Automated Reasoning*

Christoph Weidenbach

Summer Term 2008

Topics of the Course I

Propositional logic

> language: syntax, semantics
> calculi: DPLL-procedure
> implementation: 2-watched literal, clause learning

Linear arithmetic

> language: syntax, semantics
> calculi: Fourier-Motzkin

Propositional logic modulo a theory T

> syntax, semantics
> calculi: DPLL(T)-procedure, ...
> implementation: coupling

*This document contains the text of the lecture slides (almost verbatim) plus some additional information, mostly proofs of theorems that are presented on the blackboard during the course. It is not a full script and does not contain the examples and additional explanations given during the lecture. Moreover it should not be taken as an example how to write a research paper – neither stylistically nor typographically.

Topics of the Course II

First-order predicate logic with equality

syntax, semantics, model theory, ...
calculi: superposition (SUP)
implementation: sharing, indexing

First-order predicate logic with equality modulo a theory T

syntax, semantics, model theory, ...
calculi: SUP(T)
implementation: coupling

1 Propositional Logic

Propositional logic

- logic of truth values
- decidable (but NP-complete)
- can be used to describe functions over a finite domain
- important for hardware applications (e. g., model checking)

1.1 Syntax

- propositional variables
- logical symbols
  $\Rightarrow$ Boolean combinations

Propositional Variables

Let Π be a set of *propositional variables*.

We use letters P, Q, R, S, to denote propositional variables.

Propositional Formulas

F_Π is the set of propositional formulas over Π defined as follows:

$$
\begin{array}{lcll}
F, G, H & ::= & \bot & \text{(falsum)} \\
& \mid & \top & \text{(verum)} \\
& \mid & P, \quad P \in \Pi & \text{(atomic formula)} \\
& \mid & \neg F & \text{(negation)} \\
& \mid & (F \wedge G) & \text{(conjunction)} \\
& \mid & (F \vee G) & \text{(disjunction)} \\
& \mid & (F \rightarrow G) & \text{(implication)} \\
& \mid & (F \leftrightarrow G) & \text{(equivalence)}
\end{array}
$$

Notational Conventions

- We omit brackets according to the following rules:
  - $\neg \quad >_p \quad \vee \quad >_p \quad \wedge \quad >_p \quad \rightarrow \quad >_p \quad \leftrightarrow$ (binding precedences)
  - $\vee$ and $\wedge$ are associative
  - $\rightarrow$ is right-associative,
    i. e., $F \rightarrow G \rightarrow H$ means $F \rightarrow (G \rightarrow H)$.

1.2 Semantics

In *classical logic* (dating back to Aristoteles) there are "only" two truth values "true" and "false" which we shall denote, respectively, by 1 and 0.

There are *multi-valued logics* having more than two truth values.

Valuations

A propositional variable has no intrinsic meaning. The meaning of a propositional variable has to be defined by a valuation.

A Π-*valuation* is a map

$$\mathcal{A} : \Pi \rightarrow \{0, 1\}.$$

where $\{0, 1\}$ is the set of *truth values*.

Truth Value of a Formula in $\mathcal{A}$

Given a Π-valuation $\mathcal{A}$, the function $\mathcal{A}^* : \Sigma\text{-formulas} \to \{0, 1\}$ is defined inductively over the structure of F as follows:

$$\begin{aligned}
\mathcal{A}^*(\bot) &= 0 \\
\mathcal{A}^*(\top) &= 1 \\
\mathcal{A}^*(P) &= \mathcal{A}(P) \\
\mathcal{A}^*(\neg F) &= \mathsf{B}_\neg(\mathcal{A}^*(F)) \\
\mathcal{A}^*(F \rho G) &= \mathsf{B}_\rho(\mathcal{A}^*(F), \mathcal{A}^*(G))
\end{aligned}$$

where B_ρ is the Boolean function associated with ρ defined by the usual truth table.

For simplicity, we write $\mathcal{A}$ instead of $\mathcal{A}^*$.

We also write ρ instead of B_ρ, i. e., we use the same notation for a logical symbol and for its meaning (but remember that formally these are different things.)

1.3 Models, Validity, and Satisfiability

F is *valid* in $\mathcal{A}$ ($\mathcal{A}$ is a *model* of F; F holds under $\mathcal{A}$):

$$\mathcal{A} \models F \;:\Leftrightarrow\; \mathcal{A}(F) = 1$$

F is *valid* (or is a *tautology*):

$$\models F \;:\Leftrightarrow\; \mathcal{A} \models F \text{ for all Π-valuations } \mathcal{A}$$

F is called *satisfiable* if there exists an $\mathcal{A}$ such that $\mathcal{A} \models F$. Otherwise F is called *unsatisfiable* (or *contradictory*).

Entailment and Equivalence

F *entails* (*implies*) G (or G *is a consequence of* F), written $F \models G$, if for all Π-valuations $\mathcal{A}$, whenever $\mathcal{A} \models F$ then $\mathcal{A} \models G$.

F and G are called *equivalent*, written $F \models\!\!\!\mid G$, if for all Π-valuations $\mathcal{A}$ we have $\mathcal{A} \models F \Leftrightarrow \mathcal{A} \models G$.

Proposition 1.1 *$F \models G$ if and only if $\models (F \rightarrow G)$.(Proof follows)*

Proof. ($\Rightarrow$) Suppose that F entails G. Let $\mathcal{A}$ be an arbitrary Π-valuation. We have to show that $\mathcal{A} \models F \rightarrow G$. If $\mathcal{A}(F) = 1$, then $\mathcal{A}(G) = 1$ (since $F \models G$), and hence $\mathcal{A}(F \rightarrow G) = 1$. Otherwise $\mathcal{A}(F) = 0$, then $\mathcal{A}(F \rightarrow G) = \mathsf{B}_{\rightarrow}(0, \mathcal{A}(G)) = 1$ independently of $\mathcal{A}(G)$. In both cases, $\mathcal{A} \models F \rightarrow G$.

($\Leftarrow$) Suppose that F does not entail G. Then there exists a Π-valuation $\mathcal{A}$ such that $\mathcal{A} \models F$, but not $\mathcal{A} \models G$. Consequently, $\mathcal{A}(F \rightarrow G) = \mathsf{B}_{\rightarrow}(\mathcal{A}(F), \mathcal{A}(G)) = \mathsf{B}_{\rightarrow}(1, 0) = 0$, so $(F \rightarrow G)$ does not hold in $\mathcal{A}$. $\square$

Proposition 1.2 *$F \models\!\mid G$ if and only if $\models (F \leftrightarrow G)$.*

Proof. Follows from Prop. 1.1. $\square$

Extension to sets of formulas N in the "natural way":

$N \models F$ if for all Π-valuations $\mathcal{A}$:
if $\mathcal{A} \models G$ for all $G \in N$, then $\mathcal{A} \models F$.

Validity vs. Unsatisfiability

Validity and unsatisfiability are just two sides of the same medal as explained by the following proposition.

Proposition 1.3 *F is valid if and only if $\neg F$ is unsatisfiable.(Proof follows)*

Proof. ($\Rightarrow$) If F is valid, then $\mathcal{A}(F) = 1$ for every valuation $\mathcal{A}$. Hence $\mathcal{A}(\neg F) = \mathsf{B}_{\neg}(\mathcal{A}(F)) = \mathsf{B}_{\neg}(1) = 0$ for every valuation $\mathcal{A}$, so $\neg F$ is unsatisfiable.

($\Leftarrow$) Analogously. $\square$

Hence in order to design a theorem prover (validity checker) it is sufficient to design a checker for unsatisfiability.

In a similar way, entailment $N \models F$ can be reduced to unsatisfiability:

Proposition 1.4 *$N \models F$ if and only if $N \cup \{\neg F\}$ is unsatisfiable.*

Checking Unsatisfiability

Every formula F contains only finitely many propositional variables. Obviously, $\mathcal{A}(F)$ depends only on the values of those finitely many variables in F under $\mathcal{A}$.

If F contains n distinct propositional variables, then it is sufficient to check 2^n valuations to see whether F is satisfiable or not.
$\Rightarrow$ truth table.

So the satisfiability problem is clearly deciadable (but, by Cook's Theorem, NP-complete).

Nevertheless, in practice, there are (much) better methods than truth tables to check the satisfiability of a formula. (later more)

Substitution Theorem

Proposition 1.5 *Let F and G be equivalent formulas, let H be a formula in which F occurs as a subformula.*

Then H is equivalent to H' where H' is obtained from H by replacing the occurrence of the subformula F by G. (Notation: $H = H[F]$, $H' = H[G]$. Proof follows)

Proof. The proof proceeds by induction over the formula structure of H.

Each of the formulas $\bot$, $\top$, and P for $P \in \Pi$ contains only one subformula, namely itself. Hence, if $H = H[F]$ equals $\bot$, $\top$, or P, then $H = F$, $H' = G$, and H and H' are equivalent by assumption.

If $H = H_1 \wedge H_2$, then either F equals H (this case is treated as above), or F is a subformula of H_1 or H_2. Without loss of generality, assume that F is a subformula of H_1, so $H = H_1[F] \wedge H_2$. By the induction hypothesis, $H_1[F]$ and $H_1[G]$ are equivalent. Hence, for every valuation $\mathcal{A}$, $\mathcal{A}(H') = \mathcal{A}(H_1[G] \wedge H_2) = \mathcal{A}(H_1[G]) \wedge \mathcal{A}(H_2) = \mathcal{A}(H_1[F]) \wedge \mathcal{A}(H_2) = \mathcal{A}(H_1[F] \wedge H_2) = \mathcal{A}(H)$.

The other boolean connectives are handled analogously. $\square$

Some Important Equivalences

Proposition 1.6 *The following equivalences are valid for all formulas F, G, H:*

$$
\begin{aligned}
(F \wedge F) &\leftrightarrow F \\
(F \vee F) &\leftrightarrow F && \textit{(Idempotency)} \\
(F \wedge G) &\leftrightarrow (G \wedge F) \\
(F \vee G) &\leftrightarrow (G \vee F) && \textit{(Commutativity)} \\
(F \wedge (G \wedge H)) &\leftrightarrow ((F \wedge G) \wedge H) \\
(F \vee (G \vee H)) &\leftrightarrow ((F \vee G) \vee H) && \textit{(Associativity)} \\
(F \wedge (G \vee H)) &\leftrightarrow ((F \wedge G) \vee (F \wedge H)) \\
(F \vee (G \wedge H)) &\leftrightarrow ((F \vee G) \wedge (F \vee H)) && \textit{(Distributivity)}
\end{aligned}
$$

$$
\begin{aligned}
(F \wedge (F \vee G)) &\leftrightarrow F \\
(F \vee (F \wedge G)) &\leftrightarrow F && \textit{(Absorption)} \\
(\neg\neg F) &\leftrightarrow F && \textit{(Double Negation)} \\
\neg(F \wedge G) &\leftrightarrow (\neg F \vee \neg G) \\
\neg(F \vee G) &\leftrightarrow (\neg F \wedge \neg G) && \textit{(De Morgan's Laws)} \\
(F \wedge G) &\leftrightarrow F, \textit{ if } G \textit{ is a tautology} \\
(F \vee G) &\leftrightarrow \top, \textit{ if } G \textit{ is a tautology} \\
(F \wedge G) &\leftrightarrow \bot, \textit{ if } G \textit{ is unsatisfiable} \\
(F \vee G) &\leftrightarrow F, \textit{ if } G \textit{ is unsatisfiable} && \textit{(Tautology Laws)}
\end{aligned}
$$

$$
\begin{aligned}
(F \leftrightarrow G) &\leftrightarrow ((F \rightarrow G) \wedge (G \rightarrow F)) && \textit{(Equivalence)} \\
(F \rightarrow G) &\leftrightarrow (\neg F \vee G) && \textit{(Implication)}
\end{aligned}
$$

1.4 Normal Forms

We define *conjunctions* of formulas as follows:

$\bigwedge_{i=1}^{0} F_i = \top.$

$\bigwedge_{i=1}^{1} F_i = F_1.$

$\bigwedge_{i=1}^{n+1} F_i = \bigwedge_{i=1}^{n} F_i \wedge F_{n+1}.$

and analogously *disjunctions:*

$\bigvee_{i=1}^{0} F_i = \bot.$

$\bigvee_{i=1}^{1} F_i = F_1.$

$\bigvee_{i=1}^{n+1} F_i = \bigvee_{i=1}^{n} F_i \vee F_{n+1}.$

Literals and Clauses

A *literal* is either a propositional variable P or a negated propositional variable $\neg P$.

A *clause* is a (possibly empty) disjunction of literals.

CNF and DNF

A formula is in *conjunctive normal form (CNF, clause normal form),* if it is a conjunction of disjunctions of literals (or in other words, a conjunction of clauses).

A formula is in *disjunctive normal form (DNF),* if it is a disjunction of conjunctions of literals.

Warning: definitions in the literature differ:

are complementary literals permitted?
are duplicated literals permitted?
are empty disjunctions/conjunctions permitted?

Checking the validity of CNF formulas or the unsatisfiability of DNF formulas is easy:

A formula in CNF is valid, if and only if each of its disjunctions contains a pair of complementary literals P and $\neg P$.

Conversely, a formula in DNF is unsatisfiable, if and only if each of its conjunctions contains a pair of complementary literals P and $\neg P$.

On the other hand, checking the unsatisfiability of CNF formulas or the validity of DNF formulas is known to be coNP-complete.

Conversion to CNF/DNF

Proposition 1.7 *For every formula there is an equivalent formula in CNF (and also an equivalent formula in DNF).*

Proof. We consider the case of CNF.

Apply the following rules as long as possible (modulo associativity and commutativity of $\wedge$ and $\vee$):

Step 1: Eliminate equivalences:

$$(F \leftrightarrow G) \;\Rightarrow_K\; (F \rightarrow G) \wedge (G \rightarrow F)$$

Step 2: Eliminate implications:

$$(F \rightarrow G) \;\Rightarrow_K\; (\neg F \vee G)$$

Step 3: Push negations downward:

$$\begin{aligned} \neg(F \vee G) \;&\Rightarrow_K\; (\neg F \wedge \neg G) \\ \neg(F \wedge G) \;&\Rightarrow_K\; (\neg F \vee \neg G) \end{aligned}$$

Step 4: Eliminate multiple negations:

$$\neg\neg F \;\Rightarrow_K\; F$$

Step 5: Push disjunctions downward:

$$(F \wedge G) \vee H \;\Rightarrow_K\; (F \vee H) \wedge (G \vee H)$$

Step 6: Eliminate $\top$ and $\bot$:

$$\begin{aligned} (F \wedge \top) \;&\Rightarrow_K\; F \\ (F \wedge \bot) \;&\Rightarrow_K\; \bot \\ (F \vee \top) \;&\Rightarrow_K\; \top \\ (F \vee \bot) \;&\Rightarrow_K\; F \\ \neg\bot \;&\Rightarrow_K\; \top \\ \neg\top \;&\Rightarrow_K\; \bot \end{aligned}$$

Proving termination is easy for most of the steps; only step 3 and step 5 are a bit more complicated.

For step 3, we can prove termination in the following way: We define a function μ from formulas to positive integers such that $\mu(\bot) = \mu(\top) = \mu(P) = 1$, $\mu(\neg F) = 2\mu(F)$, $\mu(F \wedge G) = \mu(F \vee G) = \mu(F \rightarrow G) = \mu(F \leftrightarrow G) = \mu(F) + \mu(G) + 1$. Whenever a formula H' is the result of applying a rule of step 3 to a formula H, then $\mu(H) > \mu(H')$. Since μ takes only integer values and $\mu(H) \geq 1$ for all formulas H, step 3 must terminate.

Termination of step 5 is proved similarly using a function ν from formulas to positive integers such that $\nu(\bot) = \nu(\top) = \nu(P) = 1$, $\nu(\neg F) = \nu(F) + 1$, $\nu(F \wedge G) = \nu(F \rightarrow G) = \nu(F \leftrightarrow G) = \nu(F) + \nu(G) + 1$, and $\nu(F \vee G) = 2\nu(F)\nu(G)$. Again, if a formula H' is the result of applying a rule of step 5 to a formula H, then $\nu(H) > \nu(H')$. Since ν takes only integer values and Since $\nu(H) \geq 1$ for all formulas H, step 5 terminates, too.

The resulting formula is equivalent to the original one and in CNF.

The conversion of a formula to DNF works in the same way, except that disjunctions have to be pushed downward in step 5. $\square$

Complexity

Conversion to CNF (or DNF) may produce a formula whose size is *exponential* in the size of the original one.

Satisfiability-preserving Transformations

The goal

“find a formula G in CNF such that $F \models\!\!\!\!\mid G$”

is unpractical.

But if we relax the requirement to

“find a formula G in CNF such that $F \models \bot \Leftrightarrow G \models \bot$”

we can get an efficient transformation.

Idea: A formula $F[F']$ is satisfiable if and only if $F[P] \wedge (P \leftrightarrow F')$ is satisfiable (where P is a new propositional variable that works as an abbreviation for F').

We can use this rule recursively for all subformulas in the original formula (this introduces a linear number of new propositional variables).

Conversion of the resulting formula to CNF increases the size only by an additional factor (each formula $P \leftrightarrow F'$ gives rise to at most one application of the distributivity law).

Optimized Transformations

A further improvement is possible by taking the *polarity* of the subformula F into account.

Assume that F contains neither $\rightarrow$ nor $\leftrightarrow$. A subformula F' of F has *positive polarity* in F, if it occurs below an even number of negation signs; it has *negative polarity* in F, if it occurs below an odd number of negation signs.

Proposition 1.8 *Let $F[F']$ be a formula containing neither $\rightarrow$ nor $\leftrightarrow$; let P be a propositional variable not occurring in $F[F']$.*

If F' has positive polarity in F, then $F[F']$ is satisfiable if and only if $F[P] \wedge (P \rightarrow F')$ is satisfiable.

If F' has negative polarity in F, then $F[F']$ is satisfiable if and only if $F[P] \wedge (F' \rightarrow P)$ is satisfiable.

Proof. Exercise. $\square$

1.5 The DPLL Procedure

Goal:
Given a propositional formula in CNF (or alternatively, a finite set N of clauses), check whether it is satisfiable (and optionally: output *one* solution, if it is satisfiable).

Assumption:
Clauses contain neither duplicated literals nor complementary literals.

Notation:
$\overline{L}$ is the complementary literal of L, i. e., $\overline{P} = \neg P$ and $\overline{\neg P} = P$.

Satisfiability of Clause Sets

$\mathcal{A} \models N$ if and only if $\mathcal{A} \models C$ for all clauses C in N.

$\mathcal{A} \models C$ if and only if $\mathcal{A} \models L$ for some literal $L \in C$.

Partial Valuations

Since we will construct satisfying valuations incrementally, we consider *partial valuations* (that is, partial mappings $\mathcal{A} : \Pi \to \{0, 1\}$).

Every partial valuation $\mathcal{A}$ corresponds to a set M of literals that does not contain complementary literals, and vice versa:

$\mathcal{A}(L)$ is true, if $L \in M$.

$\mathcal{A}(L)$ is false, if $\overline{L} \in M$.

$\mathcal{A}(L)$ is undefined, if neither $L \in M$ nor $\overline{L} \in M$.

We will use $\mathcal{A}$ and M interchangeably.

A clause is true under a partial valuation $\mathcal{A}$ (or under a set M of literals) if one of its literals is true; it is false (or "*conflicting*") if all its literals are false; otherwise it is undefined (or "*unresolved*").

Unit Clauses

Observation:
Let $\mathcal{A}$ be a partial valuation. If the set N contains a clause C, such that all literals but one in C are false under $\mathcal{A}$, then the following properties are equivalent:

- there is a valuation that is a model of N and extends $\mathcal{A}$.
- there is a valuation that is a model of N and extends $\mathcal{A}$ and makes the remaining literal L of C true.

C is called a *unit clause;* L is called a *unit literal.*

Pure Literals

One more observation:
Let $\mathcal{A}$ be a partial valuation and P a variable that is undefined under $\mathcal{A}$. If P occurs only positively (or only negatively) in the unresolved clauses in N, then the following properties are equivalent:

- there is a valuation that is a model of N and extends $\mathcal{A}$.
- there is a valuation that is a model of N and extends $\mathcal{A}$ and assigns true (false) to P.

P is called a *pure literal.*

The Davis-Putnam-Logemann-Loveland Proc.

```
boolean DPLL(literal set M, clause set N) {
    if (all clauses in N are true under M) return true;
    elsif (some clause in N is false under M) return false;
    elsif (N contains unit clause P) return DPLL(M ∪ {P}, N);
    elsif (N contains unit clause ¬P) return DPLL(M ∪ {¬P}, N);
    elsif (N contains pure literal P) return DPLL(M ∪ {P}, N);
    elsif (N contains pure literal ¬P) return DPLL(M ∪ {¬P}, N);
    else {
        let P be some undefined variable in N;
        if (DPLL(M ∪ {¬P}, N)) return true;
        else return DPLL(M ∪ {P}, N);
    }
}
```

Initially, DPLL is called with an empty literal set and the clause set N.

DPLL Iteratively

In practice, there are several changes to the procedure:

The pure literal check is often omitted (it is too expensive).

The branching variable is not chosen randomly.

The algorithm is implemented iteratively;
the backtrack stack is managed explicitly
(it may be possible and useful to backtrack more than one level).

Information is reused by learning.

Branching Heuristics

Choosing the right undefined variable to branch is important for efficiency, but the branching heuristics may be expensive itself.

State of the art: use branching heuristics that need not be recomputed too frequently.

In general: choose variables that occur frequently.

The Deduction Algorithm

For applying the unit rule, we need to know the number of literals in a clause that are not false.

Maintaining this number is expensive, however.

Better approach: *"Two watched literals"*:

In each clause, select two (currently undefined) "watched" literals.

For each variable P, keep a list of all clauses in which P is watched and a list of all clauses in which $\neg P$ is watched.

If an undefined variable is set to 0 (or to 1), check all clauses in which P (or $\neg P$) is watched and watch another literal (that is true or undefined) in this clause if possible.

Watched literal information need not be restored upon backtracking.

Conflict Analysis and Learning

Goal: Reuse information that is obtained in one branch in further branches.

Method: *Learning:*

> If a conflicting clause is found, derive a new clause from the conflict and add it to the current set of clauses.

Problem: This may produce a large number of new clauses; therefore it may become necessary to delete some of them afterwards to save space.

Backjumping

Related technique:
non-chronological backtracking ("backjumping"):

> If a conflict is independent of some earlier branch, try to skip over that backtrack level.

Restart

Runtimes of DPLL-style procedures depend extremely on the choice of branching variables.

If no solution is found within a certain time limit, it can be useful to *restart* from scratch with another choice of branchings (but learned clauses may be kept).

Formalizing DPLL with Refinements

The DPLL procedure is modelled by a transition relation $\Rightarrow_{\mathrm{DPLL}}$ on a set of states.

States:

- *fail*
- $M \parallel N$,

where M is a *list of annotated literals* and N is a set of clauses.

Annotated literal:

- L: deduced literal, due to unit propagation.
- L^{d}: decision literal (guessed literal).

Unit Propagate:

$M \parallel N \cup \{C \vee L\} \;\Rightarrow_{\text{DPLL}}\; M\, L \parallel N \cup \{C \vee L\}$

if C is false under M and L is undefined under M.

Decide:

$M \parallel N \;\Rightarrow_{\text{DPLL}}\; M\, L^{\text{d}} \parallel N$

if L is undefined under M.

Fail:

$M \parallel N \cup \{C\} \;\Rightarrow_{\text{DPLL}}\; \textit{fail}$

if C is false under M and M contains no decision literals.

Backjump:

$M'\, L^{\text{d}}\, M'' \parallel N \;\Rightarrow_{\text{DPLL}}\; M'\, L' \parallel N$

if there is some "backjump clause" $C \vee L'$ such that
$N \models C \vee L'$,
C is false under M', and
L' is undefined under M'.

