  $read_{\mathcal{A}} : array_{\mathcal{A}} \times int_{\mathcal{A}} \rightarrow data_{\mathcal{A}}$

# Extensions

---

If we consider combinations of theories with shared sorts but disjoint function and predicate symbols, then we get essentially the same combination results as before.

However, stable infiniteness and/or convexity are only required for the shared sorts.

# Extensions

---

Non-stably infinite theories:

If we impose stronger conditions on one theory, we can relax the conditions on the other one.

For instance, EUF can be combined with any other theory; stable infiniteness is not required.

E.g.: Strongly polite theories, shiny theories, flexible theories.

## Strong Politeness

---

A theory  $\mathcal{T}$  is called **smooth**, if every quantifier-free formula that has a  $\mathcal{T}$ -model with some (finite or infinite) cardinality  $\kappa_0$  has also  $\mathcal{T}$ -models with cardinality  $\kappa$  for every  $\kappa \geq \kappa_0$ .

# Strong Politeness

---

A theory  $\mathcal{T}$  is called **finitely witnessable**, if there is a computable function  $wit$  that maps every quantifier-free formula  $F$  to a quantifier-free formula  $G$  such that

1.  $F$  and  $\exists \vec{w} G$  are  $\mathcal{T}$ -equivalent, where  $\vec{w} = \text{var}(G) \setminus \text{var}(F)$ ,
2. if  $G \wedge \Delta$  is  $\mathcal{T}$ -satisfiable for some arrangement  $\Delta$ , then there is a  $\mathcal{T}$ -model  $\mathcal{A}$  and an assignment  $\beta$  such that  $\mathcal{A}, \beta \models (G \wedge \Delta)$  and  $U_{\mathcal{A}} = \{ \beta(x) \mid x \in \text{var}(G \wedge \Delta) \}$

# Strong Politeness

---

A theory  $\mathcal{T}$  is called **strongly polite**, if it is smooth and finitely witnessable.

Theorem 2.15 (Barrett & Jovanović):

We can combine two theories  $\mathcal{T}_1$  and  $\mathcal{T}_2$  if one of them is strongly polite.

Again, in the many-sorted case, smoothness and finite witnessability must hold for all the shared sorts.

# Strong Politeness

---

Non-disjoint combinations:

Have to ensure that both decision procedures interpret shared symbols in a compatible way.

Some results, e.g. by Ghilardi, using strong model theoretical conditions on the theories.

# Another Combination Method

---

Shostak's method:

Applicable to combinations of EUF and *solvable* theories.

## Another Combination Method

---

A  $\Sigma$ -theory  $\mathcal{T}$  is called **solvable**, if there exists an effectively computable function *solve* such that, for any  $\mathcal{T}$ -equation  $s \approx t$ :

(A)  $\text{solve}(s \approx t) = \perp$  if and only if  $\mathcal{T} \models \forall \vec{x} (s \not\approx t)$ ;

(B)  $\text{solve}(s \approx t) = \emptyset$  if and only if  $\mathcal{T} \models \forall \vec{x} (s \approx t)$ ;  
and otherwise

(C)  $\text{solve}(s \approx t) = \{x_1 \approx u_1, \dots, x_n \approx u_n\}$ , where

– the  $x_i$  are pairwise different variables occurring in  $s \approx t$ ;

– the  $x_i$  do not occur in the  $u_j$ ; and

–  $\mathcal{T} \models \forall \vec{x} ((s \approx t) \leftrightarrow \exists \vec{y} (x_1 \approx u_1 \wedge \dots \wedge x_n \approx u_n))$ , where  $\vec{y}$  are the variables occurring in one of the  $u_j$  but not in  $s \approx t$ , and  $\vec{x} \cap \vec{y} = \emptyset$ .

## Another Combination Method

---

Additionally useful (but not required):

A canonizer, that is, a function that simplifies terms by computing some unique normal form

# Another Combination Method

---

Main idea of the procedure:

If  $s \approx t$  is a positive equation and

$$\text{solve}(s \approx t) = \{x_1 \approx u_1, \dots, x_n \approx u_n\},$$

replace  $s \approx t$  by  $x_1 \approx u_1 \wedge \dots \wedge x_n \approx u_n$

and use these equations to eliminate the  $x_i$  elsewhere.

Practical problem:

Solvability is a rather restrictive condition.