We will see later that the Backjump rule is always applicable, if the list of literals M contains at least one decision literal and some clause in N is false under M.

There are many possible backjump clauses. One candidate: $\overline{L_1} \vee \ldots \vee \overline{L_n}$, where the L_i are all the decision literals in $M\, L^{\text{d}}\, M'$. (But usually there are better choices.)

Lemma 1.9 *If we reach a state $M \parallel N$ starting from $\emptyset \parallel N$, then:*

(1) M does not contain complementary literals.

(2) Every deduced literal L in M follows from N and decision literals occurring before L in M.

Proof. By induction on the length of the derivation. □

Lemma 1.10 *Every derivation starting from $\emptyset \parallel N$ terminates.*

[The proof is relatively easy but requires techniques that will be introduced later in the lecture.]

Lemma 1.11 *Suppose that we reach a state $M \parallel N$ starting from $\emptyset \parallel N$ such that some clause $D \in N$ is false under M. Then:*

(1) If M does not contain any decision literal, then "Fail" is applicable.

(2) Otherwise, "Backjump" is applicable.

(Proof follows)

Proof. (1) Obvious.

(2) Let $L_1, \dots, L_n$ be the decision literals occurring in M (in this order). Since $M \models \neg D$, we obtain, by Lemma 1.9, $N \cup \{L_1, \dots, L_n\} \models \neg D$. Since $D \in N$, $N \models \overline{L_1} \vee \cdots \vee \overline{L_n}$. Now let $C = \overline{L_1} \vee \cdots \vee \overline{L_{n-1}}$, $L' = \overline{L_n}$, $L = L_n$, and let M' be the list of all literals of M occurring before L_n, then the condition of "Backjump" is satisfied. □

Theorem 1.12 *(1) If we reach a final state $M \parallel N$ starting from $\emptyset \parallel N$, then N is satisfiable and M is a model of N.*

(2) If we reach a final state fail starting from $\emptyset \parallel N$, then N is unsatisfiable.

(Proof follows)

Proof. (1) Observe that the "Decide" rule is applicable as long as literals are undefined under M. Hence, in a final state, all literals must be defined. Furthermore, in a final state, no clause in N can be false under M, otherwise "Fail" or "Backjump" would be applicable. Hence M is a model of every clause in N.

(2) If we reach *fail*, then in the previous step we must have reached a state $M \parallel N$ such that some $C \in N$ is false under M and M contains no decision literals. By part (2) of Lemma 1.9, every literal in M follows from N. On the other hand, $C \in N$, so N must be unsatisfiable. □

Getting Better Backjump Clauses

Suppose that we have reached a state $M \parallel N$ such that some clause $C \in N$ (or following from N) is false under M.

Consequently, every literal of C is the complement of some literal in M.

(1) If every literal in C is the complement of a decision literal of M. Then C is a backjump clause.

(2) Otherwise, $C = C' \vee \overline{L}$, such that L is a deduced literal.

    For every deduced literal L, there is a clause $D \vee L$, such that $N \models D \vee L$ and D is false under M.

    Consequently, $N \models D \vee C'$ and $D \vee C'$ is also false under M.

By repeating this process, we will eventually obtain a clause that consists only of complements of decision literals and can be used in the “Backjump” rule.

Moreover, such a clause is a good candidate for learning.

Learning Clauses

The DPLL system can be extended by two rules to learn and to forget clauses:

Learn:

$M \parallel N \;\Rightarrow_{\mathrm{DPLL}}\; M \parallel N \cup \{C\}$

if $N \models C$.

Forget:

$M \parallel N \cup \{C\} \;\Rightarrow_{\mathrm{DPLL}}\; M \parallel N$

if $N \models C$.

If we ensure that no clause is learned infinitely often, then termination is guaranteed.

The other properties of the basic DPLL system hold also for the extended system.

Further Information

The ideas described so far heve been implemented in the SAT checker *zChaff*.

Further information:
Lintao Zhang and Sharad Malik: The Quest for Efficient Boolean Satisfiability Solvers, Proc. CADE-18, LNAI 2392, pp. 295–312, Springer, 2002.

Robert Nieuwenhuis, Albert Oliveras, Cesare Tinelli: Solvin SAT and SAT Modulo Theories: From an abstract Davis-Putnam-Logemann-Loveland precedure to DPLL(T), pp 937–977, Journal of the ACM, 53(6), 2006.

1.6 Splitting into Horn Clauses (Extra Topic)

- A *Horn clause* is a clause with at most one positive literal.
- They are typically denoted as implications: $P_1, \ldots, P_n \rightarrow Q$.
  (In general we can write $P_1, \ldots, P_n \rightarrow Q_1, \ldots, Q_m$ for $\neg P_1 \vee \ldots \vee \neg P_n \vee Q_1 \vee \ldots \vee Q_m$.)
- Compared to arbitrary clause sets, Horn clause sets enjoy further properties:
  - Horn clause sets have unique minimal models.
  - Checking satisfiability is often of lower complexity.

Propositional Horn Clause SAT is in $\mathbf{P}$

boolean HornSAT(literal set M, Horn clause set N) {
  if (all clauses in N are supported by M) return true;
  elsif (a negative clause in N is not supported by M) return false;
  elsif (N contains clause $P_1, \ldots, P_n \rightarrow Q$ where
      $\{P_1, \ldots, P_n\} \subseteq M$ and $Q \notin M$)
    return HornSAT($M \cup \{Q\}$, N);
}

A clause $P_1, \ldots, P_n \rightarrow Q_1, \ldots, Q_m$ is *supported* by M if $\{P_1, \ldots, P_n\} \not\subseteq M$ or some $Q_i \in M$. A *negative* clause consists of negative literals only.

Initially, HornSAT is called with an empty literal set M.

Lemma 1.13 *Let N be a set of propositional Horn clauses.Then:*

(1) HornSAT($\emptyset$, N)=true iff N is satisfiable

(2) HornSAT is in $\mathbf{P}$

Proof. (1) (Idea) For example, by induction on the number of positive literals in N.

(2) (Scetch) For each recursive call M contains one more positive literal. Thus HornSAT terminates after at most n recursive calls, where n is the number of propositional variables in N. $\square$

SplitHornSAT

boolean SplitHornSAT(clause set N) {
  if (N is Horn)
g   return HornSAT($\emptyset$,N);
  else {
    select non Horn clause $P_1, \ldots, P_n \rightarrow Q_1, \ldots, Q_m$ from N;
    $N' = N \setminus \{P_1, \ldots, P_n \rightarrow Q_1, \ldots, Q_m\}$;
    if (SplitHornSAT($N' \cup \{P_1, \ldots, P_n \rightarrow Q_1\}$)) return true;
    else return
     SplitHornSAT($N' \cup \{\rightarrow Q_2, \ldots, Q_m\} \cup \bigcup_i \{\rightarrow P_i\} \cup \{Q_1 \rightarrow\}$);
  }
}

Lemma 1.14 *Let N be a set of propositional clauses. Then:*

(1) SplitHornSAT(N)=true iff N is satisfiable

(2) SplitHornSAT(N) terminates

Proof. (1) (Idea) Show that N is satisfiable iff $N' \cup \{P_1, \ldots, P_n \rightarrow Q_1\}$ is satisfiable or $N' \cup \{\rightarrow Q_2, \ldots, Q_m\} \cup \bigcup_i \{\rightarrow P_i\} \cup \{Q_1 \rightarrow\}$ is satisfiable for some clause $P_1, \ldots, P_n \rightarrow Q_1, \ldots, Q_m$ from N.

(2) (Idea) Each recursive call reduces the number of positive literals in non Horn clauses. □

1.7 Other Calculi

OBDDs (Ordered Binary Decision Diagrams):

Minimized graph representation of decision trees, based on a fixed ordering on propositional variables,

see script of the Computational Logic course,

see Chapter 6.1/6.2 of Michael Huth and Mark Ryan: *Logic in Computer Science: Modelling and Reasoning about Systems*, Cambridge Univ. Press, 2000.

FRAIGs (Fully Reduced And-Inverter Graphs)

Minimized graph representation of boolean circuits.

1.8 Example: SUDOKU

	1	2	3	4	5	6	7	8	9
1								1	
2	4								
3		2							
4					5		4		7
5			8				3		
6			1		9				
7	3			4			2		
8		5		1					
9				8		6			

Idea: $p^d_{i,j}$=true iff the value of square i, j is d

For example: $p^8_{3,5} = true$

Coding SUDOKU by propositional clauses

- Concrete values result in units: $p^d_{i,j}$
- For every value, column we generate: $\neg p^d_{i,j} \vee \neg p^d_{i,j+k}$
  Accordingly for all rows and 3×3 boxes
- For every square we generate: $p^1_{i,j} \vee \ldots \vee p^9_{i,j}$
- For every two different values, square we generate: $\neg p^d_{i,j} \vee \neg p^{d'}_{i,j}$
- For every value, column we generate: $p^d_{i,0} \vee \ldots \vee p^d_{i,9}$
  Accordingly for all rows and 3×3 boxes

Constraint Propagation is Unit Propagation

	1	2	3	4	5	6	7	8	9
1								1	
2	4								
3		2							
4					5		4		7
5			8				3		
6			1		9				
7	3			4	**7**		2		
8		5		1					
9				8		6			

From $\neg p^3_{1,7} \vee \neg p^3_{5,7}$ and $p^3_{1,7}$ we obtain by unit propagating $\neg p^3_{5,7}$ and further from $p^1_{5,7} \vee p^2_{5,7} \vee p^3_{5,7} \vee p^4_{5,7} \vee \ldots \vee p^9_{5,7}$ we get $p^1_{5,7} \vee p^2_{5,7} \vee p^4_{5,7} \vee \ldots \vee p^9_{5,7}$.

2 Linear Arithmetic (LA)

We consider boolean combinations of linear arithmetic atoms such as $3.5x - 4y \geq 7$ and search rational values for the variables x, y such that the disequation holds.

2.1 Syntax

Syntax:

- non-logical symbols (domain-specific) (e.g. $x, +$, values from $\mathbb{Q}, \geq$)
  $\Rightarrow$ terms, atomic formulas
- logical symbols (domain-independent) (e.g. $\wedge, \rightarrow$)
  $\Rightarrow$ Boolean combinations (no quantification)

Signature

A signature

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

fixes an alphabet of non-logical symbols, where

- Ω is a set of *function symbols* f with *arity* $n \geq 0$, written $\mathsf{arity}(f) = n$,
- Π is a set of *predicate symbols* p with arity $m \geq 0$, written $\mathsf{arity}(p) = m$.

The linear arithmetic signature is

$$\Sigma_{\mathrm{LA}} = (\mathbb{Q} \cup \{+, -, *\}, \{\geq, \leq, >, <\})$$

Variables

Linear arithmetic admits the formulation of abstract, schematic assertions. (Object) variables are the technical tool for schematization.

We assume that

$$X$$

is a given countably infinite set of symbols which we use for (the denotation of) *variables*.

Context-Free Grammars

We define many of our notions on the bases of context-free grammars. Recall, that a context-free grammar $G = (N, T, P, S)$ consists of:

- a set of non-terminal symbols N
- a set of terminal symbols T
- a set P of rules $A ::= w$ where $A \in N$ and $w \in (N \cup T)^*$
- a start symbol S where $S \in N$

For rules $A ::= w_1$, $A ::= w_2$ we write $A ::= w_1 \mid w_2$

Terms

Terms over Σ_{LA} (resp., Σ_{LA}-terms) are formed according to these syntactic rules:

$$\begin{array}{lcll} s,t,u,v & ::= & x \mid q * x \mid q \quad , x \in X, q \in \mathbb{Q} & \text{(variable, rational)} \\ & & \mid s + t \mid s - t & \text{(sum, difference)} \end{array}$$

By $\mathrm{T}_{\Sigma_{\mathrm{LA}}}(X)$ we denote the set of Σ_{LA}-terms (over X). A term not containing any variable is called a *ground term*. By $\mathrm{T}_{\Sigma_{\mathrm{LA}}}$ we denote the set of Σ_{LA}-ground terms.

Atoms

Atoms (also called atomic formulas) over Σ_{LA} are formed according to this syntax:

$$\begin{array}{lcll} A, B & ::= & s \geq t \mid s \leq t & , s,t \in \mathrm{T}_{\Sigma_{\mathrm{LA}}}(X) \text{ (non-strict)} \\ & & \mid s > t \mid s < t & , s,t \in \mathrm{T}_{\Sigma_{\mathrm{LA}}}(X) \text{ (strict)} \end{array}$$

Quantifier Free Formulas

$\mathrm{QF}_{\Sigma_{\mathrm{LA}}}(X)$ is the set of positive boolean formulas over Σ_{LA} defined as follows:

$$\begin{array}{lcll} F, G, H & ::= & \bot & \text{(falsum)} \\ & \mid & \top & \text{(verum)} \\ & \mid & A & \text{(atomic formula)} \\ & \mid & \neg F & \text{(negation)} \\ & \mid & (F \wedge G) & \text{(conjunction)} \\ & \mid & (F \vee G) & \text{(disjunction)} \\ & \mid & (F \rightarrow G) & \text{(implication)} \\ & \mid & (F \leftrightarrow G) & \text{(equivalence)} \end{array}$$

Linear Arithmetic Semantics

The Σ_{LA}-*algebra* (also called Σ_{LA}-interpretation or Σ_{LA}-structure) is the triple

$$\mathcal{A}_{\mathrm{LA}} = (\mathbb{Q},\ (+_{\mathcal{A}_{\mathrm{LA}}}, -_{\mathcal{A}_{\mathrm{LA}}}, *_{\mathcal{A}_{\mathrm{LA}}}),\ (\leq_{\mathcal{A}_{\mathrm{LA}}}, \geq_{\mathcal{A}_{\mathrm{LA}}}, <_{\mathcal{A}_{\mathrm{LA}}}, >_{\mathcal{A}_{\mathrm{LA}}}))$$

where $+_{\mathcal{A}_{\mathrm{LA}}}, -_{\mathcal{A}_{\mathrm{LA}}}, *_{\mathcal{A}_{\mathrm{LA}}}, \leq_{\mathcal{A}_{\mathrm{LA}}}, \geq_{\mathcal{A}_{\mathrm{LA}}}, <_{\mathcal{A}_{\mathrm{LA}}}, >_{\mathcal{A}_{\mathrm{LA}}}$ are the "standard" intepretations of $+, -, *, \leq, \geq, <, >$, respectively.

Linear Arithmetic Assignments

A variable has no intrinsic meaning. The meaning of a variable has to be defined externally (explicitly or implicitly in a given context) by an assignment.

A *(variable) assignment*, also called a *valuation* for linear arithmetic is a map $\beta : X \to \mathbb{Q}$.

Truth Value of a Formula with Respect to β

$\mathcal{A}_{\mathrm{LA}}(\beta) : \mathrm{QF}_{\Sigma_{\mathrm{LA}}}(X) \to \{0, 1\}$ is defined inductively as follows:

$$\begin{aligned}
\mathcal{A}_{\mathrm{LA}}(\beta)(\bot) &= 0\\
\mathcal{A}_{\mathrm{LA}}(\beta)(\top) &= 1\\
\mathcal{A}_{\mathrm{LA}}(\beta)(s \sharp t) = 1 \quad &\Leftrightarrow \quad (\mathcal{A}_{\mathrm{LA}}(\beta)(s) \sharp_{\mathcal{A}_{\mathrm{LA}}} \mathcal{A}_{\mathrm{LA}}(\beta)(t))\\
&\quad \sharp \in \{\leq, \geq, <, >\}\\
\mathcal{A}_{\mathrm{LA}}(\beta)(\neg F) = 1 \quad &\Leftrightarrow \quad \mathcal{A}_{\mathrm{LA}}(\beta)(F) = 0\\
\mathcal{A}_{\mathrm{LA}}(\beta)(F \rho G) &= \mathsf{B}_\rho(\mathcal{A}_{\mathrm{LA}}(\beta)(F), \mathcal{A}_{\mathrm{LA}}(\beta)(G))\\
&\quad \text{with } \mathsf{B}_\rho \text{ the Boolean function associated with } \rho
\end{aligned}$$

$\mathcal{A}_{\mathrm{LA}}(\beta)(x) = \beta(x)$, $\mathcal{A}_{\mathrm{LA}}(\beta)(s \circ t) = \mathcal{A}_{\mathrm{LA}}(\beta)(s) \circ_{\mathcal{A}_{\mathrm{LA}}} \mathcal{A}_{\mathrm{LA}}(\beta)(t)$, $\circ \in \{+, -, *\}$, $\mathcal{A}_{\mathrm{LA}}(\beta)(q) = q$ for all $q \in \mathbb{Q}$.

2.2 Models, Validity, and Satisfiability

F is *valid* in $\mathcal{A}_{\mathrm{LA}}$ under assignment β:

$$\mathcal{A}_{\mathrm{LA}}, \beta \models F \quad :\Leftrightarrow \quad \mathcal{A}_{\mathrm{LA}}(\beta)(F) = 1$$

F is *valid* in $\mathcal{A}_{\mathrm{LA}}$ ($\mathcal{A}_{\mathrm{LA}}$ is a *model* of F):

$$\mathcal{A}_{\mathrm{LA}} \models F \quad :\Leftrightarrow \quad \mathcal{A}_{\mathrm{LA}}, \beta \models F, \text{ for all } \beta \in X \to \mathbb{Q}$$

F is called *satisfiable* iff there exist a β such that $\mathcal{A}_{\mathrm{LA}}, \beta \models F$. Otherwise F is called *unsatisfiable*.

On Quantification

Linear arithmetic can also be considered with respect to quantification. The quantifiers are $\exists$ meaning "there exists" and $\forall$ meaning "for all". For example, $\exists x\,(x \geq 0)$ is valid (or true) in $\mathcal{A}_{\mathrm{LA}}$, $\forall x\,(x \geq 0)$ is unsatisfiable (or false) and $\forall x\,(x \geq 0 \vee x < 0)$ is again valid.

Note that a quantifier free formula is satisfiable iff the existential closure of the formula is valid. If we introduce new free constants c_i for the variables x_i of a quantifier free formula, where $\mathcal{A}_{\mathrm{LA}}(c_i) = q_i$ for some $q_i \in \mathbb{Q}$, then a quantifier free formula is satisfiable iff the same formula where variables are replaced by new free constants is satisfiable.

Some Important LA Equivalences

Proposition 2.1 *The following equivalences are valid for all LA terms s, t:*

$$\begin{array}{ll} \neg s \geq t \leftrightarrow s < t & \\ \neg s \leq t \leftrightarrow s > t & \textit{(Negation)} \\ & \\ (s = t) \leftrightarrow (s \leq t \wedge s \geq t) & \textit{(Equality)} \\ & \\ s \geq t \leftrightarrow t \leq s & \\ s > t \leftrightarrow t < s & \textit{(Swap)} \end{array}$$

With $\lesssim$ we abbreviate $<$ or $\leq$.

The Fourier-Motzkin Procedure

boolean FM(Set N of LA atoms) {
  if ($N = \emptyset$) return true;
  elsif (N is ground) return $\mathcal{A}_{\mathrm{LA}}(N)$;
  else {
    select a variable x from N;
    transform all atoms in N containing x into $s_i \lesssim x$, $x \lesssim t_j$
    and the subset N' of atoms not containing x;
    compute $N^* := \{s_i \lesssim_{i,j} t_j \mid s_i \lesssim_i x \in N,\ x \lesssim_j t_j \in N \text{ for all } i, j\}$
    where $\lesssim_{i,j}$ is strict iff at least one of $\lesssim_i$, $\lesssim_j$ is strict
    return FM($N' \cup N^*$);
  }
}

Properties of the Fourier-Motzkin Procedure

- Any ground set N of linear arithmetic atoms can be easily decided.
- FM(N) terminates on any N as in recursive calls N has strictly less variables.
- The set $N' \cup N^*$ is worst case of size $O(|N|^2)$.
- FM(N)=true iff N is satisfiable in $\mathcal{A}_{\mathrm{LA}}$.
- The procedure was invented by Fourier (1826), forgotten, and then rediscovered by Dines (1919) and Motzkin (1936).
- There are more efficient methods known, e.g., the simplex algorithm.

2.3 The DPLL(T) Procedure

Goal:
Given a propositional formula in CNF (or alternatively, a finite set N of clauses), where the atoms represent ground formulas over some theory T, check whether it is satisfiable in T. (and optionally: output *one* solution, if it is satisfiable).

Assumption:
Again, clauses contain neither duplicated literals nor complementary literals.

Remark:
We will use LA as an ongoing example for T and consider DPLL(LA).

On LA as a Theory

We consider a specific formula language together with a satisfiability check for conjunctions of atoms (literals) as a theory T. Note that a valuation M is interpreted as the conjunction of its literals.

Later on we will introduce theory notions based on sets of formulas or models.

For LA we consider the language defined before and Fourier-Motzkin as the satisfiability check for conjunctions of atoms. Variables in formulas without quantification can actually be considered as constants.

Notions with Respect to the Theory T

If a partial valuation M is T-consistent (satisfiable) and F a formula such that $M \models F$, then we say that *M is a T-model of F*.

If F and G are formulas then *F entails G in T*, written $F \models_T G$ if $F \wedge \neg G$ is T-inconsistent.

Example: $x > 1 \not\models x > 0$ but $x > 1 \models_{\mathrm{LA}} x > 0$

Remark

M stands again for a list of propositional literals. As every propositional literal stands for a ground literal from T, there are actually two interpretations of M. We write $M \models F$ if F is entailed by M propositionally. We write $M \models_T F$ if the T ground formulas represented by M entail F.

DPLL(T) Rules from DPLL

Unit Propagate:

$$M \parallel N \cup \{C \vee L\} \;\Rightarrow_{\mathrm{DPLL(T)}}\; M\ L \parallel N \cup \{C \vee L\}$$

if C is false under M and L is undefined under M.

Decide:

$$M \parallel N \;\Rightarrow_{\mathrm{DPLL(T)}}\; M\ L^{\mathrm{d}} \parallel N$$

if L is undefined under M.

Fail:

$$M \parallel N \cup \{C\} \;\Rightarrow_{\mathrm{DPLL(T)}}\; \mathit{fail}$$

if C is false under M and M contains no decision literals.

Specific DPLL(T) Rules

T-Backjump:

$$M\ L^{\mathrm{d}}\ M' \parallel N \cup \{C\} \ \Rightarrow_{\mathrm{DPLL(T)}}\ M\ L' \parallel N \cup \{C\}$$

if $M\ L^{\mathrm{d}}\ M' \models \neg C$
there is some "backjump clause" $C' \vee L'$ such that
$N \cup \{C\} \models_T C' \vee L'$ and $M \models \neg C'$
L' is undefined under M', and
L' or $\overline{L'}$ occurs in N or in $M\ L^{\mathrm{d}}\ M'$.

T-Learn:

$$M \parallel N \ \Rightarrow_{\mathrm{DPLL(T)}}\ M \parallel N \cup \{C\}$$

if $N \models_T C$ and each atom of C occurs in N or M.

T-Forget:

$$M \parallel N \cup \{C\} \ \Rightarrow_{\mathrm{DPLL(T)}}\ M \parallel N$$

if $N \models_T C$.

T-Propagate:

$$M \parallel N \ \Rightarrow_{\mathrm{DPLL(T)}}\ M\ L \parallel N$$

if $M \models_T L$ where L is undefined in M and
L or $\overline{L}$ occurs in N.

DPLL(T) Properties

The DPLL modulo theories system DPLL(T) consists of the rules Decide, Fail, Unit-Propagate, T-Propagate, T-Backjump, T-Learn and T-Forget.

The Lemma 1.9 and the Lemma 1.10 from DPLL hold accordingly for DPLL(T). Again we will reconsider termination when the needed notions on orderings are established.

Lemma 2.2 *If $\emptyset \parallel N \ \Rightarrow^*_{\mathrm{DPLL(T)}}\ M \parallel N'$ and there is some conflicting clause in $M \parallel N'$, that is, $M \models \neg C$ for some clause C in N, then either Fail or T-Backjump applies to $M \parallel N'$.*

Proof. As in Lemma 1.11. □

Lemma 2.3 *If $\emptyset \parallel N \Rightarrow^*_{\text{DPLL(T)}} M \parallel N'$ and M is T-inconsistent, then either there is a conflicting clause in $M \parallel N'$, or else T-Learn applies to $M \parallel N'$, generating a conflicting clause. (Proof follows)*

Proof. If M is T-inconsistent, then there exists a subsequence $(L_1, \dots, L_n)$ of M such that $\emptyset \models_T \overline{L_1} \vee \dots \vee \overline{L_n}$. Hence the conflicting clause $\overline{L_1} \vee \dots \vee \overline{L_n}$ is either in $M \parallel N'$, or else it can be learned by one T-Learn step. □

Theorem 2.4 *Consider a derivation $\emptyset \parallel N \Rightarrow^*_{\text{DPLL(T)}} S$, where no more rule of the DPLL(T) procedure is applicable to S except T-Learn or T-forget, and if S has the form $M \parallel N'$ then M is T-consistent. Then*

(1) S is fail iff N is T-unsatisfiable.

(2) If S has the form $M \parallel N'$, then M is a T-model of N.

3 First-Order Logic

First-order logic

- formalizes fundamental mathematical concepts
- is expressive (Turing-complete)
- is not too expressive (e. g. not axiomatizable: natural numbers, uncountable sets)
- has a rich structure of decidable fragments
- has a rich model and proof theory

First-order logic is also called (first-order) *predicate logic*.

3.1 Syntax

Syntax:

- non-logical symbols (domain-specific)
  ⇒ terms, atomic formulas
- logical symbols (domain-independent)
  ⇒ Boolean combinations, quantifiers

Signature

A signature

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

fixes an alphabet of non-logical symbols, where

- Ω is a set of *function symbols* f with *arity* $n \geq 0$, written $\mathsf{arity}(f) = n$,
- Π is a set of *predicate symbols* p with arity $m \geq 0$, written $\mathsf{arity}(p) = m$.

If $n = 0$ then f is also called a *constant (symbol)*.
If $m = 0$ then p is also called a *propositional variable*.
We use letters P, Q, R, S, to denote propositional variables.

Refined concept for practical applications:
many-sorted signatures (corresponds to simple type systems in programming languages); not so interesting from a logical point of view.

Variables

Predicate logic admits the formulation of abstract, schematic assertions. (Object) variables are the technical tool for schematization.

We assume that

$$X$$

is a given countably infinite set of symbols which we use for (the denotation of) *variables*.

Context-Free Grammars

We define many of our notions on the bases of context-free grammars. Recall, that a context-free grammar $G = (N, T, P, S)$ consists of:

- a set of non-terminal symbols N
- a set of terminal symbols T
- a set P of rules $A ::= w$ where $A \in N$ and $w \in (N \cup T)^*$
- a start symbol S where $S \in N$

For rules $A ::= w_1$, $A ::= w_2$ we write $A ::= w_1 \mid w_2$

Terms

Terms over Σ (resp., Σ-terms) are formed according to these syntactic rules:

$$\begin{array}{llll} s,t,u,v & ::= & x & ,\ x \in X \quad \text{(variable)} \\ & \mid & f(s_1,...,s_n) & ,\ f \in \Omega,\ \mathsf{arity}(f) = n \quad \text{(functional term)} \end{array}$$

By $\mathrm{T}_\Sigma(X)$ we denote the set of Σ-terms (over X). A term not containing any variable is called a *ground term*. By T_Σ we denote the set of Σ-ground terms.

In other words, terms are formal expressions with well-balanced brackets which we may also view as marked, ordered trees. The markings are function symbols or variables. The nodes correspond to the *subterms* of the term. A node v that is marked with a function symbol f of arity n has exactly n subtrees representing the n immediate subterms of v.

Atoms

Atoms (also called atomic formulas) over Σ are formed according to this syntax:

$$\begin{array}{llll} A,B & ::= & p(s_1,...,s_m) & ,\ p \in \Pi,\ \mathsf{arity}(p) = m \\ \Big[& \mid & (s \approx t) & \text{(equation)} \quad \Big] \end{array}$$

Whenever we admit equations as atomic formulas we are in the realm of *first-order logic with equality*. Admitting equality does not really increase the expressiveness of first-order logic, (cf. exercises). But deductive systems where equality is treated specifically can be much more efficient.

Literals

$$\begin{array}{llll} L & ::= & A & \text{(positive literal)} \\ & \mid & \neg A & \text{(negative literal)} \end{array}$$

Clauses

$$\begin{array}{llll} C,D & ::= & \bot & \text{(empty clause)} \\ & \mid & L_1 \vee \ldots \vee L_k,\ \ k \geq 1 & \text{(non-empty clause)} \end{array}$$

General First-Order Formulas

$F_\Sigma(X)$ is the set of first-order formulas over Σ defined as follows:

$$
\begin{array}{llll}
F, G, H & ::= & \bot & \text{(falsum)} \\
& \mid & \top & \text{(verum)} \\
& \mid & A & \text{(atomic formula)} \\
& \mid & \neg F & \text{(negation)} \\
& \mid & (F \wedge G) & \text{(conjunction)} \\
& \mid & (F \vee G) & \text{(disjunction)} \\
& \mid & (F \rightarrow G) & \text{(implication)} \\
& \mid & (F \leftrightarrow G) & \text{(equivalence)} \\
& \mid & \forall x F & \text{(universal quantification)} \\
& \mid & \exists x F & \text{(existential quantification)}
\end{array}
$$

Positions in terms, formulas

Positions of a term s (formula F):

$\mathrm{pos}(x) = \{\varepsilon\}$,
$\mathrm{pos}(f(s_1, \ldots, s_n)) = \{\varepsilon\} \cup \bigcup_{i=1}^n \{\, ip \mid p \in \mathrm{pos}(s_i) \,\}$.

$\mathrm{pos}(\forall x F) = \{\varepsilon\} \cup \{\, 1p \mid p \in \mathrm{pos}(F) \,\}$
Analogously for all other formulas.

Prefix order for $p, q \in \mathrm{pos}(s)$:

p above q: $p \leq q$ if $pp' = q$ for some p',
p strictly above q: $p < q$ if $p \leq q$ and not $q \leq p$,
p and q parallel: $p \parallel q$ if neither $p \leq q$ nor $q \leq p$.

Subterm of s (F) at a position $p \in \mathrm{pos}(s)$:

$s/\varepsilon = s$,
$f(s_1, \ldots, s_n)/ip = s_i/p$.

Analougously for formulas (F/p).

Replacement of the subterm at position $p \in \mathrm{pos}(s)$ by t:

$s[t]_\varepsilon = t$,
$f(s_1, \ldots, s_n)[t]_{ip} = f(s_1, \ldots, s_i[t]_p, \ldots, s_n)$.

Analougously for formulas ($F[G]_p$).

Size of a term s:

$|s| =$ cardinality of $\mathrm{pos}(s)$.

Notational Conventions

We omit brackets according to the following rules:

- $\neg \quad >_p \quad \vee \quad >_p \quad \wedge \quad >_p \quad \rightarrow \quad >_p \quad \leftrightarrow$
  (binding precedences)
- $\vee$ and $\wedge$ are associative and commutative
- $\rightarrow$ is right-associative

$Qx_1, \ldots, x_n\, F$ abbreviates $Qx_1 \ldots Qx_n\, F$.

We use infix-, prefix-, postfix-, or mixfix-notation with the usual operator precedences.

Examples:

$$\begin{array}{ccc} s + t * u & \text{for} & +(s, *(t, u)) \\ s * u \leq t + v & \text{for} & \leq (*(s, u), +(t, v)) \\ -s & \text{for} & -(s) \\ 0 & \text{for} & 0() \end{array}$$

Example: Peano Arithmetic

$$\begin{array}{lcl} \Sigma_{PA} & = & (\Omega_{PA},\ \Pi_{PA}) \\ \Omega_{PA} & = & \{0/0,\ +/2,\ */2,\ s/1\} \\ \Pi_{PA} & = & \{\leq /2,\ < /2\} \end{array}$$

$+, *, <, \leq$ infix; $* \ >_p \ + \ >_p \ < \ >_p \ \leq$

Examples of formulas over this signature are:

$\forall x, y(x \leq y \leftrightarrow \exists z(x + z \approx y))$
$\exists x \forall y(x + y \approx y)$
$\forall x, y(x * s(y) \approx x * y + x)$
$\forall x, y(s(x) \approx s(y) \rightarrow x \approx y)$
$\forall x \exists y(x < y \wedge \neg \exists z(x < z \wedge z < y))$

Remarks About the Example

We observe that the symbols $\leq$, $<$, 0, s are redundant as they can be defined in first-order logic with equality just with the help of $+$. The first formula defines $\leq$, while the second defines zero. The last formula, respectively, defines s.

Eliminating the existential quantifiers by Skolemization (cf. below) reintroduces the "redundant" symbols.

Consequently there is a *trade-off* between the complexity of the quantification structure and the complexity of the signature.

Bound and Free Variables

In QxF, $Q \in \{\exists, \forall\}$, we call F the *scope* of the quantifier Qx. An *occurrence* of a variable x is called *bound*, if it is inside the scope of a quantifier Qx. Any other occurrence of a variable is called *free*.

Formulas without free variables are also called *closed formulas* or *sentential forms*.

Formulas without variables are called *ground*.

Example:

$$\forall y \quad \overbrace{(\forall \overbrace{x \quad p(x)}^{scope} \quad \rightarrow \quad q(x, y))}^{scope}$$

The occurrence of y is bound, as is the first occurrence of x. The second occurrence of x is a free occurrence.

Substitutions

Substitution is a fundamental operation on terms and formulas that occurs in all inference systems for first-order logic.

In general, *substitutions* are mappings

$$\sigma : X \rightarrow \mathrm{T}_\Sigma(X)$$

such that the *domain* of σ, that is, the set

$$\mathit{dom}(\sigma) = \{x \in X \mid \sigma(x) \neq x\},$$

is finite. The set of variables *introduced* by σ, that is, the set of variables occurring in one of the terms $\sigma(x)$, with $x \in \mathit{dom}(\sigma)$, is denoted by $\mathit{codom}(\sigma)$.

Substitutions are often written as $[s_1/x_1, \ldots, s_n/x_n]$, with x_i pairwise distinct, and then denote the mapping

$$[s_1/x_1, \ldots, s_n/x_n](y) = \begin{cases} s_i, & \text{if } y = x_i \\ y, & \text{otherwise} \end{cases}$$

We also write $x\sigma$ for $\sigma(x)$.

The *modification* of a substitution σ at x is defined as follows:

$$\sigma[x \mapsto t](y) = \begin{cases} t, & \text{if } y = x \\ \sigma(y), & \text{otherwise} \end{cases}$$

Why Substitution is Complicated

We define the application of a substitution σ to a term t or formula F by structural induction over the syntactic structure of t or F by the equations depicted on the next page.

In the presence of quantification it is surprisingly complex: We need to make sure that the (free) variables in the codomain of σ are not *captured* upon placing them into the scope of a quantifier Qy, hence the bound variable must be renamed into a "fresh", that is, previously unused, variable z.

Why this definition of substitution is well-defined will be discussed below.

Application of a Substitution

"Homomorphic" extension of σ to terms and formulas:

$$
\begin{aligned}
f(s_1, \dots, s_n)\sigma &= f(s_1\sigma, \dots, s_n\sigma) \\
\bot\sigma &= \bot \\
\top\sigma &= \top \\
p(s_1, \dots, s_n)\sigma &= p(s_1\sigma, \dots, s_n\sigma) \\
(u \approx v)\sigma &= (u\sigma \approx v\sigma) \\
\neg F\sigma &= \neg(F\sigma) \\
(F\rho G)\sigma &= (F\sigma\,\rho\,G\sigma)\ ;\quad \text{for each binary connective } \rho \\
(Qx\,F)\sigma &= Qz\,(F\,\sigma[x \mapsto z])\ ;\quad \text{with } z \text{ a fresh variable}
\end{aligned}
$$

Structural Induction

Proposition 3.1 *Let $G = (N, T, P, S)$ be a context-free grammar (possibly infinite) and let q be a property of T^* (the words over the alphabet T of terminal symbols of G).*

q holds for all *words $w \in L(G)$, whenever one can prove the following two properties:*

1. *(*base cases*)*
   $q(w')$ holds for each $w' \in T^*$ such that $X ::= w'$ is a rule in P.
2. *(*step cases*)*
   If $X ::= w_0X_0w_1 \dots w_nX_nw_{n+1}$ is in P with $X_i \in N$, $w_i \in T^*$, $n \geq 0$, then for all $w_i' \in L(G, X_i)$, whenever $q(w_i')$ holds for $0 \leq i \leq n$, then also $q(w_0w_0'w_1 \dots w_nw_n'w_{n+1})$ holds.

Here $L(G, X_i) \subseteq T^*$ denotes the language generated by the grammar G from the nonterminal X_i.

Structural Recursion

Proposition 3.2 *Let $G = (N, T, P, S)$ be a* unambiguous *(why?) context-free grammar. A function f is well-defined on $L(G)$ (that is, unambiguously defined) whenever these 2 properties are satisfied:*

1. *(base cases)*
   f is well-defined on the words $w' \in T^*$ for each rule $X ::= w'$ in P.
2. *(step cases)*
   If $X ::= w_0 X_0 w_1 \ldots w_n X_n w_{n+1}$ is a rule in P then $f(w_0 w'_0 w_1 \ldots w_n w'_n w_{n+1})$ is well-defined, assuming that each of the $f(w'_i)$ is well-defined.

Substitution Revisited

Q: Does Proposition 3.2 justify that our homomorphic extension

$$apply : \mathrm{F}_\Sigma(X) \times (X \to \mathrm{T}_\Sigma(X)) \quad \to \quad \mathrm{F}_\Sigma(X),$$

with $apply(F, \sigma)$ denoted by $F\sigma$, of a substitution is well-defined?

A: We have two problems here. One is that *"fresh"* is (deliberately) left unspecified. That can be easily fixed by adding an extra variable counter argument to the apply function.

The second problem is that Proposition 3.2 applies to unary functions only. The standard solution to this problem is to curryfy, that is, to consider the binary function as a unary function producing a unary (residual) function as a result:

$$apply : \mathrm{F}_\Sigma(X) \quad \to \quad ((X \to \mathrm{T}_\Sigma(X)) \to \mathrm{F}_\Sigma(X))$$

where we have denoted $(apply(F))(\sigma)$ as $F\sigma$.

E: Convince yourself that this does the trick.

3.2 Semantics

To give semantics to a logical system means to define a notion of truth for the formulas. The concept of truth that we will now define for first-order logic goes back to Tarski.

As in the propositional case, we use a two-valued logic with truth values "true" and "false" denoted by 1 and 0, respectively.

Structures

A Σ-*algebra* (also called Σ-interpretation or Σ-structure) is a triple

$$\mathcal{A} = (U_{\mathcal{A}},\ (f_{\mathcal{A}} : U^n \to U)_{f\in\Omega},\ (p_{\mathcal{A}} \subseteq U_{\mathcal{A}}^m)_{p\in\Pi})$$

where $\mathsf{arity}(f) = n$, $\mathsf{arity}(p) = m$, $U_{\mathcal{A}} \neq \emptyset$ is a set, called the *universe* of $\mathcal{A}$.

By Σ-*Alg* we denote the class of all Σ-algebras.

Assignments

A variable has no intrinsic meaning. The meaning of a variable has to be defined externally (explicitly or implicitly in a given context) by an assignment.

A *(variable) assignment*, also called a *valuation* (over a given Σ-algebra $\mathcal{A}$), is a map $\beta : X \to U_{\mathcal{A}}$.

Variable assignments are the semantic counterparts of substitutions.

Value of a Term in $\mathcal{A}$ with Respect to β

By structural induction we define

$$\mathcal{A}(\beta) : \mathrm{T}_{\Sigma}(X) \to U_{\mathcal{A}}$$

as follows:

$$\begin{array}{rcl} \mathcal{A}(\beta)(x) & = & \beta(x), \qquad x \in X \\ \mathcal{A}(\beta)(f(s_1,\dots,s_n)) & = & f_{\mathcal{A}}(\mathcal{A}(\beta)(s_1),\dots,\mathcal{A}(\beta)(s_n)), \\ & & \qquad\qquad f \in \Omega, \mathsf{arity}(f) = n \end{array}$$

In the scope of a quantifier we need to evaluate terms with respect to modified assignments. To that end, let $\beta[x \mapsto a] : X \to U_{\mathcal{A}}$, for $x \in X$ and $a \in \mathcal{A}$, denote the assignment

$$\beta[x \mapsto a](y) := \begin{cases} a & \text{if } x = y \\ \beta(y) & \text{otherwise} \end{cases}$$

Truth Value of a Formula in $\mathcal{A}$ with Respect to β

$\mathcal{A}(\beta) : \mathrm{F}_\Sigma(X) \to \{0,1\}$ is defined inductively as follows:

$$
\begin{aligned}
\mathcal{A}(\beta)(\bot) &= 0 \\
\mathcal{A}(\beta)(\top) &= 1 \\
\mathcal{A}(\beta)(p(s_1,\ldots,s_n)) &= 1 \quad \Leftrightarrow \quad (\mathcal{A}(\beta)(s_1),\ldots,\mathcal{A}(\beta)(s_n)) \in p_{\mathcal{A}} \\
\mathcal{A}(\beta)(s \approx t) &= 1 \quad \Leftrightarrow \quad \mathcal{A}(\beta)(s) = \mathcal{A}(\beta)(t) \\
\mathcal{A}(\beta)(\neg F) &= 1 \quad \Leftrightarrow \quad \mathcal{A}(\beta)(F) = 0 \\
\mathcal{A}(\beta)(F \rho G) &= \mathsf{B}_\rho(\mathcal{A}(\beta)(F), \mathcal{A}(\beta)(G)) \\
& \quad \text{with } \mathsf{B}_\rho \text{ the Boolean function associated with } \rho \\
\mathcal{A}(\beta)(\forall x F) &= \min_{a \in U} \{\mathcal{A}(\beta[x \mapsto a])(F)\} \\
\mathcal{A}(\beta)(\exists x F) &= \max_{a \in U} \{\mathcal{A}(\beta[x \mapsto a])(F)\}
\end{aligned}
$$

Example

The "Standard" Interpretation for Peano Arithmetic:

$$
\begin{aligned}
U_{\mathbb{N}} &= \{0,1,2,\ldots\} \\
0_{\mathbb{N}} &= 0 \\
s_{\mathbb{N}} &: n \mapsto n+1 \\
+_{\mathbb{N}} &: (n,m) \mapsto n+m \\
*_{\mathbb{N}} &: (n,m) \mapsto n*m \\
\leq_{\mathbb{N}} &= \{(n,m) \mid n \text{ less than or equal to } m\} \\
<_{\mathbb{N}} &= \{(n,m) \mid n \text{ less than } m\}
\end{aligned}
$$

Note that $\mathbb{N}$ is just one out of many possible Σ_{PA}-interpretations.

Values over $\mathbb{N}$ for Sample Terms and Formulas:

Under the assignment $\beta : x \mapsto 1, y \mapsto 3$ we obtain

$$
\begin{aligned}
\mathbb{N}(\beta)(s(x)+s(0)) &= 3 \\
\mathbb{N}(\beta)(x+y \approx s(y)) &= 1 \\
\mathbb{N}(\beta)(\forall x, y(x+y \approx y+x)) &= 1 \\
\mathbb{N}(\beta)(\forall z\; z \leq y) &= 0 \\
\mathbb{N}(\beta)(\forall x \exists y\; x < y) &= 1
\end{aligned}
$$

3.3 Models, Validity, and Satisfiability

F is *valid* in $\mathcal{A}$ under assignment β:

$$\mathcal{A}, \beta \models F \quad :\Leftrightarrow \quad \mathcal{A}(\beta)(F) = 1$$

F is *valid* in $\mathcal{A}$ ($\mathcal{A}$ is a *model* of F):

$$\mathcal{A} \models F \quad :\Leftrightarrow \quad \mathcal{A}, \beta \models F, \text{ for all } \beta \in X \to U_{\mathcal{A}}$$

F is *valid* (or is a *tautology*):

$$\models F \quad :\Leftrightarrow \quad \mathcal{A} \models F, \text{ for all } \mathcal{A} \in \Sigma\text{-}\mathsf{Alg}$$

F is called *satisfiable* iff there exist $\mathcal{A}$ and β such that $\mathcal{A}, \beta \models F$. Otherwise F is called *unsatisfiable*.

Substitution Lemma

The following propositions, to be proved by structural induction, hold for all Σ-algebras $\mathcal{A}$, assignments β, and substitutions σ.

Lemma 3.3 *For any Σ-term t*

$$\mathcal{A}(\beta)(t\sigma) = \mathcal{A}(\beta \circ \sigma)(t),$$

where $\beta \circ \sigma : X \to \mathcal{A}$ is the assignment $\beta \circ \sigma(x) = \mathcal{A}(\beta)(x\sigma)$.

Proposition 3.4 *For any Σ-formula F, $\mathcal{A}(\beta)(F\sigma) = \mathcal{A}(\beta \circ \sigma)(F)$.*

Corollary 3.5 $\mathcal{A}, \beta \models F\sigma \quad \Leftrightarrow \quad \mathcal{A}, \beta \circ \sigma \models F$

These theorems basically express that the syntactic concept of substitution corresponds to the semantic concept of an assignment.

Entailment and Equivalence

F *entails* (*implies*) G (or G *is a consequence of* F), written $F \models G$, if for all $\mathcal{A} \in \Sigma\text{-}\mathsf{Alg}$ and $\beta \in X \to U_{\mathcal{A}}$, whenever $\mathcal{A}, \beta \models F$, then $\mathcal{A}, \beta \models G$.

F and G are called *equivalent*, written $F \models\!\mid G$, if for all $\mathcal{A} \in \Sigma\text{-}\mathsf{Alg}$ und $\beta \in X \to U_{\mathcal{A}}$ we have $\mathcal{A}, \beta \models F \quad \Leftrightarrow \quad \mathcal{A}, \beta \models G$.

Proposition 3.6 *F entails G iff $(F \to G)$ is valid*

Proposition 3.7 *F and G are equivalent iff $(F \leftrightarrow G)$ is valid.*

Extension to sets of formulas N in the "natural way", e. g., $N \models F$

$:\Leftrightarrow$ for all $\mathcal{A} \in \Sigma\text{-}\mathsf{Alg}$ and $\beta \in X \to U_{\mathcal{A}}$: if $\mathcal{A}, \beta \models G$, for all $G \in N$, then $\mathcal{A}, \beta \models F$.

Validity vs. Unsatisfiability

Validity and unsatisfiability are just two sides of the same medal as explained by the following proposition.

Proposition 3.8 *Let F and G be formulas, let N be a set of formulas. Then*

(i) F is valid if and only if $\neg F$ is unsatisfiable.

(ii) $F \models G$ if and only if $F \wedge \neg G$ is unsatisfiable.

(iii) $N \models G$ if and only if $N \cup \{\neg G\}$ is unsatisfiable.

Hence in order to design a theorem prover (validity checker) it is sufficient to design a checker for unsatisfiability.

Theory of a Structure

Let $\mathcal{A} \in \Sigma\text{-}\mathsf{Alg}$. The *(first-order) theory* of $\mathcal{A}$ is defined as

$$Th(\mathcal{A}) = \{G \in \mathrm{F}_{\Sigma}(X) \mid \mathcal{A} \models G\}$$

Problem of axiomatizability:

For which structures $\mathcal{A}$ can one *axiomatize* $Th(\mathcal{A})$, that is, can one write down a formula F (or a recursively enumerable set F of formulas) such that

$$Th(\mathcal{A}) = \{G \mid F \models G\}?$$

Analogously for sets of structures.

Two Interesting Theories

Let $\Sigma_{Pres} = (\{0/0, s/1, +/2\},\ \emptyset)$ and $\mathbb{Z}_+ = (\mathbb{Z}, 0, s, +)$ its standard interpretation on the integers. $Th(\mathbb{Z}_+)$ is called *Presburger arithmetic* (M. Presburger, 1929). (There is no essential difference when one, instead of $\mathbb{Z}$, considers the natural numbers $\mathbb{N}$ as standard interpretation.)

Presburger arithmetic is decidable in 3EXPTIME (D. Oppen, JCSS, 16(3):323–332, 1978), and in 2EXPSPACE, using automata-theoretic methods (and there is a constant $c \geq 0$ such that $Th(\mathbb{Z}_+) \notin \text{NTIME}(2^{2^{cn}})$).

However, $\mathbb{N}_* = (\mathbb{N}, 0, s, +, *)$, the standard interpretation of $\Sigma_{PA} = (\{0/0, s/1, +/2, */2\}, \emptyset)$, has as theory the so-called *Peano arithmetic* which is undecidable, not even recursively enumerable.

Note: The choice of signature can make a big difference with regard to the computational complexity of theories.

3.4 Algorithmic Problems

Validity(F): $\models F$?

Satisfiability(F): F satisfiable?

Entailment(F,G): does F entail G?

Model(A,F): $A \models F$?

Solve(A,F): find an assignment β such that $A, \beta \models F$.

Solve(F): find a substitution σ such that $\models F\sigma$.

Abduce(F): find G with "certain properties" such that $G \models F$.

Gödel's Famous Theorems

1. For most signatures Σ, validity is undecidable for Σ-formulas. (One can easily encode Turing machines in most signatures.)
2. For each signature Σ, the set of valid Σ-formulas is recursively enumerable. (We will prove this by giving complete deduction systems.)
3. For $\Sigma = \Sigma_{PA}$ and $\mathbb{N}_* = (\mathbb{N}, 0, s, +, *)$, the theory $Th(\mathbb{N}_*)$ is not recursively enumerable.

These complexity results motivate the study of subclasses of formulas (*fragments*) of first-order logic

Q: Can you think of any fragments of first-order logic for which validity is decidable?

Some Decidable Fragments

Some decidable fragments:

- *Monadic class*: no function symbols, all predicates unary; validity is NEXPTIME-complete.
- Variable-free formulas without equality: satisfiability is NP-complete. (why?)
- Variable-free Horn clauses (clauses with at most one positive atom): entailment is decidable in linear time.
- Finite model checking is decidable in time polynomial in the size of the structure and the formula.

3.5 Normal Forms and Skolemization (Traditional)

Study of normal forms motivated by

- reduction of logical concepts,
- efficient data structures for theorem proving.

The main problem in first-order logic is the treatment of quantifiers. The subsequent normal form transformations are intended to eliminate many of them.

Prenex Normal Form

Prenex formulas have the form

$$Q_1 x_1 \ldots Q_n x_n \ F,$$

where F is quantifier-free and $Q_i \in \{\forall, \exists\}$; we call $Q_1 x_1 \ldots Q_n x_n$ the *quantifier prefix* and F the *matrix* of the formula.

Computing prenex normal form by the rewrite relation $\Rightarrow_P$:

$$\begin{array}{rcll}
(F \leftrightarrow G) & \Rightarrow_P & (F \rightarrow G) \wedge (G \rightarrow F) & \\
\neg QxF & \Rightarrow_P & \overline{Q}x\neg F & (\neg Q) \\
(QxF \ \rho \ G) & \Rightarrow_P & Qy(F[y/x] \ \rho \ G), \ y \text{ fresh}, \ \rho \in \{\wedge, \vee\} & \\
(QxF \rightarrow G) & \Rightarrow_P & \overline{Q}y(F[y/x] \rightarrow G), \ y \text{ fresh} & \\
(F \ \rho \ QxG) & \Rightarrow_P & Qy(F \ \rho \ G[y/x]), \ y \text{ fresh}, \ \rho \in \{\wedge, \vee, \rightarrow\} &
\end{array}$$

Here $\overline{Q}$ denotes the quantifier *dual* to Q, i. e., $\overline{\forall} = \exists$ and $\overline{\exists} = \forall$.

Skolemization

Intuition: replacement of $\exists y$ by a concrete choice function computing y from all the arguments y depends on.

Transformation $\Rightarrow_S$ (to be applied outermost, *not* in subformulas):

$$\forall x_1, \ldots, x_n \exists y F \ \Rightarrow_S \ \forall x_1, \ldots, x_n F[f(x_1, \ldots, x_n)/y]$$

where f, where $\mathsf{arity}(f) = n$, is a new function symbol (*Skolem function*).

Together: $F \stackrel{*}{\Rightarrow}_P \underbrace{G}_{\text{prenex}} \stackrel{*}{\Rightarrow}_S \underbrace{H}_{\text{prenex, no } \exists}$

Theorem 3.9 *Let F, G, and H as defined above and closed. Then* *(i)* F *and* G *are equivalent.*

(ii) $H \models G$ *but the converse is not true in general.*

(iii) G *satisfiable (w.r.t.* Σ*-Alg)* $\Leftrightarrow$ H *satisfiable (w.r.t.* Σ'*-Alg) where* $\Sigma' = (\Omega \cup SKF, \Pi)$*, if* $\Sigma = (\Omega, \Pi)$*.*

Clausal Normal Form (Conjunctive Normal Form)

$$\begin{array}{rcl}
(F \leftrightarrow G) & \Rightarrow_K & (F \rightarrow G) \wedge (G \rightarrow F) \\
(F \rightarrow G) & \Rightarrow_K & (\neg F \vee G) \\
\neg(F \vee G) & \Rightarrow_K & (\neg F \wedge \neg G) \\
\neg(F \wedge G) & \Rightarrow_K & (\neg F \vee \neg G) \\
\neg\neg F & \Rightarrow_K & F \\
(F \wedge G) \vee H & \Rightarrow_K & (F \vee H) \wedge (G \vee H) \\
(F \wedge \top) & \Rightarrow_K & F \\
(F \wedge \bot) & \Rightarrow_K & \bot \\
(F \vee \top) & \Rightarrow_K & \top \\
(F \vee \bot) & \Rightarrow_K & F
\end{array}$$

These rules are to be applied modulo associativity and commutativity of $\wedge$ and $\vee$. The first five rules, plus the rule $(\neg Q)$, compute the *negation normal form* (NNF) of a formula.

The Complete Picture

$$\begin{array}{lll}
F & \stackrel{*}{\Rightarrow}_P & Q_1 y_1 \ldots Q_n y_n\ G \qquad\qquad (G \text{ quantifier-free}) \\
 & \stackrel{*}{\Rightarrow}_S & \forall x_1, \ldots, x_m\ H \qquad\qquad (m \leq n,\ H \text{ quantifier-free}) \\
 & \stackrel{*}{\Rightarrow}_K & \underbrace{\underbrace{\forall x_1, \ldots, x_m}_{\text{leave out}}\ \bigwedge_{i=1}^{k}\ \underbrace{\bigvee_{j=1}^{n_i} L_{ij}}_{\text{clauses } C_i}}_{F'}
\end{array}$$

$N = \{C_1, \ldots, C_k\}$ is called the *clausal (normal) form* (CNF) of F.
Note: the variables in the clauses are implicitly universally quantified.

Theorem 3.10 *Let* F *be closed. Then* $F' \models F$*. (The converse is not true in general.)*

Theorem 3.11 *Let* F *be closed. Then* F *is satisfiable iff* F' *is satisfiable iff* N *is satisfiable*

Optimization

Here is lots of room for optimization since we only can preserve satisfiability anyway:

- size of the CNF exponential when done naively;
  but see the transformations we introduced for propositional logic
- want to preserve the original formula structure;
- want small arity of Skolem functions (follows)

3.6 Getting small Skolem Functions

- produce a negation normal form (NNF)
- apply miniscoping
- rename all variables
- skolemize

Negation Normal Form (NNF)

Apply the rewrite relation $\Rightarrow_{NNF}$, F is the overall formula:

$$
\begin{array}{rcl}
G \leftrightarrow H & \Rightarrow_{NNF} & (G \rightarrow H) \wedge (H \rightarrow G) \\
& & \text{if } F/p = G \leftrightarrow H \text{ and } F/p \text{ has positive polarity} \\
G \leftrightarrow H & \Rightarrow_{NNF} & (G \wedge H) \vee (\neg H \wedge \neg G) \\
& & \text{if } F/p = G \leftrightarrow H \text{ and } F/p \text{ has negative polarity} \\
\neg Qx\, G & \Rightarrow_{NNF} & \overline{Q}x \neg G \\
\neg(G \vee H) & \Rightarrow_{NNF} & \neg G \wedge \neg H \\
\neg(G \wedge H) & \Rightarrow_{NNF} & \neg G \vee \neg H \\
G \rightarrow H & \Rightarrow_{NNF} & \neg G \vee H \\
\neg\neg G & \Rightarrow_{NNF} & G
\end{array}
$$

Miniscoping

Apply the rewrite relation $\Rightarrow_{MS}$. For the below rules we assume that x occurs freely in G, H, but x does not occur freely in F:

$$
\begin{array}{rcl}
Qx\,(G \wedge F) & \Rightarrow_{MS} & Qx\, G \wedge F \\
Qx\,(G \vee F) & \Rightarrow_{MS} & Qx\, G \vee F \\
\forall x\,(G \wedge H) & \Rightarrow_{MS} & \forall x\, G \wedge \forall x\, H \\
\exists x\,(G \vee H) & \Rightarrow_{MS} & \exists x\, G \vee \exists x\, H
\end{array}
$$

Variable Renaming

Rename all variables in F such that there are no two different positions p, q with $F/p = Qx\,G$ and $F/q = Qx\,H$.

Standard Skolemization

Let F be the overall formula, then apply the rewrite rule:

$$\begin{array}{lll} \exists x\, H & \Rightarrow_{SK} & H[f(y_1,\dots,y_n)/x] \\ & \multicolumn{2}{l}{\text{if } F/p = \exists x\, H \text{ and } p \text{ has minimal length,}} \\ & \multicolumn{2}{l}{\{y_1,\dots,y_n\} \text{ are the free variables in } \exists x\, H,} \\ & \multicolumn{2}{l}{f \text{ is a new function symbol, } \mathsf{arity}(f) = n} \end{array}$$

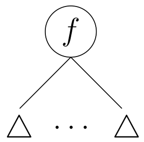
3.7 Herbrand Interpretations

From now an we shall consider PL without equality. Ω shall contains at least one constant symbol.

A *Herbrand interpretation* (over Σ) is a Σ-algebra $\mathcal{A}$ such that

- $U_\mathcal{A} = \mathrm{T}_\Sigma$ (= the set of ground terms over Σ)
- $f_\mathcal{A} : (s_1,\dots,s_n) \mapsto f(s_1,\dots,s_n),\ f \in \Omega, \mathsf{arity}(f) = n$

$f_\mathcal{A}(\triangle,\dots,\triangle) =$



In other words, *values are fixed* to be ground terms and *functions are fixed* to be the *term constructors*. Only predicate symbols $p \in \Pi$, $\mathsf{arity}(p) = m$ may be freely interpreted as relations $p_\mathcal{A} \subseteq \mathrm{T}_\Sigma^m$.

Proposition 3.12 *Every set of ground atoms I uniquely determines a Herbrand interpretation $\mathcal{A}$ via*

$$(s_1,\dots,s_n) \in p_\mathcal{A} \quad :\Leftrightarrow \quad p(s_1,\dots,s_n) \in I$$

Thus we shall identify Herbrand interpretations (over Σ) with sets of Σ-ground atoms.

Example: $\Sigma_{Pres} = (\{0/0, s/1, +/2\},\ \{</2, \leq/2\})$

$\mathbb{N}$ as Herbrand interpretation over Σ_{Pres}:

$$
\begin{array}{ll}
I = \{ & 0 \leq 0,\ 0 \leq s(0),\ 0 \leq s(s(0)),\ \ldots, \\
& 0+0 \leq 0,\ 0+0 \leq s(0),\ \ldots, \\
& \ldots,\ (s(0)+0)+s(0) \leq s(0)+(s(0)+s(0)) \\
& \ldots \\
& s(0)+0 < s(0)+0+0+s(0) \\
& \ldots\}
\end{array}
$$

Existence of Herbrand Models

A Herbrand interpretation I is called a *Herbrand model* of F, if $I \models F$.

Theorem 3.13 (Herbrand) *Let N be a set of Σ-clauses.*

$$
\begin{array}{lll}
N \text{ satisfiable} & \Leftrightarrow & N \text{ has a Herbrand model (over } \Sigma) \\
& \Leftrightarrow & G_\Sigma(N) \text{ has a Herbrand model (over } \Sigma)
\end{array}
$$

where $G_\Sigma(N) = \{C\sigma \text{ ground clause} \mid C \in N,\ \sigma : X \to \mathrm{T}_\Sigma\}$ *is the set of* ground instances *of* N.

[The proof will be given below in the context of the completeness proof for resolution.]

Example of a G_Σ

For Σ_{Pres} one obtains for

$$C = (x < y) \vee (y \leq s(x))$$

the following ground instances:

$(0 < 0) \vee (0 \leq s(0))$
$(s(0) < 0) \vee (0 \leq s(s(0)))$
$\ldots$
$(s(0)+s(0) < s(0)+0) \vee (s(0)+0 \leq s(s(0)+s(0)))$
$\ldots$

3.8 Inference Systems and Proofs

Inference systems Γ (proof calculi) are sets of tuples

$$(F_1, \ldots, F_n, F_{n+1}),\ n \geq 0,$$

called *inferences* or *inference rules*, and written

$$\frac{\overbrace{F_1 \ \ldots \ F_n}^{\text{premises}}}{\underbrace{F_{n+1}}_{\text{conclusion}}}\ .$$

Clausal inference system: premises and conclusions are clauses. One also considers inference systems over other data structures (cf. below).

Proofs

A *proof* in Γ of a formula F from a a set of formulas N (called *assumptions*) is a sequence $F_1, \ldots, F_k$ of formulas where

(i) $F_k = F$,

(ii) for all $1 \leq i \leq k$: $F_i \in N$, or else there exists an inference

$$\frac{F_{i_1} \ \ldots \ F_{i_{n_i}}}{F_i}$$

in Γ, such that $0 \leq i_j < i$, for $1 \leq j \leq n_i$.

Soundness and Completeness

Provability $\vdash_\Gamma$ of F from N in Γ: $N \vdash_\Gamma F$ $:\Leftrightarrow$ there exists a proof Γ of F from N.

Γ is called *sound* $:\Leftrightarrow$

$$\frac{F_1 \ \ldots \ F_n}{F} \in \Gamma \quad \Rightarrow \quad F_1, \ldots, F_n \models F$$

Γ is called *complete* $:\Leftrightarrow$

$$N \models F \quad \Rightarrow \quad N \vdash_\Gamma F$$

Γ is called *refutationally complete* $:\Leftrightarrow$

$$N \models \bot \quad \Rightarrow \quad N \vdash_\Gamma \bot$$

Proposition 3.14

(i) Let Γ be sound. Then $N \vdash_\Gamma F \Rightarrow N \models F$

(ii) $N \vdash_\Gamma F \Rightarrow$ there exist $F_1, \dots, F_n \in N$ s.t. $F_1, \dots, F_n \vdash_\Gamma F$ (resembles compactness).

Proofs as Trees

markings	$\widehat{=}$	formulas			
leaves	$\widehat{=}$	assumptions and axioms			
other nodes	$\widehat{=}$	inferences:	conclusion	$\widehat{=}$	ancestor
			premises	$\widehat{=}$	direct descendants

$$\frac{P(f(c)) \qquad \dfrac{\dfrac{\dfrac{P(f(c)) \lor Q(b) \qquad \dfrac{\dfrac{P(f(c)) \lor Q(b) \qquad \neg P(f(c)) \lor \neg P(f(c)) \lor Q(b)}{\neg P(f(c)) \lor Q(b) \lor Q(b)}}{\neg P(f(c)) \lor Q(b)}}{Q(b) \lor Q(b)}}{Q(b)} \qquad \neg P(f(c)) \lor \neg Q(b)}{\neg P(f(c))}}{\bot}$$

3.9 Propositional Resolution

We observe that propositional clauses and ground clauses are the same concept.

In this section we only deal with ground clauses.

The Resolution Calculus *Res*

Resolution inference rule:

$$\frac{D \lor A \qquad \neg A \lor C}{D \lor C}$$

Terminology: $D \lor C$: *resolvent*; A: *resolved atom*

(Positive) factorisation inference rule:

$$\frac{C \lor A \lor A}{C \lor A}$$

These are *schematic inference rules*; for each substitution of the *schematic variables* C, D, and A, respectively, by ground clauses and ground atoms we obtain an inference rule.

As "$\lor$" is considered associative and commutative, we assume that A and $\neg A$ can occur anywhere in their respective clauses.

Sample Refutation

1.	$\neg P(f(c)) \vee \neg P(f(c)) \vee Q(b)$	(given)
2.	$P(f(c)) \vee Q(b)$	(given)
3.	$\neg P(g(b,c)) \vee \neg Q(b)$	(given)
4.	$P(g(b,c))$	(given)
5.	$\neg P(f(c)) \vee Q(b) \vee Q(b)$	(Res. 2. into 1.)
6.	$\neg P(f(c)) \vee Q(b)$	(Fact. 5.)
7.	$Q(b) \vee Q(b)$	(Res. 2. into 6.)
8.	$Q(b)$	(Fact. 7.)
9.	$\neg P(g(b,c))$	(Res. 8. into 3.)
10.	$\perp$	(Res. 4. into 9.)

Resolution with Implicit Factorization RIF

$$\frac{D \vee A \vee \ldots \vee A \qquad \neg A \vee C}{D \vee C}$$

1.	$\neg P(f(c)) \vee \neg P(f(c)) \vee Q(b)$	(given)
2.	$P(f(c)) \vee Q(b)$	(given)
3.	$\neg P(g(b,c)) \vee \neg Q(b)$	(given)
4.	$P(g(b,c))$	(given)
5.	$\neg P(f(c)) \vee Q(b) \vee Q(b)$	(Res. 2. into 1.)
6.	$Q(b) \vee Q(b) \vee Q(b)$	(Res. 2. into 5.)
7.	$\neg P(g(b,c))$	(Res. 6. into 3.)
8.	$\perp$	(Res. 4. into 7.)

Soundness of Resolution

Theorem 3.15 *Propositional resolution is sound.*

Proof. Let $I \in \Sigma\text{-}\mathsf{Alg}$. To be shown:

(i) for resolution: $I \models D \vee A,\ I \models C \vee \neg A \ \Rightarrow\ I \models D \vee C$

(ii) for factorization: $I \models C \vee A \vee A \ \Rightarrow\ I \models C \vee A$

(i): Assume premises are valid in I. Two cases need to be considered:
If $I \models A$, then $I \models C$, hence $I \models D \vee C$.
Otherwise, $I \models \neg A$, then $I \models D$, and again $I \models D \vee C$.
(ii): even simpler. □

Note: In propositional logic (ground clauses) we have:

1. $I \models L_1 \vee \ldots \vee L_n \;\Leftrightarrow\;$ there exists i: $I \models L_i$.
2. $I \models A$ or $I \models \neg A$.

This does not hold for formulas with variables!

3.10 Well-Founded Orderings

Literature: Franz Baader and Tobias Nipkow: *Term rewriting and all that*, Cambridge Univ. Press, 1998, Chapter 2.

To show the refutational completeness of resolution, we will make use of the concept of well-founded orderings.

Partial Orderings

A *(strict) partial ordering* $\succ$ on a set M is a transitive and irreflexive binary relation on M.

An $a \in M$ is called *minimal*, if there is no b in M such that $a \succ b$.

An $a \in M$ is called *smallest*, if $b \succ a$ for all $b \in M$ different from a.

Notation:
$\prec$ for the inverse relation $\succ^{-1}$
$\succeq$ for the reflexive closure $(\succ \cup =)$ of $\succ$

Well-Foundedness

A (strict) partial ordering $\succ$ is called *well-founded (Noetherian)*, if there is no infinite descending chain $a_0 \succ a_1 \succ a_2 \succ \dots$ with $a_i \in M$.

Well-Founded Orderings: Examples

Natural numbers. $(\mathbb{N}, >)$

Lexicographic orderings. Let $(M_1, \succ_1), (M_2, \succ_2)$ be well-founded orderings. Then let their *lexicographic combination*

$$\succ = (\succ_1, \succ_2)_{lex}$$

on $M_1 \times M_2$ be defined as

$$(a_1, a_2) \succ (b_1, b_2) \quad :\Leftrightarrow \quad a_1 \succ_1 b_1, \text{ or else } a_1 = b_1 \;\&\; a_2 \succ_2 b_2$$

(analogously for more than two orderings)

This again yields a well-founded ordering (proof below).

Length-based ordering on words. For alphabets Σ with a well-founded ordering $>_\Sigma$, the relation $\succ$, defined as

$$w \succ w' \quad := \quad \alpha)\ |w| > |w'| \text{ or}$$
$$\beta)\ |w| = |w'| \text{ and } w >_{\Sigma,lex} w',$$

is a well-founded ordering on Σ^* (proof below).

Counterexamples:

$(\mathbb{Z}, >)$;
$(\mathbb{N}, <)$;
the lexicographic ordering on Σ^*

Basic Properties of Well-Founded Orderings

Lemma 3.16 *$(M, \succ)$ is well-founded if and only if every $\emptyset \subset M' \subseteq M$ has a minimal element.*

Lemma 3.17 *$(M_i, \succ_i)$ is well-founded for $i = 1, 2$ if and only if $(M_1 \times M_2,\ \succ)$ with $\succ = (\succ_1, \succ_2)_{lex}$ is well-founded.*

Proof. (i) "$\Rightarrow$": Suppose $(M_1 \times M_2,\ \succ)$ is not well-founded. Then there is an infinite sequence $(a_0, b_0) \succ (a_1, b_1) \succ (a_2, b_2) \succ \ldots$.

Let $A = \{a_i \mid i \geq 0\} \subseteq M_1$. Since $(M_1, \succ_1)$ is well-founded, A has a minimal element a_n. But then $B = \{b_i \mid i \geq n\} \subseteq M_2$ can not have a minimal element, contradicting the well-foundedness of $(M_2, \succ_2)$.

(ii) "$\Leftarrow$": obvious. $\square$

Noetherian Induction

Theorem 3.18 (Noetherian Induction) *Let $(M, \succ)$ be a well-founded ordering, let Q be a property of elements of M.*

If for all $m \in M$ the implication

if $Q(m')$, for all $m' \in M$ such that $m \succ m'$,[1]
then $Q(m)$.[2]

is satisfied, then the property $Q(m)$ holds for all $m \in M$.

[1] induction hypothesis
[2] induction step

Proof. Let $X = \{m \in M \mid Q(m) \text{ false}\}$. Suppose, $X \neq \emptyset$. Since $(M, \succ)$ is well-founded, X has a minimal element m_1. Hence for all $m' \in M$ with $m' \prec m_1$ the property $Q(m')$ holds. On the other hand, the implication which is presupposed for this theorem holds in particular also for m_1, hence $Q(m_1)$ must be true so that m_1 can not be in X. *Contradiction.* □

Multi-Sets

Let M be a set. A *multi-set* S over M is a mapping $S : M \to \mathbb{N}$. Hereby $S(m)$ specifies the number of occurrences of elements m of the base set M within the multi-set S.

We say that m is an *element* of S, if $S(m) > 0$.

We use set notation ($\in$, $\subset$, $\subseteq$, $\cup$, $\cap$, etc.) with analogous meaning also for multi-sets, e. g.,

$$\begin{aligned}
(S_1 \cup S_2)(m) &= S_1(m) + S_2(m) \\
(S_1 \cap S_2)(m) &= \min\{S_1(m), S_2(m)\}
\end{aligned}$$

A multi-set is called *finite*, if

$$|\{m \in M|\ s(m) > 0\}| < \infty,$$

for each m in M.

From now on we only consider finite multi-sets.

Example. $S = \{a, a, a, b, b\}$ is a multi-set over $\{a, b, c\}$, where $S(a) = 3$, $S(b) = 2$, $S(c) = 0$.

Multi-Set Orderings

Lemma 3.19 (König's Lemma) *Every finitely branching tree with infinitely many nodes contains an infinite path.*

Let $(M, \succ)$ be a partial ordering. The *multi-set extension* of $\succ$ to multi-sets over M is defined by

$$\begin{aligned}
S_1 \succ_{\text{mul}} S_2 :\Leftrightarrow\ & S_1 \neq S_2 \\
& \text{and } \forall m \in M : [S_2(m) > S_1(m) \\
& \qquad \Rightarrow \quad \exists m' \in M : (m' \succ m \text{ and } S_1(m') > S_2(m'))]
\end{aligned}$$

Theorem 3.20
(a) $\succ_{\text{mul}}$ *is a partial ordering.*
(b) $\succ$ *well-founded* $\Rightarrow$ $\succ_{\text{mul}}$ *well-founded.*
(c) $\succ$ *total* $\Rightarrow$ $\succ_{\text{mul}}$ *total.*

Proof. see Baader and Nipkow, page 22–24. □

Proof of DPLL Termination: Lemma 1.10

Proof. (Idea) Consider a DPLL derivation step $M \parallel N \Rightarrow_{\text{DPLL}} M' \parallel N'$ and a decomposition $M_0 l_1^d M_1 \dots l_k^d M_k$ of M (accordingly for M'). Let n be the number of distinct propositional variables in N. Then k, k' and the length of M, M' are always smaller than n. We define $f(M) = n - \text{length}(M)$ and finally

$$M \parallel N \succ M' \parallel N' \quad \text{if}$$

(i) $f(M_0) = f(M'_0), \dots, f(M_{i-1}) = f(M'_{i-1}), f(M_i) > f(M'_i)$ for some $i < k, k'$ or

(ii) $f(M_j) = f(M'_j)$ for all $1 \leq j \leq k$ and $f(M) > f(M')$.

3.11 Refutational Completeness of Resolution

How to show refutational completeness of propositional resolution:

- We have to show: $N \models \bot \Rightarrow N \vdash_{Res} \bot$, or equivalently: If $N \not\vdash_{Res} \bot$, then N has a model.
- Idea: Suppose that we have computed sufficiently many inferences (and not derived $\bot$).
- Now order the clauses in N according to some appropriate ordering, inspect the clauses in ascending order, and construct a series of Herbrand interpretations.
- The limit interpretation can be shown to be a model of N.

Clause Orderings

1. We assume that $\succ$ is any fixed ordering on ground atoms that is *total* and *well-founded*. (There exist many such orderings, e. g., the lenght-based ordering on atoms when these are viewed as words over a suitable alphabet.)
2. Extend $\succ$ to an *ordering* $\succ_L$ *on ground literals:*

$$\begin{array}{lcll} [\neg]A & \succ_L & [\neg]B & \text{, if } A \succ B \\ \neg A & \succ_L & A & \end{array}$$

3. Extend $\succ_L$ to an *ordering* $\succ_C$ *on ground clauses:*
   $\succ_C = (\succ_L)_{\text{mul}}$, the multi-set extension of $\succ_L$.

   Notation: $\succ$ also for $\succ_L$ and $\succ_C$.

Example

Suppose $A_5 \succ A_4 \succ A_3 \succ A_2 \succ A_1 \succ A_0$. Then:

$$\begin{array}{cc} & A_0 \vee A_1 \\ \prec & A_1 \vee A_2 \\ \prec & \neg A_1 \vee A_2 \\ \prec & \neg A_1 \vee A_4 \vee A_3 \\ \prec & \neg A_1 \vee \neg A_4 \vee A_3 \\ \prec & \neg A_5 \vee A_5 \end{array}$$

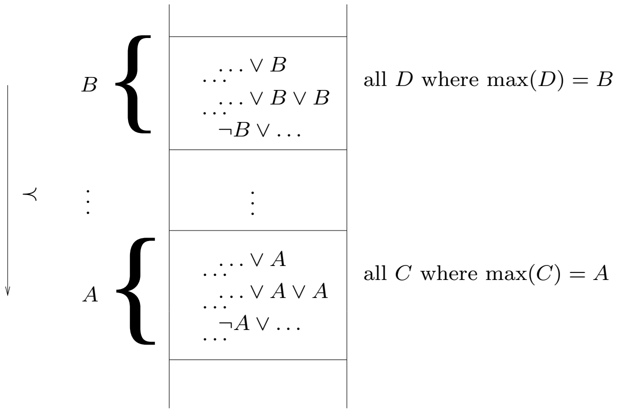
Properties of the Clause Ordering

Proposition 3.21

1. *The orderings on literals and clauses are total and well-founded.*
2. *Let C and D be clauses with $A = \max(C)$, $B = \max(D)$, where $\max(C)$ denotes the maximal atom in C.*
   (i) If $A \succ B$ then $C \succ D$.
   (ii) If $A = B$, A occurs negatively in C but only positively in D, then $C \succ D$.

Stratified Structure of Clause Sets

Let $A \succ B$. Clause sets are then stratified in this form:



Closure of Clause Sets under Res

$$\begin{aligned}
Res(N) &= \{C \mid C \text{ is concl. of a rule in } Res \text{ w/ premises in } N\} \\
Res^0(N) &= N \\
Res^{n+1}(N) &= Res(Res^n(N)) \cup Res^n(N), \text{ for } n \geq 0 \\
Res^*(N) &= \textstyle\bigcup_{n\geq 0} Res^n(N)
\end{aligned}$$

N is called *saturated* (w. r. t. resolution), if $Res(N) \subseteq N$.

Proposition 3.22

(i) $Res^*(N)$ is saturated.

(ii) Res is refutationally complete, iff for each set N of ground clauses:

$$N \models \bot \;\Leftrightarrow\; \bot \in Res^*(N)$$

Construction of Interpretations

Given: set N of ground clauses, atom ordering $\succ$.
Wanted: Herbrand interpretation I such that

- "many" clauses from N are valid in I;
- *$I \models N$, if N is saturated and $\bot \notin N$.*

Construction according to $\succ$, starting with the minimal clause.

Example

Let $A_5 \succ A_4 \succ A_3 \succ A_2 \succ A_1 \succ A_0$ (max. literals in red)

	clauses C	I_C	Δ_C	Remarks
1	$\neg A_0$	$\emptyset$	$\emptyset$	true in I_C
2	$A_0 \vee A_1$	$\emptyset$	$\{A_1\}$	A_1 maximal
3	$A_1 \vee A_2$	$\{A_1\}$	$\emptyset$	true in I_C
4	$\neg A_1 \vee A_2$	$\{A_1\}$	$\{A_2\}$	A_2 maximal
5	$\neg A_1 \vee A_4 \vee A_3 \vee A_0$	$\{A_1, A_2\}$	$\{A_4\}$	A_4 maximal
6	$\neg A_1 \vee \neg A_4 \vee A_3$	$\{A_1, A_2, A_4\}$	$\emptyset$	A_3 not maximal; *min. counter-ex.*
7	$\neg A_1 \vee A_5$	$\{A_1, A_2, A_4\}$	$\{A_5\}$	

$I = \{A_1, A_2, A_4, A_5\}$ is not a model of the clause set
$\Rightarrow$ there exists a *counterexample*.

Main Ideas of the Construction

- Clauses are considered in the order given by $\prec$.
- When considering C, one already has a partial interpretation I_C (initially $I_C = \emptyset$) available.
- If C is true in the partial interpretation I_C, nothing is done. ($\Delta_C = \emptyset$).
- If C is false, one would like to change I_C such that C becomes true.
- Changes should, however, be *monotone*. One never deletes anything from I_C and the truth value of clauses smaller than C should be maintained the way it was in I_C.
- Hence, one chooses $\Delta_C = \{A\}$ if, and only if, C is false in I_C, if A occurs positively in C (*adding A will make C become true*) and if this occurrence in C is strictly maximal in the ordering on literals (*changing the truth value of A has no effect on smaller clauses*).

Resolution Reduces Counterexamples

$$\frac{\neg A_1 \vee A_4 \vee A_3 \vee A_0 \qquad \neg A_1 \vee \neg A_4 \vee A_3}{\neg A_1 \vee \neg A_1 \vee A_3 \vee A_3 \vee A_0}$$

Construction of I for the extended clause set:

clauses C	I_C	Δ_C	Remarks
$\neg A_0$	$\emptyset$	$\emptyset$	
$A_0 \vee A_1$	$\emptyset$	$\{A_1\}$	
$A_1 \vee A_2$	$\{A_1\}$	$\emptyset$	
$\neg A_1 \vee A_2$	$\{A_1\}$	$\{A_2\}$	
$\neg A_1 \vee \neg A_1 \vee A_3 \vee A_3 \vee A_0$	$\{A_1, A_2\}$	$\emptyset$	A_3 occurs twice *minimal counter-ex.*
$\neg A_1 \vee A_4 \vee A_3 \vee A_0$	$\{A_1, A_2\}$	$\{A_4\}$	
$\neg A_1 \vee \neg A_4 \vee A_3$	$\{A_1, A_2, A_4\}$	$\emptyset$	counterexample
$\neg A_1 \vee A_5$	$\{A_1, A_2, A_4\}$	$\{A_5\}$	

The same I, but smaller counterexample, hence some progress was made.

Factorization Reduces Counterexamples

$$\frac{\neg A_1 \vee \neg A_1 \vee A_3 \vee A_3 \vee A_0}{\neg A_1 \vee \neg A_1 \vee A_3 \vee A_0}$$

Construction of I for the extended clause set:

clauses C	I_C	Δ_C	Remarks
$\neg A_0$	$\emptyset$	$\emptyset$	
$A_0 \vee A_1$	$\emptyset$	$\{A_1\}$	
$A_1 \vee A_2$	$\{A_1\}$	$\emptyset$	
$\neg A_1 \vee A_2$	$\{A_1\}$	$\{A_2\}$	
$\neg A_1 \vee \neg A_1 \vee A_3 \vee A_0$	$\{A_1, A_2\}$	$\{A_3\}$	
$\neg A_1 \vee \neg A_1 \vee A_3 \vee A_3 \vee A_0$	$\{A_1, A_2, A_3\}$	$\emptyset$	true in I_C
$\neg A_1 \vee A_4 \vee A_3 \vee A_0$	$\{A_1, A_2, A_3\}$	$\emptyset$	
$\neg A_1 \vee \neg A_4 \vee A_3$	$\{A_1, A_2, A_3\}$	$\emptyset$	true in I_C
$\neg A_3 \vee A_5$	$\{A_1, A_2, A_3\}$	$\{A_5\}$	

The resulting $I = \{A_1, A_2, A_3, A_5\}$ is a model of the clause set.

Construction of Candidate Interpretations

Let $N, \succ$ be given. We define sets I_C and Δ_C for all ground clauses C over the given signature inductively over $\succ$:

$$I_C \quad := \quad \textstyle\bigcup_{C \succ D} \Delta_D$$

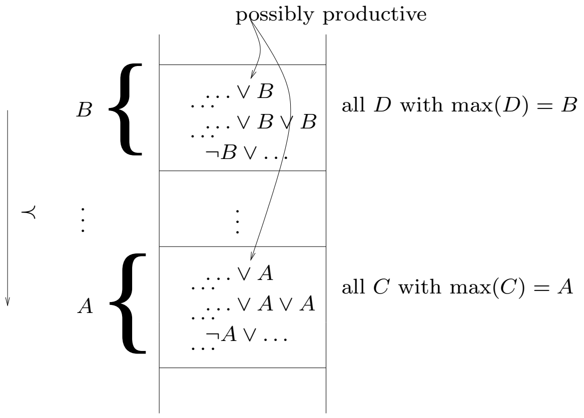
$$\Delta_C \quad := \quad \begin{cases} \{A\}, & \text{if } C \in N,\ C = C' \vee A,\ A \succ C',\ I_C \not\models C \\ \emptyset, & \text{otherwise} \end{cases}$$

We say that C *produces* A, if $\Delta_C = \{A\}$.

The *candidate interpretation* for N (w. r. t. $\succ$) is given as $I_N^{\succ} := \bigcup_C \Delta_C$. (We also simply write I_N or I for $I_N^{\succ}$ if $\succ$ is either irrelevant or known from the context.)

Structure of $N, \succ$

Let $A \succ B$; producing a new atom does not affect smaller clauses.



Some Properties of the Construction

Proposition 3.23

(i) $C = \neg A \vee C' \;\Rightarrow\;$ *no* $D \succeq C$ *produces* A.

(ii) C *productive* $\Rightarrow I_C \cup \Delta_C \models C$.

(iii) *Let* $D' \succ D \succeq C$. *Then*

$$I_D \cup \Delta_D \models C \Rightarrow I_{D'} \cup \Delta_{D'} \models C \text{ and } I_N \models C.$$

If, in addition, $C \in N$ *or* $\max(D) \succ \max(C)$*:*

$$I_D \cup \Delta_D \not\models C \Rightarrow I_{D'} \cup \Delta_{D'} \not\models C \text{ and } I_N \not\models C.$$

(iv) *Let* $D' \succ D \succ C$. *Then*

$$I_D \models C \Rightarrow I_{D'} \models C \text{ and } I_N \models C.$$

If, in addition, $C \in N$ *or* $\max(D) \succ \max(C)$*:*

$$I_D \not\models C \Rightarrow I_{D'} \not\models C \text{ and } I_N \not\models C.$$

(v) $D = C \vee A$ *produces* $A \Rightarrow I_N \not\models C$.

Model Existence Theorem

Theorem 3.24 (Bachmair & Ganzinger 1990) *Let $\succ$ be a clause ordering, let N be saturated w. r. t. Res, and suppose that $\bot \notin N$. Then $I_N^\succ \models N$.*

Corollary 3.25 *Let N be saturated w. r. t. Res. Then $N \models \bot \Leftrightarrow \bot \in N$.*

Proof of Theorem 3.24. Suppose $\bot \notin N$, but $I_N^\succ \not\models N$. Let $C \in N$ minimal (in $\succ$) such that $I_N^\succ \not\models C$. Since C is false in I_N, C is not productive. As $C \neq \bot$ there exists a maximal atom A in C.

Case 1: $C = \neg A \vee C'$ (i. e., the maximal atom occurs negatively)
$\Rightarrow I_N \models A$ and $I_N \not\models C'$
$\Rightarrow$ some $D = D' \vee A \in N$ produces A. As $\frac{D' \vee A \qquad \neg A \vee C'}{D' \vee C'}$, we infer that $D' \vee C' \in N$, and $C \succ D' \vee C'$ and $I_N \not\models D' \vee C'$
$\Rightarrow$ contradicts minimality of C.

Case 2: $C = C' \vee A \vee A$. Then $\frac{C' \vee A \vee A}{C' \vee A}$ yields a smaller counterexample $C' \vee A \in N$. $\Rightarrow$ contradicts minimality of C. $\square$

Compactness of Propositional Logic

Theorem 3.26 (Compactness) *Let N be a set of propositional formulas. Then N is unsatisfiable, if and only if some finite subset $M \subseteq N$ is unsatisfiable.*

Proof. "$\Leftarrow$": trivial.

"$\Rightarrow$": Let N be unsatisfiable.
$\Rightarrow$ $Res^*(N)$ unsatisfiable
$\Rightarrow \bot \in Res^*(N)$ by refutational completeness of resolution
$\Rightarrow \exists n \geq 0 : \bot \in Res^n(N)$
$\Rightarrow \bot$ has a finite resolution proof P;
choose M as the set of assumptions in P. $\square$

3.12 General Resolution

Propositional resolution:

refutationally complete,

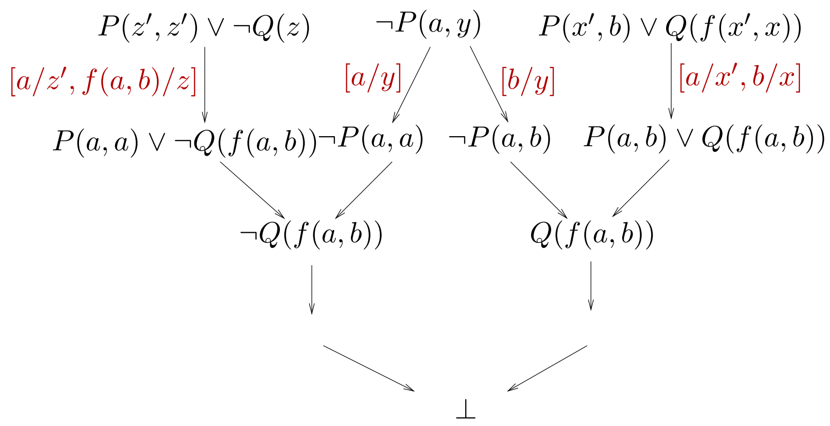
in its most naive version: not guaranteed to terminate for satisfiable sets of clauses, (improved versions do terminate, however)

in practice clearly inferior to the DPLL procedure (even with various improvements).

But: in contrast to the DPLL procedure, resolution can be easily extended to non-ground clauses.

General Resolution through Instantiation

Idea: instantiate clauses appropriately:



Problems:

More than one instance of a clause can participate in a proof.

Even worse: There are infinitely many possible instances.

Observation:

Instantiation must produce complementary literals (so that inferences become possible).

Idea:

Do not instantiate more than necessary to get complementary literals.

Idea: do not instantiate more than necessary:

$$
\begin{tikzcd}
P(z',z') \lor \neg Q(z) \arrow[d, "{[a/z']}"'] & & & & \neg P(a,y) \arrow[dll, "{[a/y]}"'] \arrow[drr, "{[b/y]}"] & & & & P(x',b) \lor Q(f(x',x)) \arrow[d, "{[a/x']}"] \\
P(a,a) \lor \neg Q(z) \arrow[dr] & & \neg P(a,a) \arrow[dl] & & & & \neg P(a,b) \arrow[dr] & & P(a,b) \lor Q(f(a,x)) \arrow[dl] \\
& \neg Q(z) \arrow[d, "{[f(a,x)/z]}"'] & & & & & & Q(f(a,x)) \arrow[d] & \\
& \neg Q(f(a,x)) \arrow[drrr] & & & & & & Q(f(a,x)) \arrow[dlll] & \\
& & & & \bot & & & &
\end{tikzcd}
$$

Lifting Principle

Problem: Make saturation of infinite sets of clauses as they arise from taking the (ground) instances of finitely many *general* clauses (with variables) effective and efficient.

Idea (Robinson 1965):

- Resolution for general clauses:
- *Equality* of ground atoms is generalized to *unifiability* of general atoms;
- Only compute *most general* (minimal) unifiers.

Significance: The advantage of the method in (Robinson 1965) compared with (Gilmore 1960) is that unification enumerates only those instances of clauses that participate in an inference. Moreover, clauses are not right away instantiated into ground clauses. Rather they are instantiated only as far as required for an inference. Inferences with non-ground clauses in general represent infinite sets of ground inferences which are computed simultaneously in a single step.

Resolution for General Clauses

General binary resolution *Res*:

$$\frac{D \lor B \qquad C \lor \neg A}{(D \lor C)\sigma} \quad \text{if } \sigma = \mathrm{mgu}(A, B) \qquad \text{[resolution]}$$

$$\frac{C \lor A \lor B}{(C \lor A)\sigma} \quad \text{if } \sigma = \mathrm{mgu}(A, B) \quad \text{[factorization]}$$

General resolution RIF with implicit factorization:

$$\frac{D \vee B_1 \vee \ldots \vee B_n \qquad C \vee \neg A}{(D \vee C)\sigma} \quad \text{if } \sigma = \text{mgu}(A, B_1, \ldots, B_n) \qquad \text{[RIF]}$$

For inferences with more than one premise, we assume that the variables in the premises are (bijectively) renamed such that they become different to any variable in the other premises. We do not formalize this. Which names one uses for variables is otherwise irrelevant.

Unification

Let $E = \{s_1 \doteq t_1, \ldots, s_n \doteq t_n\}$ (s_i, t_i terms or atoms) a multi-set of *equality problems.* A substitution σ is called a *unifier* of E if $s_i\sigma = t_i\sigma$ for all $1 \leq i \leq n$.

If a unifier of E exists, then E is called *unifiable.*

A substitution σ is called *more general* than a substitution τ, denoted by $\sigma \leq \tau$, if there exists a substitution ρ such that $\rho \circ \sigma = \tau$, where $(\rho \circ \sigma)(x) := (x\sigma)\rho$ is the composition of σ and ρ as mappings. (Note that $\rho \circ \sigma$ has a finite domain as required for a substitution.)

If a unifier of E is more general than any other unifier of E, then we speak of a *most general unifier* of E, denoted by mgu(E).

Proposition 3.27

(i) $\leq$ *is a quasi-ordering on substitutions, and* $\circ$ *is associative.*

(ii) If $\sigma \leq \tau$ *and* $\tau \leq \sigma$ *(we write* $\sigma \sim \tau$ *in this case), then* $x\sigma$ *and* $x\tau$ *are equal up to (bijective) variable renaming, for any* x *in* X*.*

A substitution σ is called *idempotent,* if $\sigma \circ \sigma = \sigma$.

Proposition 3.28 *σ is idempotent iff* $dom(\sigma) \cap codom(\sigma) = \emptyset$*.*

Rule Based Naive Standard Unification

$$
\begin{aligned}
t \doteq t, E &\;\Rightarrow_{SU}\; E \\
f(s_1, \ldots, s_n) \doteq f(t_1, \ldots, t_n), E &\;\Rightarrow_{SU}\; s_1 \doteq t_1, \ldots, s_n \doteq t_n, E \\
f(\ldots) \doteq g(\ldots), E &\;\Rightarrow_{SU}\; \bot \\
x \doteq t, E &\;\Rightarrow_{SU}\; x \doteq t, E[t/x] \\
&\;\phantom{\Rightarrow_{SU}}\; \text{if } x \in var(E), x \notin var(t) \\
x \doteq t, E &\;\Rightarrow_{SU}\; \bot \\
&\;\phantom{\Rightarrow_{SU}}\; \text{if } x \neq t, x \in var(t) \\
t \doteq x, E &\;\Rightarrow_{SU}\; x \doteq t, E \\
&\;\phantom{\Rightarrow_{SU}}\; \text{if } t \notin X
\end{aligned}
$$

SU: Main Properties

If $E = x_1 \doteq u_1, \ldots, x_k \doteq u_k$, with x_i pairwise distinct, $x_i \notin var(u_j)$, then E is called an (equational problem in) *solved form* representing the solution $\sigma_E = [u_1/x_1, \ldots, u_k/x_k]$.

Proposition 3.29 *If E is a solved form then σ_E is an mgu of E.*

Theorem 3.30

1. *If $E \Rightarrow_{SU} E'$ then σ is a unifier of E iff σ is a unifier of E'*
2. *If $E \Rightarrow^*_{SU} \bot$ then E is not unifiable.*
3. *If $E \Rightarrow^*_{SU} E'$ with E' in solved form, then $\sigma_{E'}$ is an mgu of E.*

Proof. (1) We have to show this for each of the rules. Let's treat the case for the 4th rule here. Suppose σ is a unifier of $x \doteq t$, that is, $x\sigma = t\sigma$. Thus, $\sigma \circ [t/x] = \sigma[x \mapsto t\sigma] = \sigma[x \mapsto x\sigma] = \sigma$. Therefore, for any equation $u \doteq v$ in E: $u\sigma = v\sigma$, iff $u[t/x]\sigma = v[t/x]\sigma$. (2) and (3) follow by induction from (1) using Proposition 3.29. □

Main Unification Theorem

Theorem 3.31 *E is unifiable if and only if there is a most general unifier σ of E, such that σ is idempotent and $dom(\sigma) \cup codom(\sigma) \subseteq var(E)$.*

Problem: *exponential growth* of terms possible

Proof of Theorem 3.31.

- $\Rightarrow_{SU}$ is Noetherian. A suitable lexicographic ordering on the multisets E (with $\bot$ minimal) shows this. Compare in this order:

1. the number of defined variables (d.h. variables x in equations $x \doteq t$ with $x \notin var(t)$), which also occur outside their definition elsewhere in E;
2. the multi-set ordering induced by (i) the size (number of symbols) in an equation; (ii) if sizes are equal consider $x \doteq t$ smaller than $t \doteq x$, if $t \notin X$. $\square$

- A system E that is irreducible w. r. t. $\Rightarrow_{SU}$ is either $\perp$ or a solved form.
- Therefore, reducing any E by SU will end (no matter what reduction strategy we apply) in an irreducible E' having the same unifiers as E, and we can read off the mgu (or non-unifiability) of E from E' (Theorem 3.30, Proposition 3.29).
- σ is idempotent because of the substitution in rule 4. $dom(\sigma) \cup codom(\sigma) \subseteq var(E)$, as no new variables are generated.

Rule Based Polynomial Unification

$$
\begin{aligned}
t \doteq t, E &\Rightarrow_{PU} E \\
f(s_1, \ldots, s_n) \doteq f(t_1, \ldots, t_n), E &\Rightarrow_{PU} s_1 \doteq t_1, \ldots, s_n \doteq t_n, E \\
f(\ldots) \doteq g(\ldots), E &\Rightarrow_{PU} \perp \\
x \doteq y, E &\Rightarrow_{PU} x \doteq y, E[y/x] \\
&\quad \text{if } x \in var(E), x \neq y \\
x_1 \doteq t_1, \ldots, x_n \doteq t_n, E &\Rightarrow_{PU} \perp
\end{aligned}
$$

if there are positions p_i with $t_i/p_i = x_{i+1}, t_n/p_n = x_1$ and some $p_i \neq \epsilon$

$$
\begin{aligned}
x \doteq t, E &\Rightarrow_{PU} \perp \\
&\quad \text{if } x \neq t, x \in var(t) \\
t \doteq x, E &\Rightarrow_{PU} x \doteq t, E \\
&\quad \text{if } t \notin X \\
x \doteq t, x \doteq s, E &\Rightarrow_{PU} x \doteq t, t \doteq s, E \\
&\quad \text{if } t, s \notin X \text{ and } |t| \leq |s|
\end{aligned}
$$

Properties of PU

Theorem 3.32

1. *If $E \Rightarrow_{PU} E'$ then σ is a unifier of E iff σ is a unifier of E'*
2. *If $E \Rightarrow^*_{PU} \perp$ then E is not unifiable.*
3. *If $E \Rightarrow^*_{PU} E'$ with E' in solved form, then $\sigma_{E'}$ is an mgu of E.*

The solved form of $\Rightarrow_{PU}$ is different form the solved form obtained from $\Rightarrow_{SU}$. In order to obtain a unifier, the substitutions generated by the single equations have to be composed.

Lifting Lemma

Lemma 3.33 *Let C and D be variable-disjoint clauses. If*

$$\begin{array}{c} \begin{array}{cc} D & C \\ \downarrow \sigma & \downarrow \rho \\ D\sigma & C\rho \end{array} \\ \hline C' \end{array} \qquad \textit{[propositional resolution]}$$

then there exists a substitution τ such that

$$\begin{array}{c} \dfrac{D \qquad C}{C''} \\ \downarrow \tau \\ C' = C''\tau \end{array} \qquad \textit{[general resolution]}$$

An analogous lifting lemma holds for factorization.

Saturation of Sets of General Clauses

Corollary 3.34 *Let N be a set of general clauses saturated under Res, i. e., $Res(N) \subseteq N$. Then also $G_\Sigma(N)$ is saturated, that is,*

$$Res(G_\Sigma(N)) \subseteq G_\Sigma(N).$$

Proof. W.l.o.g. we may assume that clauses in N are pairwise variable-disjoint. (Otherwise make them disjoint, and this renaming process changes neither $Res(N)$ nor $G_\Sigma(N)$.)

Let $C' \in Res(G_\Sigma(N))$, meaning (i) there exist resolvable ground instances $D\sigma$ and $C\rho$ of N with resolvent C', or else (ii) C' is a factor of a ground instance $C\sigma$ of C.

Case (i): By the Lifting Lemma, D and C are resolvable with a resolvent C'' with $C''\tau = C'$, for a suitable substitution τ. As $C'' \in N$ by assumption, we obtain that $C' \in G_\Sigma(N)$.

Case (ii): Similar. $\square$

Herbrand's Theorem

Lemma 3.35 *Let N be a set of Σ-clauses, let $\mathcal{A}$ be an interpretation. Then $\mathcal{A} \models N$ implies $\mathcal{A} \models G_\Sigma(N)$.*

Lemma 3.36 *Let N be a set of Σ-clauses, let $\mathcal{A}$ be a* Herbrand *interpretation. Then $\mathcal{A} \models G_\Sigma(N)$ implies $\mathcal{A} \models N$.*

Theorem 3.37 (Herbrand) *A set N of Σ-clauses is satisfiable if and only if it has a Herbrand model over Σ.*

Proof. The "$\Leftarrow$" part is trivial. For the "$\Rightarrow$" part let $N \not\models \bot$.

$$\begin{aligned}
N \not\models \bot &\Rightarrow \bot \notin \mathit{Res}^*(N) \qquad \text{(resolution is sound)}\\
&\Rightarrow \bot \notin G_\Sigma(\mathit{Res}^*(N))\\
&\Rightarrow I_{G_\Sigma(\mathit{Res}^*(N))} \models G_\Sigma(\mathit{Res}^*(N)) \qquad \text{(Thm. 3.24; Cor. 3.34)}\\
&\Rightarrow I_{G_\Sigma(\mathit{Res}^*(N))} \models \mathit{Res}^*(N) \qquad \text{(Lemma 3.36)}\\
&\Rightarrow I_{G_\Sigma(\mathit{Res}^*(N))} \models N \qquad (N \subseteq \mathit{Res}^*(N)) \quad \Box
\end{aligned}$$

The Theorem of Löwenheim-Skolem

Theorem 3.38 (Löwenheim–Skolem) *Let Σ be a countable signature and let S be a set of closed Σ-formulas. Then S is satisfiable iff S has a model over a countable universe.*

Proof. If both X and Σ are countable, then S can be at most countably infinite. Now generate, maintaining satisfiability, a set N of clauses from S. This extends Σ by at most countably many new Skolem functions to Σ'. As Σ' is countable, so is $T_{\Sigma'}$, the universe of Herbrand-interpretations over Σ'. Now apply Theorem 3.37. $\Box$

Refutational Completeness of General Resolution

Theorem 3.39 *Let N be a set of general clauses where $\mathit{Res}(N) \subseteq N$. Then*

$$N \models \bot \Leftrightarrow \bot \in N.$$

Proof. Let $\mathit{Res}(N) \subseteq N$. By Corollary 3.34: $\mathit{Res}(G_\Sigma(N)) \subseteq G_\Sigma(N)$

$$\begin{aligned}
N \models \bot &\Leftrightarrow G_\Sigma(N) \models \bot \qquad \text{(Lemma 3.35/3.36; Theorem 3.37)}\\
&\Leftrightarrow \bot \in G_\Sigma(N) \quad \text{(propositional resolution sound and complete)}\\
&\Leftrightarrow \bot \in N \quad \Box
\end{aligned}$$

Compactness of Predicate Logic

Theorem 3.40 (Compactness Theorem for First-Order Logic) *Let Φ be a set of first-order formulas. Φ is unsatisfiable $\Leftrightarrow$ some finite subset $\Psi \subseteq \Phi$ is unsatisfiable.*

Proof. The "$\Leftarrow$" part is trivial. For the "$\Rightarrow$" part let Φ be unsatisfiable and let N be the set of clauses obtained by Skolemization and CNF transformation of the formulas in Φ. Clearly $Res^*(N)$ is unsatisfiable. By Theorem 3.39, $\bot \in Res^*(N)$, and therefore $\bot \in Res^n(N)$ for some $n \in \mathbb{N}$. Consequently, $\bot$ has a finite resolution proof B of depth $\leq n$. Choose Ψ as the subset of formulas in Φ such that the corresponding clauses contain the assumptions (leaves) of B. $\square$

3.13 Ordered Resolution with Selection

Motivation: Search space for *Res very* large.

Ideas for improvement:

1. In the completeness proof (Model Existence Theorem 3.24) one only needs to resolve and factor maximal atoms
$\Rightarrow$ if the calculus is restricted to inferences involving maximal atoms, the proof remains correct
$\Rightarrow$ *order restrictions*
2. In the proof, it does not really matter with which negative literal an inference is performed
$\Rightarrow$ choose a negative literal don't-care-nondeterministically
$\Rightarrow$ *selection*

Selection Functions

A *selection function* is a mapping

$$S : C \quad \mapsto \quad \text{set of occurrences of } \textit{negative} \text{ literals in } C$$

Example of selection with selected literals indicated as $\boxed{X}$:

$$\boxed{\neg A} \vee \neg A \vee B$$

$$\boxed{\neg B_0} \vee \boxed{\neg B_1} \vee A$$

Resolution Calculus $Res_S^{\succ}$

In the completeness proof, we talk about (strictly) maximal literals of *ground* clauses.

In the non-ground calculus, we have to consider those literals that correspond to (strictly) maximal literals of ground instances:

Let $\succ$ be a total and well-founded ordering on ground atoms. A literal L is called *[strictly] maximal* in a clause C if and only if there exists a ground substitution σ such that for no other L' in C: $L\sigma \prec L'\sigma$ $[L\sigma \preceq L'\sigma]$.

Let $\succ$ be an atom ordering and S a selection function.

$$\frac{D \vee B \qquad C \vee \neg A}{(D \vee C)\sigma} \qquad [\textit{ordered resolution with selection}]$$

if $\sigma = \mathrm{mgu}(A, B)$ and

(i) $B\sigma$ strictly maximal w. r. t. $D\sigma$;

(ii) nothing is selected in D by S;

(iii) either $\neg A$ is selected, or else nothing is selected in $C \vee \neg A$ and $\neg A\sigma$ is maximal in $C\sigma$.

$$\frac{C \vee A \vee B}{(C \vee A)\sigma} \qquad [\textit{ordered factoring}]$$

if $\sigma = \mathrm{mgu}(A, B)$ and $A\sigma$ is maximal in $C\sigma$ and nothing is selected in C.

Special Case: Propositional Logic

For ground clauses the resolution inference simplifies to

$$\frac{D \vee A \qquad C \vee \neg A}{D \vee C}$$

if

(i) $A \succ D$;

(ii) nothing is selected in D by. S;

(iii) $\neg A$ is selected in $C \vee \neg A$, or else nothing is selected in $C \vee \neg A$ and $\neg A \succeq \max(C)$.

Note: For positive literals, $A \succ D$ is the same as $A \succ \max(D)$.

Search Spaces Become Smaller

1	$A \vee B$		
2	$A \vee \boxed{\neg B}$		
3	$\neg A \vee B$		
4	$\neg A \vee \boxed{\neg B}$		
5	$B \vee B$	Res 1, 3	
6	B	Fact 5	
7	$\neg A$	Res 6, 4	
8	A	Res 6, 2	
9	$\bot$	Res 8, 7	

we assume $A \succ B$ and S as indicated by $\boxed{X}$. The maximal literal in a clause is depicted in red.

With this ordering and selection function the refutation proceeds strictly deterministically in this example. Generally, proof search will still be non-deterministic but the search space will be much smaller than with unrestricted resolution.

Avoiding Rotation Redundancy

From

$$\dfrac{\dfrac{C_1 \vee A \quad C_2 \vee \neg A \vee B}{C_1 \vee C_2 \vee B} \qquad C_3 \vee \neg B}{C_1 \vee C_2 \vee C_3}$$

we can obtain by *rotation*

$$\dfrac{C_1 \vee A \qquad \dfrac{C_2 \vee \neg A \vee B \quad C_3 \vee \neg B}{C_2 \vee \neg A \vee C_3}}{C_1 \vee C_2 \vee C_3}$$

another proof of the same clause. In large proofs many rotations are possible. However, if $A \succ B$, then the second proof does not fulfill the orderings restrictions.

Conclusion: In the presence of orderings restrictions (however one chooses $\succ$) no rotations are possible. In other words, orderings identify exactly one representant in any class of of rotation-equivalent proofs.

Lifting Lemma for $Res_S^{\succ}$

Lemma 3.41 *Let D and C be variable-disjoint clauses. If*

$$\begin{array}{ccc} D & & C \\ \downarrow \sigma & & \downarrow \rho \\ \hline D\sigma & & C\rho \\ \hline & C' & \end{array} \qquad \textit{[propositional inference in } Res_S^{\succ}\textit{]}$$

and if $S(D\sigma) \simeq S(D)$, $S(C\rho) \simeq S(C)$ *(that is, "corresponding" literals are selected), then there exists a substitution* τ *such that*

$$\begin{array}{c} \dfrac{D \qquad C}{C''} \\ \big\downarrow \tau \\ C' = C''\tau \end{array} \qquad \text{[inference in } Res_S^{\succ}\text{]}$$

An analogous lifting lemma holds for factorization.

Saturation of General Clause Sets

Corollary 3.42 *Let N be a set of general clauses saturated under $Res_S^{\succ}$, i. e., $Res_S^{\succ}(N) \subseteq N$. Then there exists a selection function S' such that $S|_N = S'|_N$ and $G_\Sigma(N)$ is also saturated, i. e.,*

$$Res_{S'}^{\succ}(G_\Sigma(N)) \subseteq G_\Sigma(N).$$

Proof. We first define the selection function S' such that $S'(C) = S(C)$ for all clauses $C \in G_\Sigma(N) \cap N$. For $C \in G_\Sigma(N) \setminus N$ we choose a fixed but arbitrary clause $D \in N$ with $C \in G_\Sigma(D)$ and define $S'(C)$ to be those occurrences of literals that are ground instances of the occurrences selected by S in D. Then proceed as in the proof of Corollary 3.34 using the above lifting lemma. □

Soundness and Refutational Completeness

Theorem 3.43 *Let $\succ$ be an atom ordering and S a selection function such that $Res_S^{\succ}(N) \subseteq N$. Then*

$$N \models \bot \Leftrightarrow \bot \in N$$

Proof. The "$\Leftarrow$" part is trivial. For the "$\Rightarrow$" part consider first the propositional level: Construct a candidate interpretation I_N as for unrestricted resolution, except that clauses C in N that have selected literals are not productive, even when they are false in I_C and when their maximal atom occurs only once and positively. The result for general clauses follows using Corollary 3.42. □

Craig-Interpolation

A theoretical application of ordered resolution is Craig-Interpolation:

Theorem 3.44 (Craig 1957) *Let F and G be two propositional formulas such that $F \models G$. Then there exists a formula H (called the* interpolant *for $F \models G$), such that H contains only prop. variables occurring both in F and in G, and such that $F \models H$ and $H \models G$.*

Proof. Translate F and $\neg G$ into CNF. let N and M, resp., denote the resulting clause set. Choose an atom ordering $\succ$ for which the prop. variables that occur in F but not in G are maximal. Saturate N into N^* w. r. t. $Res_S^{\succ}$ with an empty selection function S . Then saturate $N^* \cup M$ w. r. t. $Res_S^{\succ}$ to derive $\bot$. As N^* is already saturated, due to the ordering restrictions only inferences need to be considered where premises, if they are from N^*, only contain symbols that also occur in G. The conjunction of these premises is an interpolant H. The theorem also holds for first-order formulas. For universal formulas the above proof can be easily extended. In the general case, a proof based on resolution technology is more complicated because of Skolemization. □

Redundancy

So far: local restrictions of the resolution inference rules using orderings and selection functions.

Is it also possible to delete clauses altogether? Under which circumstances are clauses unnecessary? (Conjecture: e. g., if they are tautologies or if they are subsumed by other clauses.)

Intuition: If a clause is guaranteed to be neither a minimal counterexample nor productive, then we do not need it.

A Formal Notion of Redundancy

Let N be a set of ground clauses and C a ground clause (not necessarily in N). C is called *redundant* w. r. t. N, if there exist $C_1, \dots, C_n \in N$, $n \geq 0$, such that $C_i \prec C$ and $C_1, \dots, C_n \models C$.

Redundancy for general clauses: C is called *redundant* w. r. t. N, if all ground instances $C\sigma$ of C are redundant w. r. t. $G_\Sigma(N)$.

Intuition: Redundant clauses are neither minimal counterexamples nor productive.

Note: The same ordering $\prec$ is used for ordering restrictions and for redundancy (and for the completeness proof).

Examples of Redundancy

Proposition 3.45 *Some redundancy criteria:*

- C *tautology (i.e.,* $\models C$*)* $\Rightarrow$ C *redundant w.r.t. any set* N.
- $C\sigma \subset D \;\Rightarrow\; D$ *redundant w.r.t.* $N \cup \{C\}$.
- $C\sigma \subseteq D \;\Rightarrow\; D \vee \overline{L}\sigma$ *redundant w.r.t.* $N \cup \{C \vee L,\ D\}$.

(Under certain conditions one may also use non-strict subsumption, but this requires a slightly more complicated definition of redundancy.)

Saturation up to Redundancy

N is called *saturated up to redundancy* (w.r.t. $Res_S^{\succ}$)

$$:\Leftrightarrow Res_S^{\succ}(N \setminus Red(N)) \subseteq N \cup Red(N)$$

Theorem 3.46 *Let* N *be saturated up to redundancy. Then*

$$N \models \bot \Leftrightarrow \bot \in N$$

Proof (Sketch). (i) Ground case:

- consider the construction of the candidate interpretation $I_N^{\succ}$ for $Res_S^{\succ}$
- redundant clauses are not productive
- redundant clauses in N are not minimal counterexamples for $I_N^{\succ}$

The premises of "essential" inferences are either minimal counterexamples or productive.

(ii) Lifting: no additional problems over the proof of Theorem 3.43. □

Monotonicity Properties of Redundancy

Theorem 3.47

(i) $N \subseteq M \Rightarrow Red(N) \subseteq Red(M)$

(ii) $M \subseteq Red(N) \Rightarrow Red(N) \subseteq Red(N \setminus M)$

Proof. Exercise. □

We conclude that redundancy is preserved when, during a theorem proving process, one adds (derives) new clauses or deletes redundant clauses.

A Resolution Prover

So far: static view on completeness of resolution:

Saturated sets are inconsistent if and only if they contain $\bot$.

We will now consider a dynamic view:

How can we get saturated sets in practice?

The theorems 3.46 and 3.47 are the basis for the completeness proof of our prover RP.

Rules for Simplifications and Deletion

We want to employ the following rules for simplification of prover states N:

- *Deletion of tautologies*

$$N \cup \{C \vee A \vee \neg A\} \ \triangleright \ N$$

- *Deletion of subsumed clauses*

$$N \cup \{C, D\} \ \triangleright \ N \cup \{C\}$$

  if $C\sigma \subseteq D$ (C *subsumes* D).

- *Reduction* (also called *subsumption resolution*)

$$N \cup \{C \vee L, \ \ D \vee C\sigma \vee \overline{L}\sigma\} \ \triangleright \ N \cup \{C \vee L, D \vee C\sigma\}$$

Resolution Prover RP

3 clause sets: N(ew) containing new resolvents
P(rocessed) containing simplified resolvents
clauses get into O(ld) once their inferences have been computed

Strategy: Inferences will only be computed when there are no possibilities for simplification

Transition Rules for RP (I)

Tautology elimination

$\boldsymbol{N} \cup \{C\} \mid \boldsymbol{P} \mid \boldsymbol{O} \quad \rhd \quad \boldsymbol{N} \mid \boldsymbol{P} \mid \boldsymbol{O}$

if C is a tautology

Forward subsumption

$\boldsymbol{N} \cup \{C\} \mid \boldsymbol{P} \mid \boldsymbol{O} \quad \rhd \quad \boldsymbol{N} \mid \boldsymbol{P} \mid \boldsymbol{O}$

if some $D \in \boldsymbol{P} \cup \boldsymbol{O}$ subsumes C

Backward subsumption

$\boldsymbol{N} \cup \{C\} \mid \boldsymbol{P} \cup \{D\} \mid \boldsymbol{O} \quad \rhd \quad \boldsymbol{N} \cup \{C\} \mid \boldsymbol{P} \mid \boldsymbol{O}$

$\boldsymbol{N} \cup \{C\} \mid \boldsymbol{P} \mid \boldsymbol{O} \cup \{D\} \quad \rhd \quad \boldsymbol{N} \cup \{C\} \mid \boldsymbol{P} \mid \boldsymbol{O}$

if C strictly subsumes D

Transition Rules for RP (II)

Forward reduction

$\boldsymbol{N} \cup \{C \vee L\} \mid \boldsymbol{P} \mid \boldsymbol{O} \quad \rhd \quad \boldsymbol{N} \cup \{C\} \mid \boldsymbol{P} \mid \boldsymbol{O}$

if there exists $D \vee L' \in \boldsymbol{P} \cup \boldsymbol{O}$
such that $\overline{L} = L'\sigma$ and $D\sigma \subseteq C$

Backward reduction

$\boldsymbol{N} \mid \boldsymbol{P} \cup \{C \vee L\} \mid \boldsymbol{O} \quad \rhd \quad \boldsymbol{N} \mid \boldsymbol{P} \cup \{C\} \mid \boldsymbol{O}$

$\boldsymbol{N} \mid \boldsymbol{P} \mid \boldsymbol{O} \cup \{C \vee L\} \quad \rhd \quad \boldsymbol{N} \mid \boldsymbol{P} \cup \{C\} \mid \boldsymbol{O}$

if there exists $D \vee L' \in \boldsymbol{N}$
such that $\overline{L} = L'\sigma$ and $D\sigma \subseteq C$

Transition Rules for RP (III)

Clause processing

$\boldsymbol{N} \cup \{C\} \mid \boldsymbol{P} \mid \boldsymbol{O} \quad \rhd \quad \boldsymbol{N} \mid \boldsymbol{P} \cup \{C\} \mid \boldsymbol{O}$

Inference computation

$\emptyset \mid \boldsymbol{P} \cup \{C\} \mid \boldsymbol{O} \quad \rhd \quad \boldsymbol{N} \mid \boldsymbol{P} \mid \boldsymbol{O} \cup \{C\},$

with $\boldsymbol{N} = Res_S^{\succ}(\boldsymbol{O} \cup \{C\})$

Soundness and Completeness

Theorem 3.48

$$N \models \bot \quad \Leftrightarrow \quad N \mid \emptyset \mid \emptyset \quad \overset{*}{\rhd} \quad N' \cup \{\bot\} \mid \_ \mid \_$$

Proof in L. Bachmair, H. Ganzinger: Resolution Theorem Proving appeared in the Handbook of Automated Reasoning, 2001

Fairness

Problem:

If N is inconsistent, then $N \mid \emptyset \mid \emptyset \overset{*}{\rhd} N' \cup \{\bot\} \mid \_ \mid \_$.

Does this imply that *every* derivation starting from an inconsistent set N eventually produces $\bot$?

No: a clause could be kept in $\boldsymbol{P}$ without ever being used for an inference.

We need in addition a *fairness condition:*

If an inference is possible forever (that is, none of its premises is ever deleted), then it must be computed eventually.

One possible way to guarantee fairness: Implement $\boldsymbol{P}$ as a queue (there are other techniques to guarantee fairness).

With this additional requirement, we get a stronger result: If N is inconsistent, then every *fair* derivation will eventually produce $\bot$.

Hyperresolution

There are *many* variants of resolution. (We refer to [Bachmair, Ganzinger: Resolution Theorem Proving] for further reading.)

One well-known example is hyperresolution (Robinson 1965):

Assume that several negative literals are selected in a clause C. If we perform an inference with C, then one of the selected literals is eliminated.

Suppose that the remaining selected literals of C are again selected in the conclusion. Then we must eliminate the remaining selected literals one by one by further resolution steps.

Hyperresolution replaces these successive steps by a single inference. As for $Res_S^{\succ}$, the calculus is parameterized by an atom ordering $\succ$ and a selection function S.

$$\frac{D_1 \vee B_1 \quad \ldots \quad D_n \vee B_n \qquad C \vee \neg A_1 \vee \ldots \vee \neg A_n}{(D_1 \vee \ldots \vee D_n \vee C)\sigma}$$

with $\sigma = \mathrm{mgu}(A_1 \doteq B_1, \ldots, A_n \doteq B_n)$, if

(i) $B_i\sigma$ strictly maximal in $D_i\sigma$, $1 \leq i \leq n$;

(ii) nothing is selected in D_i;

(iii) the indicated occurrences of the $\neg A_i$ are exactly the ones selected by S, or else nothing is selected in the right premise and $n = 1$ and $\neg A_1\sigma$ is maximal in $C\sigma$.

Similarly to resolution, hyperresolution has to be complemented by a factoring inference.

As we have seen, hyperresolution can be simulated by iterated binary resolution.

However this yields intermediate clauses which HR might not derive, and many of them might not be extendable into a full HR inference.

3.14 Summary: Resolution Theorem Proving

- Resolution is a machine calculus.
- Subtle interleaving of enumerating ground instances and proving inconsistency through the use of unification.
- Parameters: atom ordering $\succ$ and selection function S. On the non-ground level, ordering constraints can (only) be solved approximatively.
- Completeness proof by constructing candidate interpretations from productive clauses $C \vee A$, $A \succ C$; inferences with those reduce counterexamples.
- *Local* restrictions of inferences via $\succ$ and S
  $\Rightarrow$ fewer proof variants.
- *Global* restrictions of the search space via elimination of redundancy
  $\Rightarrow$ computing with "smaller" clause sets;
  $\Rightarrow$ termination on many decidable fragments.
- However: not good enough for dealing with orderings, equality and more specific algebraic theories (lattices, abelian groups, rings, fields)
  $\Rightarrow$ further specialization of inference systems required.

3.15 Other Inference Systems

Instantiation-based methods for FOL:

- (Semantic) Tableau;
- Resolution-based instance generation;
- Disconnection calculus.

Further (mainly propositional) proof systems:

- Hilbert calculus;
- Sequent calculus;
- Natural deduction.

Instantiation-Based Methods for FOL

Idea:

Overlaps of complementary literals produce instantiations (as in resolution);

However, contrary to resolution, clauses are not recombined.

Instead: treat remaining variables as constant and use efficient propositional proof methods, such as DPLL.

There are both saturation-based variants, such as partial instantiation [Hooker et al.] or resolution-based instance generation (Inst-Gen) [Ganzinger and Korovin], and tableau-style variants, such as the disconnection calculus [Billon; Letz and Stenz].

Hilbert Calculus

Hilbert calculus:

Direct proof method (proves a theorem from axioms, rather than refuting its negation)

Axiom schemes, e. g.,

$$F \to (G \to F)$$
$$(F \to (G \to H)) \to ((F \to G) \to (F \to H))$$

plus Modus ponens:

$$\frac{F \qquad F \to G}{G}$$

Unsuitable for both humans and machines.

Natural Deduction

Natural deduction (Prawitz):

Models the concept of proofs from assumptions as humans do it (cf. Fitting or Huth/Ryan).

Sequent Calculus

Sequent calculus (Gentzen):

Assumptions internalized into the data structure of sequents

$$F_1, \ldots, F_m \rightarrow G_1, \ldots, G_k$$

meaning

$$F_1 \wedge \cdots \wedge F_m \rightarrow G_1 \vee \cdots \vee G_k$$

A kind of mixture between natural deduction and semantic tableaux.

Perfect symmetry between the handling of assumptions and their consequences.

Can be used both backwards and forwards.

4 First-Order Logic with Equality

Equality is the most important relation in mathematics and functional programming.

In principle, problems in first-order logic with equality can be handled by, e. g., resolution theorem provers.

Equality is theoretically difficult: First-order functional programming is Turing-complete.

But: resolution theorem provers cannot even solve problems that are intuitively easy.

Consequence: to handle equality efficiently, knowledge must be integrated into the theorem prover.

4.1 Handling Equality Naively

Proposition 4.1 *Let F be a closed first-order formula with equality. Let $\sim \notin \Pi$ be a new predicate symbol. The set $Eq(\Sigma)$ contains the formulas*

$$\begin{gathered}
\forall x\,(x \sim x)\\
\forall x, y\,(x \sim y \rightarrow y \sim x)\\
\forall x, y, z\,(x \sim y \wedge y \sim z \rightarrow x \sim z)\\
\forall \vec{x}, \vec{y}\,(x_1 \sim y_1 \wedge \cdots \wedge x_n \sim y_n \rightarrow f(x_1, \ldots, x_n) \sim f(y_1, \ldots, y_n))\\
\forall \vec{x}, \vec{y}\,(x_1 \sim y_1 \wedge \cdots \wedge x_m \sim y_m \wedge p(x_1, \ldots, x_m) \rightarrow p(y_1, \ldots, y_m))
\end{gathered}$$

for every $f \in \Omega$ and $p \in \Pi$. Let $\tilde{F}$ be the formula that one obtains from F if every occurrence of $\approx$ is replaced by $\sim$. Then F is satisfiable if and only if $Eq(\Sigma) \cup \{\tilde{F}\}$ is satisfiable.

Proof. Let $\Sigma = (\Omega, \Pi)$, let $\Sigma_1 = (\Omega, \Pi \cup \{\sim\})$.

For the “only if” part assume that F is satisfiable and let $\mathcal{A}$ be a Σ-model of F. Then we define a Σ_1-algebra $\mathcal{B}$ in such a way that $\mathcal{B}$ and $\mathcal{A}$ have the same universe, $f_{\mathcal{B}} = f_{\mathcal{A}}$ for every $f \in \Omega$, $p_{\mathcal{B}} = p_{\mathcal{A}}$ for every $p \in \Pi$, and $\sim_{\mathcal{B}}$ is the identity relation on the universe. It is easy to check that $\mathcal{B}$ is a model of both $\tilde{F}$ and of $Eq(\Sigma)$.

The proof of the “if” part consists of two steps.

Assume that the Σ_1-algebra $\mathcal{B} = (U_{\mathcal{B}},\ (f_{\mathcal{B}} : U^n \to U)_{f \in \Omega},\ (p_{\mathcal{B}} \subseteq U_{\mathcal{B}}^m)_{p \in \Pi \cup \{\sim\}})$ is a model of $Eq(\Sigma) \cup \{\tilde{F}\}$. In the first step, we can show that the interpretation $\sim_{\mathcal{B}}$ of $\sim$ in $\mathcal{B}$ is a congruence relation. We will prove this for the symmetry property, the other properties of congruence relations, that is, reflexivity, transitivity, and congruence with respect to functions and predicates are shown analogously. Let $a, a' \in U_{\mathcal{B}}$ such that $a \sim_{\mathcal{B}} a'$. We have to show that $a' \sim_{\mathcal{B}} a$. Since $\mathcal{B}$ is a model of $Eq(\Sigma)$, $\mathcal{B}(\beta)(\forall x, y\,(x \sim y \rightarrow y \sim x)) = 1$ for every β, hence $\mathcal{B}(\beta[x \mapsto b_1, y \mapsto b_2])(x \sim y \rightarrow y \sim x) = 1$ for every β and every

$b_1, b_2 \in U_{\mathcal{B}}$. Set $b_1 = a$ and $b_2 = a'$, then $1 = \mathcal{B}(\beta[x \mapsto a, y \mapsto a'])(x \sim y \to y \sim x) = (a \sim_{\mathcal{B}} a' \to a' \sim_{\mathcal{B}} a)$, and since $a \sim_{\mathcal{B}} a'$ holds by assumption, $a' \sim_{\mathcal{B}} a$ must also hold.

In the second step, we will now construct a Σ-algebra $\mathcal{A}$ from $\mathcal{B}$ and the congruence relation $\sim_{\mathcal{B}}$. Let $[a]$ be the congruence class of an element $a \in U_{\mathcal{B}}$ with respect to $\sim_{\mathcal{B}}$. The universe $U_{\mathcal{A}}$ of $\mathcal{A}$ is the set $\{ [a] \mid a \in U_{\mathcal{B}} \}$ of congruence classes of the universe of $\mathcal{B}$. For a function symbol $f \in \Omega$, we define $f_{\mathcal{A}}([a_1], \dots, [a_n]) = [f_{\mathcal{B}}(a_1, \dots, a_n)]$, and for a predicate symbol $p \in \Pi$, we define $([a_1], \dots, [a_n]) \in p_{\mathcal{A}}$ if and only if $(a_1, \dots, a_n) \in p_{\mathcal{B}}$. Observe that this is well-defined: If we take different representatives of the same congruence class, we get the same result by congruence of $\sim_{\mathcal{B}}$. Now for every Σ-term t and every $\mathcal{B}$-assignment β, $[\mathcal{B}(\beta)(t)] = \mathcal{A}(\gamma)(t)$, where γ is the $\mathcal{A}$-assignment that maps every variable x to $[\beta(x)]$, and analogously for every Σ-formula G, $\mathcal{B}(\beta)(\tilde{G}) = \mathcal{A}(\gamma)(G)$. Both properties can easily shown by structural induction. Consequently, $\mathcal{A}$ is a model of F. $\square$

By giving the equality axioms explicitly, first-order problems with equality can in principle be solved by a standard resolution or tableaux prover.

But this is unfortunately not efficient (mainly due to the transitivity and congruence axioms).

Roadmap

How to proceed:

- Arbitrary binary relations.
- Equations (unit clauses with equality):

  Term rewrite systems.
  Expressing semantic consequence syntactically.
  Entailment for equations.

- Equational clauses:

  Entailment for clauses with equality.

4.2 Abstract Reduction Systems

Abstract reduction system: $(A, \to)$, where

A is a set,

$\to \subseteq A \times A$ is a binary relation on A.

$\rightarrow^0$	$= \{ (a,a) \mid a \in A \}$	*identity*
$\rightarrow^{i+1}$	$= \rightarrow^i \circ \rightarrow$	*$i+1$-fold composition*
$\rightarrow^+$	$= \bigcup_{i>0} \rightarrow^i$	*transitive closure*
$\rightarrow^*$	$= \bigcup_{i\geq 0} \rightarrow^i \; = \; \rightarrow^+ \cup \rightarrow^0$	*reflexive transitive closure*
$\rightarrow^=$	$= \rightarrow \cup \rightarrow^0$	*reflexive closure*
$\rightarrow^{-1}$	$= \leftarrow \; = \{ (b,c) \mid c \rightarrow b \}$	*inverse*
$\leftrightarrow$	$= \rightarrow \cup \leftarrow$	*symmetric closure*
$\leftrightarrow^+$	$= (\leftrightarrow)^+$	*transitive symmetric closure*
$\leftrightarrow^*$	$= (\leftrightarrow)^*$	*refl. trans. symmetric closure*

$b \in A$ is *reducible*, if there is a c such that $b \rightarrow c$.

b is *in normal form (irreducible)*, if it is not reducible.

c is a *normal form of* b, if $b \rightarrow^* c$ and c is in normal form.
Notation: $c = b{\downarrow}$ (if the normal form of b is unique).

b and c are *joinable*, if there is a a such that $b \rightarrow^* a \leftarrow^* c$.
Notation: $b \downarrow c$.

A relation $\rightarrow$ is called

Church-Rosser, if $b \leftrightarrow^* c$ implies $b \downarrow c$.

confluent, if $b \leftarrow^* a \rightarrow^* c$ implies $b \downarrow c$.

locally confluent, if $b \leftarrow a \rightarrow c$ implies $b \downarrow c$.

terminating, if there is no infinite descending chain $b_0 \rightarrow b_1 \rightarrow b_2 \rightarrow \ldots$.

normalizing, if every $b \in A$ has a normal form.

convergent, if it is confluent and terminating.

Lemma 4.2 *If $\rightarrow$ is terminating, then it is normalizing.*

Note: The reverse implication does not hold.

Theorem 4.3 *The following properties are equivalent:*

(i) $\rightarrow$ *has the Church-Rosser property.*

(ii) $\rightarrow$ *is confluent.*

Proof. (i)⇒(ii): trivial.

(ii)⇒(i): by induction on the number of peaks in
the derivation $b \leftrightarrow^* c$. □

Lemma 4.4 *If $\rightarrow$ is confluent, then every element has at most one normal form.*

Proof. Suppose that some element $a \in A$ has normal forms b and c, then $b \leftarrow^* a \rightarrow^* c$. If $\rightarrow$ is confluent, then $b \rightarrow^* d \leftarrow^* c$ for some $d \in A$. Since b and c are normal forms, both derivations must be empty, hence $b \rightarrow^0 d \leftarrow^0 c$, so b, c, and d must be identical. $\square$

Corollary 4.5 *If $\rightarrow$ is normalizing and confluent, then every element b has a unique normal form.*

Proposition 4.6 *If $\rightarrow$ is normalizing and confluent, then $b \leftrightarrow^* c$ if and only if $b\!\downarrow = c\!\downarrow$.*

Proof. Either using Thm. 4.3 or directly by induction on the length of the derivation of $b \leftrightarrow^* c$. $\square$

Well-Founded Orderings

Lemma 4.7 *If $\rightarrow$ is a terminating binary relation over A, then $\rightarrow^+$ is a well-founded partial ordering.*

Proof. Transitivity of $\rightarrow^+$ is obvious; irreflexivity and well-foundedness follow from termination of $\rightarrow$. $\square$

Lemma 4.8 *If $>$ is a well-founded partial ordering and $\rightarrow \subseteq >$, then $\rightarrow$ is terminating.*

Proving Confluence

Theorem 4.9 ("Newman's Lemma") *If a terminating relation $\rightarrow$ is locally confluent, then it is confluent.*

Proof. Let $\rightarrow$ be a terminating and locally confluent relation. Then $\rightarrow^+$ is a well-founded ordering. Define $P(a) \;\Leftrightarrow\; \big(\forall b, c : b \leftarrow^* a \rightarrow^* c \Rightarrow b \downarrow c\big)$.

We prove $P(a)$ for all $a \in A$ by well-founded induction over $\rightarrow^+$:

Case 1: $b \leftarrow^0 a \rightarrow^* c$: trivial.

Case 2: $b \leftarrow^* a \rightarrow^0 c$: trivial.

Case 3: $b \leftarrow^* b' \leftarrow a \rightarrow c' \rightarrow^* c$: use local confluence, then use the induction hypothesis. $\square$

Proving Termination: Monotone Mappings

Let $(A, >_A)$ and $(B, >_B)$ be partial orderings. A mapping $\varphi : A \to B$ is called *monotone*, if $a >_A a'$ implies $\varphi(a) >_B \varphi(a')$ for all $a, a' \in A$.

Lemma 4.10 *If $\varphi : A \to B$ is a monotone mapping from $(A, >_A)$ to $(B, >_B)$ and $(B, >_B)$ is well-founded, then $(A, >_A)$ is well-founded.*

4.3 Rewrite Systems

Let E be a set of equations.

The *rewrite relation* $\to_E \subseteq \mathrm{T}_\Sigma(X) \times \mathrm{T}_\Sigma(X)$ is defined by

$$
\begin{array}{lll}
s \to_E t & \text{iff} & \text{there exist } (l \approx r) \in E,\ p \in \mathrm{pos}(s), \\
 & & \text{and } \sigma : X \to \mathrm{T}_\Sigma(X), \\
 & & \text{such that } s/p = l\sigma \text{ and } t = s[r\sigma]_p.
\end{array}
$$

An instance of the lhs (left-hand side) of an equation is called a *redex* (reducible expression). *Contracting* a redex means replacing it with the corresponding instance of the rhs (right-hand side) of the rule.

An equation $l \approx r$ is also called a *rewrite rule*, if l is not a variable and $\mathrm{var}(l) \supseteq \mathrm{var}(r)$.

Notation: $l \to r$.

A set of rewrite rules is called a *term rewrite system (TRS)*.

We say that a set of equations E or a TRS R is terminating, if the rewrite relation $\to_E$ or $\to_R$ has this property.

(Analogously for other properties of abstract reduction systems).

Note: If E is terminating, then it is a TRS.

E-Algebras

Let E be a set of equations. A Σ-algebra $\mathcal{A}$ is called an *E-algebra*, if $\mathcal{A} \models \forall\vec{x}(s \approx t)$ for all $\forall\vec{x}(s \approx t) \in E$.

If $E \models \forall\vec{x}(s \approx t)$ (i. e., $\forall\vec{x}(s \approx t)$ is valid in all E-algebras), we write this also as $s \approx_E t$.

Goal:
Use the rewrite relation $\to_E$ to express the semantic consequence relation syntactically:

$s \approx_E t$ if and only if $s \leftrightarrow_E^* t$.

Let E be a set of equations over $\mathrm{T}_\Sigma(X)$. The following inference system allows to derive consequences of E:

$$E \vdash t \approx t \qquad \textit{(Reflexivity)}$$

$$\frac{E \vdash t \approx t'}{E \vdash t' \approx t} \qquad \textit{(Symmetry)}$$

$$\frac{E \vdash t \approx t' \qquad E \vdash t' \approx t''}{E \vdash t \approx t''} \qquad \textit{(Transitivity)}$$

$$\frac{E \vdash t_1 \approx t'_1 \quad \dots \quad E \vdash t_n \approx t'_n}{E \vdash f(t_1, \dots, t_n) \approx f(t'_1, \dots, t'_n)} \qquad \textit{(Congruence)}$$

$$E \vdash t\sigma \approx t'\sigma \qquad \textit{(Instance)}$$
if $(t \approx t') \in E$ and $\sigma : X \to \mathrm{T}_\Sigma(X)$

Lemma 4.11 *The following properties are equivalent:*

(i) $s \leftrightarrow_E^* t$

(ii) $E \vdash s \approx t$ *is derivable.*

(Proof Scetch Follows)

Proof. (i)⇒(ii): $s \leftrightarrow_E t$ implies $E \vdash s \approx t$ by induction on the depth of the position where the rewrite rule is applied; then $s \leftrightarrow_E^* t$ implies $E \vdash s \approx t$ by induction on the number of rewrite steps in $s \leftrightarrow_E^* t$.

(ii)⇒(i): By induction on the size (number of symbols) of the derivation for $E \vdash s \approx t$. □

Constructing a *quotient algebra:*

Let X be a set of variables.

For $t \in \mathrm{T}_\Sigma(X)$ let $[t] = \{\, t' \in \mathrm{T}_\Sigma(X) \mid E \vdash t \approx t' \,\}$ be the *congruence class* of t.

Define a Σ-algebra $\mathrm{T}_\Sigma(X)/E$ (abbreviated by $\mathcal{T}$) as follows:

$U_{\mathcal{T}} = \{\, [t] \mid t \in \mathrm{T}_\Sigma(X) \,\}$.

$f_{\mathcal{T}}([t_1], \dots, [t_n]) = [f(t_1, \dots, t_n)]$ for $f \in \Omega$.

Lemma 4.12 $f_{\mathcal{T}}$ *is well-defined: If* $[t_i] = [t'_i]$, *then* $[f(t_1, \dots, t_n)] = [f(t'_1, \dots, t'_n)]$.

Proof. Follows directly from the *Congruence* rule for $\vdash$. □

Lemma 4.13 $\mathcal{T} = \mathrm{T}_\Sigma(X)/E$ *is an* E*-algebra. (Proof Follows)*

Proof. Let $\forall x_1 \dots x_n(s \approx t)$ be an equation in E; let β be an arbitrary assignment.

We have to show that $\mathcal{T}(\beta)(\forall\vec{x}(s \approx t)) = 1$, or equivalently, that $\mathcal{T}(\gamma)(s) = \mathcal{T}(\gamma)(t)$ for all $\gamma = \beta[\, x_i \mapsto [t_i] \mid 1 \leq i \leq n\,]$ with $[t_i] \in U_{\mathcal{T}}$.

Let $\sigma = [t_1/x_1, \dots, t_n/x_n]$, then $s\sigma \in \mathcal{T}(\gamma)(s)$ and $t\sigma \in \mathcal{T}(\gamma)(t)$.

By the *Instance* rule, $E \vdash s\sigma \approx t\sigma$ is derivable, hence $\mathcal{T}(\gamma)(s) = [s\sigma] = [t\sigma] = \mathcal{T}(\gamma)(t)$. $\square$

Lemma 4.14 *Let* X *be a countably infinite set of variables; let* $s, t \in \mathrm{T}_\Sigma(X)$*. If* $\mathrm{T}_\Sigma(X)/E \models \forall\vec{x}(s \approx t)$*, then* $E \vdash s \approx t$ *is derivable. (Proof Follows)*

Proof. Assume that $\mathcal{T} \models \forall\vec{x}(s \approx t)$, i. e., $\mathcal{T}(\beta)(\forall\vec{x}(s \approx t)) = 1$. Consequently, $\mathcal{T}(\gamma)(s) = \mathcal{T}(\gamma)(t)$ for all $\gamma = \beta[\, x_i \mapsto [t_i] \mid 1 \leq i \leq n\,]$ with $[t_i] \in U_{\mathcal{T}}$.

Choose $t_i = x_i$, then $[s] = \mathcal{T}(\gamma)(s) = \mathcal{T}(\gamma)(t) = [t]$, so $E \vdash s \approx t$ is derivable by definition of $\mathcal{T}$. $\square$

Theorem 4.15 ("Birkhoff's Theorem") *Let* X *be a countably infinite set of variables, let* E *be a set of (universally quantified) equations. Then the following properties are equivalent for all* $s, t \in \mathrm{T}_\Sigma(X)$*:*

(i) $s \leftrightarrow_E^* t$.

(ii) $E \vdash s \approx t$ *is derivable.*

(iii) $s \approx_E t$, *i. e.,* $E \models \forall\vec{x}(s \approx t)$.

(iv) $\mathrm{T}_\Sigma(X)/E \models \forall\vec{x}(s \approx t)$.

Proof. (i)$\Leftrightarrow$(ii): Lemma 4.11.

(ii)$\Rightarrow$(iii): By induction on the size of the derivation for $E \vdash s \approx t$.

(iii)$\Rightarrow$(iv): Obvious, since $\mathcal{T} = \mathrm{T}_\Sigma(X)/E$ is an E-algebra.

(iv)$\Rightarrow$(ii): Lemma 4.14. $\square$

Universal Algebra

$\mathrm{T}_\Sigma(X)/E = \mathrm{T}_\Sigma(X)/{\approx_E} = \mathrm{T}_\Sigma(X)/{\leftrightarrow_E^*}$ is called the *free E-algebra* with generating set $X/{\approx_E} = \{\, [x] \mid x \in X \,\}$:

> Every mapping $\varphi : X/{\approx_E} \to \mathcal{B}$ for some E-algebra $\mathcal{B}$ can be extended to a homomorphism $\hat{\varphi} : \mathrm{T}_\Sigma(X)/E \to \mathcal{B}$.

$\mathrm{T}_\Sigma(\emptyset)/E = \mathrm{T}_\Sigma(\emptyset)/{\approx_E} = \mathrm{T}_\Sigma(\emptyset)/{\leftrightarrow_E^*}$ is called the *initial E-algebra.*

$\approx_E = \{\, (s,t) \mid E \models s \approx t \,\}$ is called the *equational theory* of E.

$\approx_E^I = \{\, (s,t) \mid \mathrm{T}_\Sigma(\emptyset)/E \models s \approx t \,\}$ is called the *inductive theory* of E.

Example:

> Let $E = \{\forall x(x + 0 \approx x),\ \forall x \forall y(x + s(y) \approx s(x + y))\}$. Then $x + y \approx_E^I y + x$, but $x + y \not\approx_E y + x$.

Rewrite Relations

Corollary 4.16 *If E is convergent (i. e., terminating and confluent), then $s \approx_E t$ if and only if $s \leftrightarrow_E^* t$ if and only if $s{\downarrow_E} = t{\downarrow_E}$.*

Corollary 4.17 *If E is finite and convergent, then $\approx_E$ is decidable.*

Reminder:
If E is terminating, then it is confluent if and only if it is locally confluent.

Problems:

> Show local confluence of E.
>
> Show termination of E.
>
> Transform E into an equivalent set of equations that is locally confluent and terminating.

4.4 Critical Pairs

Showing local confluence (Sketch):

Problem: If $t_1 \leftarrow_E t_0 \rightarrow_E t_2$, does there exist a term s such that $t_1 \rightarrow_E^* s \leftarrow_E^* t_2$?

If the two rewrite steps happen in different subtrees (disjoint redexes): yes.

If the two rewrite steps happen below each other (overlap at or below a variable position): yes.

If the left-hand sides of the two rules overlap at a non-variable position: needs further investigation.

Question:
Are there rewrite rules $l_1 \rightarrow r_1$ and $l_2 \rightarrow r_2$ such that some subterm l_1/p and l_2 have a common instance $(l_1/p)\sigma_1 = l_2\sigma_2$?

Observation:
If we assume w.o.l.o.g. that the two rewrite rules do not have common variables, then only a single substitution is necessary: $(l_1/p)\sigma = l_2\sigma$.

Further observation:
The mgu of l_1/p and l_2 subsumes all unifiers σ of l_1/p and l_2.

Let $l_i \rightarrow r_i$ $(i = 1, 2)$ be two rewrite rules in a TRS R whose variables have been renamed such that $\mathrm{var}(l_1) \cap \mathrm{var}(l_2) = \emptyset$. (Remember that $\mathrm{var}(l_i) \supseteq \mathrm{var}(r_i)$.)

Let $p \in \mathrm{pos}(l_1)$ be a position such that l_1/p is not a variable and σ is an mgu of l_1/p and l_2.

Then $r_1\sigma \leftarrow l_1\sigma \rightarrow (l_1\sigma)[r_2\sigma]_p$.

$\langle r_1\sigma, (l_1\sigma)[r_2\sigma]_p\rangle$ is called a *critical pair* of R.

The critical pair is *joinable* (or: converges), if $r_1\sigma \downarrow_R (l_1\sigma)[r_2\sigma]_p$.

Theorem 4.18 ("Critical Pair Theorem") *A TRS R is locally confluent if and only if all its critical pairs are joinable.*

Proof. "only if": obvious, since joinability of a critical pair is a special case of local confluence.

"if": Suppose s rewrites to t_1 and t_2 using rewrite rules $l_i \rightarrow r_i \in R$ at positions $p_i \in \mathrm{pos}(s)$, where $i = 1, 2$. Without loss of generality, we can assume that the two rules are variable disjoint, hence $s/p_i = l_i\theta$ and $t_i = s[r_i\theta]_{p_i}$.

We distinguish between two cases: Either p_1 and p_2 are in disjoint subtrees ($p_1 \mid\mid p_2$), or one is a prefix of the other (w.o.l.o.g., $p_1 \leq p_2$).

Case 1: $p_1 \mid\mid p_2$.

Then $s = s[l_1\theta]_{p_1}[l_2\theta]_{p_2}$, and therefore $t_1 = s[r_1\theta]_{p_1}[l_2\theta]_{p_2}$ and $t_2 = s[l_1\theta]_{p_1}[r_2\theta]_{p_2}$.

Let $t_0 = s[r_1\theta]_{p_1}[r_2\theta]_{p_2}$. Then clearly $t_1 \to_R t_0$ using $l_2 \to r_2$ and $t_2 \to_R t_0$ using $l_1 \to r_1$.

Case 2: $p_1 \leq p_2$.

Case 2.1: $p_2 = p_1q_1q_2$, where l_1/q_1 is some variable x.

In other words, the second rewrite step takes place at or below a variable in the first rule. Suppose that x occurs m times in l_1 and n times in r_1 (where $m \geq 1$ and $n \geq 0$).

Then $t_1 \to_R^* t_0$ by applying $l_2 \to r_2$ at all positions $p_1q'q_2$, where q' is a position of x in r_1.

Conversely, $t_2 \to_R^* t_0$ by applying $l_2 \to r_2$ at all positions p_1qq_2, where q is a position of x in l_1 different from q_1, and by applying $l_1 \to r_1$ at p_1 with the substitution θ', where $\theta' = \theta[x \mapsto (x\theta)[r_2\theta]_{q_2}]$.

Case 2.2: $p_2 = p_1p$, where p is a non-variable position of l_1.

Then $s/p_2 = l_2\theta$ and $s/p_2 = (s/p_1)/p = (l_1\theta)/p = (l_1/p)\theta$, so θ is a unifier of l_2 and l_1/p.

Let σ be the mgu of l_2 and l_1/p, then $\theta = \tau \circ \sigma$ and $\langle r_1\sigma, (l_1\sigma)[r_2\sigma]_p \rangle$ is a critical pair.

By assumption, it is joinable, so $r_1\sigma \to_R^* v \leftarrow_R^* (l_1\sigma)[r_2\sigma]_p$.

Consequently, $t_1 = s[r_1\theta]_{p_1} = s[r_1\sigma\tau]_{p_1} \to_R^* s[v\tau]_{p_1}$ and $t_2 = s[r_2\theta]_{p_2} = s[(l_1\theta)[r_2\theta]_p]_{p_1} = s[(l_1\sigma\tau)[r_2\sigma\tau]_p]_{p_1} = s[((l_1\sigma)[r_2\sigma]_p)\tau]_{p_1} \to_R^* s[v\tau]_{p_1}$.

This completes the proof of the Critical Pair Theorem. □

Note: Critical pairs between a rule and (a renamed variant of) itself must be considered – except if the overlap is at the root (i. e., $p = \varepsilon$).

Corollary 4.19 *A terminating TRS R is confluent if and only if all its critical pairs are joinable.*

Proof. By Newman's Lemma and the Critical Pair Theorem. □

Corollary 4.20 *For a finite terminating TRS, confluence is decidable.*

Proof. For every pair of rules and every non-variable position in the first rule there is at most one critical pair $\langle u_1, u_2 \rangle$.

Reduce every u_i to some normal form u_i'. If $u_1' = u_2'$ for every critical pair, then R is confluent, otherwise there is some non-confluent situation $u_1' \leftarrow_R^* u_1 \leftarrow_R s \to_R u_2 \to_R^* u_2'$. □

4.5 Termination

Termination problems:

> Given a finite TRS R and a term t, are all R-reductions starting from t terminating?
>
> Given a finite TRS R, are all R-reductions terminating?

Proposition 4.21 *Both termination problems for TRSs are undecidable in general.*

Proof. Encode Turing machines using rewrite rules and reduce the (uniform) halting problems for TMs to the termination problems for TRSs. □

Consequence:

> Decidable criteria for termination are not complete.

Reduction Orderings

Goal:

> Given a finite TRS R, show termination of R by looking at finitely many rules $l \to r \in R$, rather than at infinitely many possible replacement steps $s \to_R s'$.

A binary relation $\sqsupset$ over $T_\Sigma(X)$ is called *compatible with* Σ*-operations*, if $s \sqsupset s'$ implies $f(t_1, \ldots, s, \ldots, t_n) \sqsupset f(t_1, \ldots, s', \ldots, t_n)$ for all $f \in \Omega$ and $s, s', t_i \in T_\Sigma(X)$.

Lemma 4.22 *The relation $\sqsupset$ is compatible with Σ-operations, if and only if $s \sqsupset s'$ implies $t[s]_p \sqsupset t[s']_p$ for all $s, s', t \in T_\Sigma(X)$ and $p \in \text{pos}(t)$.*

Note: *compatible with* Σ*-operations* = *compatible with contexts*.

A binary relation $\sqsupset$ over $T_\Sigma(X)$ is called *stable under substitutions*, if $s \sqsupset s'$ implies $s\sigma \sqsupset s'\sigma$ for all $s, s' \in T_\Sigma(X)$ and substitutions σ.

A binary relation $\sqsupset$ is called a *rewrite relation*, if it is compatible with Σ-operations and stable under substitutions.

Example: If R is a TRS, then $\to_R$ is a rewrite relation.

A strict partial ordering over $T_\Sigma(X)$ that is a rewrite relation is called *rewrite ordering*.

A well-founded rewrite ordering is called *reduction ordering*.

Theorem 4.23 *A TRS R terminates if and only if there exists a reduction ordering $\succ$ such that $l \succ r$ for every rule $l \to r \in R$.*

Proof. "if": $s \rightarrow_R s'$ if and only if $s = t[l\sigma]_p$, $s' = t[r\sigma]_p$. If $l \succ r$, then $l\sigma \succ r\sigma$ and therefore $t[l\sigma]_p \succ t[r\sigma]_p$. This implies $\rightarrow_R \subseteq \succ$. Since $\succ$ is a well-founded ordering, $\rightarrow_R$ is terminating.

"only if": Define $\succ = \rightarrow_R^+$. If $\rightarrow_R$ is terminating, then $\succ$ is a reduction ordering. □

The Interpretation Method

Proving termination by interpretation:

Let $\mathcal{A}$ be a Σ-algebra; let $\succ$ be a well-founded strict partial ordering on its universe.

Define the ordering $\succ_\mathcal{A}$ over $\mathrm{T}_\Sigma(X)$ by $s \succ_\mathcal{A} t$ iff $\mathcal{A}(\beta)(s) \succ \mathcal{A}(\beta)(t)$ for all assignments $\beta : X \rightarrow U_\mathcal{A}$.

Is $\succ_\mathcal{A}$ a reduction ordering?

Lemma 4.24 *$\succ_\mathcal{A}$ is stable under substitutions.*

Proof. Let $s \succ_\mathcal{A} s'$, that is, $\mathcal{A}(\beta)(s) \succ \mathcal{A}(\beta)(s')$ for all assignments $\beta : X \rightarrow U_\mathcal{A}$. Let σ be a substitution. We have to show that $\mathcal{A}(\gamma)(s\sigma) \succ \mathcal{A}(\gamma)(s'\sigma)$ for all assignments $\gamma : X \rightarrow U_\mathcal{A}$. Choose $\beta = \gamma \circ \sigma$, then by the substitution lemma, $\mathcal{A}(\gamma)(s\sigma) = \mathcal{A}(\beta)(s) \succ \mathcal{A}(\beta)(s') = \mathcal{A}(\gamma)(s'\sigma)$. Therefore $s\sigma \succ_\mathcal{A} s'\sigma$. □

A function $F : U_\mathcal{A}^n \rightarrow U_\mathcal{A}$ is called *monotone* (with respect to $\succ$), if $a \succ a'$ implies $F(b_1, \dots, a, \dots, b_n) \succ F(b_1, \dots, a', \dots, b_n)$ for all $a, a', b_i \in U_\mathcal{A}$.

Lemma 4.25 *If the interpretation $f_\mathcal{A}$ of every function symbol f is monotone w. r. t. $\succ$, then $\succ_\mathcal{A}$ is compatible with Σ-operations.*

Proof. Let $s \succ s'$, that is, $\mathcal{A}(\beta)(s) \succ \mathcal{A}(\beta)(s')$ for all $\beta : X \rightarrow U_\mathcal{A}$. Let $\beta : X \rightarrow U_\mathcal{A}$ be an arbitrary assignment. Then

$$\begin{aligned}\mathcal{A}(\beta)(f(t_1, \dots, s, \dots, t_n)) &= f_\mathcal{A}(\mathcal{A}(\beta)(t_1), \dots, \mathcal{A}(\beta)(s), \dots, \mathcal{A}(\beta)(t_n)) \\ &\succ f_\mathcal{A}(\mathcal{A}(\beta)(t_1), \dots, \mathcal{A}(\beta)(s'), \dots, \mathcal{A}(\beta)(t_n)) \\ &= \mathcal{A}(\beta)(f(t_1, \dots, s', \dots, t_n))\end{aligned}$$

Therefore $f(t_1, \dots, s, \dots, t_n) \succ_\mathcal{A} f(t_1, \dots, s', \dots, t_n)$. □

Theorem 4.26 *If the interpretation $f_\mathcal{A}$ of every function symbol f is monotone w. r. t. $\succ$, then $\succ_\mathcal{A}$ is a reduction ordering.*

Proof. By the previous two lemmas, $\succ_\mathcal{A}$ is a rewrite relation. If there were an infinite chain $s_1 \succ_\mathcal{A} s_2 \succ_\mathcal{A} \dots$, then it would correspond to an infinite chain $\mathcal{A}(\beta)(s_1) \succ \mathcal{A}(\beta)(s_2) \succ \dots$ (with β chosen arbitrarily). Thus $\succ_\mathcal{A}$ is well-founded. Irreflexivity and transitivity are proved similarly. □

Polynomial Orderings

Polynomial orderings:

Instance of the interpretation method:

The carrier set $U_{\mathcal{A}}$ is some subset of the natural numbers.

To every function symbol f with arity n we associate a polynomial $P_f(X_1, \dots, X_n) \in \mathbb{N}[X_1, \dots, X_n]$ with coefficients in $\mathbb{N}$ and indeterminates $X_1, \dots, X_n$. Then we define $f_{\mathcal{A}}(a_1, \dots, a_n) = P_f(a_1, \dots, a_n)$ for $a_i \in U_{\mathcal{A}}$.

Requirement 1:

If $a_1, \dots, a_n \in U_{\mathcal{A}}$, then $f_{\mathcal{A}}(a_1, \dots, a_n) \in U_{\mathcal{A}}$. (Otherwise, $\mathcal{A}$ would not be a Σ-algebra.)

Requirement 2:

$f_{\mathcal{A}}$ must be monotone (w. r. t. $\succ$).

From now on:

$U_{\mathcal{A}} = \{ n \in \mathbb{N} \mid n \geq 2 \}$.

If $\mathsf{arity}(f) = 0$, then P_f is a constant ≥ 2.

If $\mathsf{arity}(f) = n \geq 1$, then P_f is a polynomial $P(X_1, \dots, X_n)$, such that every X_i occurs in some monomial with exponent at least 1 and non-zero coefficient.

$\Rightarrow$ Requirements 1 and 2 are satisfied.

The mapping from function symbols to polynomials can be extended to terms: A term t containing the variables $x_1, \dots, x_n$ yields a polynomial P_t with indeterminates $X_1, \dots, X_n$ (where X_i corresponds to $\beta(x_i)$).

Example:

$\Omega = \{b,\ f,\ g\}$ with $\mathsf{arity}(b) = 0$, $\mathsf{arity}(f) = 1$, $\mathsf{arity}(g) = 3$,
$U_{\mathcal{A}} = \{ n \in \mathbb{N} \mid n \geq 2 \}$,
$P_b = 3, \quad P_f(X_1) = X_1^2, \quad P_g(X_1, X_2, X_3) = X_1 + X_2 X_3$.

Let $t = g(f(b), f(x), y)$, then $P_t(X, Y) = 9 + X^2 Y$.

If P, Q are polynomials in $\mathbb{N}[X_1, \dots, X_n]$, we write $P > Q$ if $P(a_1, \dots, a_n) > Q(a_1, \dots, a_n)$ for all $a_1, \dots, a_n \in U_{\mathcal{A}}$.

Clearly, $l \succ_{\mathcal{A}} r$ iff $P_l > P_r$.

Question: Can we check $P_l > P_r$ automatically?

Hilbert's 10th Problem:

Given a polynomial $P \in \mathbb{Z}[X_1, \dots, X_n]$ with integer coefficients, is $P = 0$ for some n-tuple of natural numbers?

Theorem 4.27 *Hilbert's 10th Problem is undecidable.*

Proposition 4.28 *Given a polynomial interpretation and two terms l, r, it is undecidable whether $P_l > P_r$.*

Proof. By reduction of Hilbert's 10th Problem. □

One possible solution:

Test whether $P_l(a_1, \dots, a_n) > P_r(a_1, \dots, a_n)$ for all $a_1, \dots, a_n \in \{ x \in \mathbb{R} \mid x \geq 2 \}$.

This is decidable (but very slow). Since $U_\mathcal{A} \subseteq \{ x \in \mathbb{R} \mid x \geq 2 \}$, it implies $P_l > P_r$.

Another solution (Ben Cherifa and Lescanne):

Consider the difference $P_l(X_1, \dots, X_n) - P_r(X_1, \dots, X_n)$ as a polynomial with real coefficients and apply the following inference system to it to show that it is positive for all $a_1, \dots, a_n \in U_\mathcal{A}$:

$P \ \Rightarrow_{BCL} \ \top$,

if P contains at least one monomial with a positive coefficient and no monomial with a negative coefficient.

$P + cX_1^{p_1} \cdots X_n^{p_n} - dX_1^{q_1} \cdots X_n^{q_n} \ \Rightarrow_{BCL} \ P + c'X_1^{p_1} \dots X_n^{p_n}$,

if $c, d > 0$, $p_i \geq q_i$ for all i, and $c' = c - d \cdot 2^{(q_1 - p_1) + \cdots + (q_n - p_n)} \geq 0$.

$P + cX_1^{p_1} \cdots X_n^{p_n} - dX_1^{q_1} \cdots X_n^{q_n} \ \Rightarrow_{BCL} \ P - d'X_1^{q_1} \dots X_n^{q_n}$,

if $c, d > 0$, $p_i \geq q_i$ for all i, and $d' = d - c \cdot 2^{(p_1 - q_1) + \cdots + (p_n - q_n)} > 0$.

Lemma 4.29 *If $P \Rightarrow_{BCL} P'$, then $P(a_1, \dots, a_n) \geq P'(a_1, \dots, a_n)$ for all $a_1, \dots, a_n \in U_\mathcal{A}$.*

Proof. Follows from the fact that $a_i \in U_\mathcal{A}$ implies $a_i \geq 2$. □

Proposition 4.30 *If $P \Rightarrow_{BCL}^{+} \top$, then $P(a_1, \dots, a_n) > 0$ for all $a_1, \dots, a_n \in U_\mathcal{A}$.*

Simplification Orderings

The *proper subterm ordering* $\rhd$ is defined by $s \rhd t$ if and only if $s/p = t$ for some position $p \neq \varepsilon$ of s.

A rewrite ordering $\succ$ over $\mathrm{T}_\Sigma(X)$ is called *simplification ordering*, if it has the *subterm property:* $s \rhd t$ implies $s \succ t$ for all $s, t \in \mathrm{T}_\Sigma(X)$.

Example:

Let R_{emb} be the rewrite system $R_{\text{emb}} = \{\, f(x_1, \dots, x_n) \to x_i \mid f \in \Omega,\, 1 \leq i \leq n = \mathsf{arity}(f)\,\}$.

Define $\rhd_{\text{emb}} = \to^+_{R_{\text{emb}}}$ and $\unrhd_{\text{emb}} = \to^*_{R_{\text{emb}}}$ (*"homeomorphic embedding relation"*).

$\rhd_{\text{emb}}$ is a simplification ordering.

Lemma 4.31 *If $\succ$ is a simplification ordering, then $s \rhd_{\text{emb}} t$ implies $s \succ t$ and $s \unrhd_{\text{emb}} t$ implies $s \succeq t$.*

Proof. Since $\succ$ is transitive and $\succeq$ is transitive and reflexive, it suffices to show that $s \to_{R_{\text{emb}}} t$ implies $s \succ t$. By definition, $s \to_{R_{\text{emb}}} t$ if and only if $s = s[l\sigma]$ and $t = s[r\sigma]$ for some rule $l \to r \in R_{\text{emb}}$. Obviously, $l \rhd r$ for all rules in R_{emb}, hence $l \succ r$. Since $\succ$ is a rewrite relation, $s = s[l\sigma] \succ s[r\sigma] = t$. □

Goal:

Show that every simplification ordering is well-founded (and therefore a reduction ordering).

Note: This works only for *finite* signatures!

To fix this for infinite signatures, the definition of simplification orderings and the definition of embedding have to be modified.

Theorem 4.32 ("Kruskal's Theorem") *Let Σ be a finite signature, let X be a finite set of variables. Then for every infinite sequence $t_1, t_2, t_3, \dots$ there are indices $j > i$ such that $t_j \unrhd_{\text{emb}} t_i$. ($\unrhd_{\text{emb}}$ is called a* well-partial-ordering (wpo).*)*

Proof. See Baader and Nipkow, page 113–115. □

Theorem 4.33 (Dershowitz) *If Σ is a finite signature, then every simplification ordering $\succ$ on $\mathrm{T}_\Sigma(X)$ is well-founded (and therefore a reduction ordering).*

Proof. Suppose that $t_1 \succ t_2 \succ t_3 \succ \ldots$ is an infinite descending chain.

First assume that there is an $x \in \mathrm{var}(t_{i+1}) \setminus \mathrm{var}(t_i)$. Let $\sigma = [t_i/x]$, then $t_{i+1}\sigma \trianglerighteq x\sigma = t_i$ and therefore $t_i = t_i\sigma \succ t_{i+1}\sigma \succeq t_i$, contradicting reflexivity.

Consequently, $\mathrm{var}(t_i) \supseteq \mathrm{var}(t_{i+1})$ and $t_i \in \mathrm{T}_\Sigma(V)$ for all i, where V is the finite set $\mathrm{var}(t_1)$. By Kruskal's Theorem, there are $i < j$ with $t_i \trianglelefteq_{\mathrm{emb}} t_j$. Hence $t_i \preceq t_j$, contradicting $t_i \succ t_j$. □

There are reduction orderings that are not simplification orderings and terminating TRSs that are not contained in any simplification ordering.

Example:

Let $R = \{f(f(x)) \to f(g(f(x)))\}$.

R terminates and $\to_R^+$ is therefore a reduction ordering.

Assume that $\to_R$ were contained in a simplification ordering $\succ$. Then $f(f(x)) \to_R f(g(f(x)))$ implies $f(f(x)) \succ f(g(f(x)))$, and $f(g(f(x))) \trianglerighteq_{\mathrm{emb}} f(f(x))$ implies $f(g(f(x))) \succeq f(f(x))$, hence $f(f(x)) \succ f(f(x))$.

Recursive Path Orderings

Let $\Sigma = (\Omega, \Pi)$ be a finite signature, let $\succ$ be a strict partial ordering *("precedence")* on Ω.

The *lexicographic path ordering* $\succ_{\mathrm{lpo}}$ on $\mathrm{T}_\Sigma(X)$ induced by $\succ$ is defined by: $s \succ_{\mathrm{lpo}} t$ iff

(1) $t \in \mathrm{var}(s)$ and $t \neq s$, or

(2) $s = f(s_1, \ldots, s_m)$, $t = g(t_1, \ldots, t_n)$, and

  (a) $s_i \succeq_{\mathrm{lpo}} t$ for some i, or

  (b) $f \succ g$ and $s \succ_{\mathrm{lpo}} t_j$ for all j, or

  (c) $f = g$, $s \succ_{\mathrm{lpo}} t_j$ for all j, and $(s_1, \ldots, s_m)\ (\succ_{\mathrm{lpo}})_{\mathrm{lex}}\ (t_1, \ldots, t_n)$.

Lemma 4.34 $s \succ_{\mathrm{lpo}} t$ *implies* $\mathrm{var}(s) \supseteq \mathrm{var}(t)$.

Proof. By induction on $|s| + |t|$ and case analysis. □

Theorem 4.35 $\succ_{\text{lpo}}$ *is a simplification ordering on* $\mathrm{T}_\Sigma(X)$.

Proof. Show transitivity, subterm property, stability under substitutions, compatibility with Σ-operations, and irreflexivity, usually by induction on the sum of the term sizes and case analysis. Details: Baader and Nipkow, page 119/120. $\square$

Theorem 4.36 *If the precedence* $\succ$ *is total, then the lexicographic path ordering* $\succ_{\text{lpo}}$ *is total on ground terms, i. e., for all* $s, t \in \mathrm{T}_\Sigma(\emptyset)$*:* $s \succ_{\text{lpo}} t \vee t \succ_{\text{lpo}} s \vee s = t$.

Proof. By induction on $|s| + |t|$ and case analysis. $\square$

Recapitulation:

Let $\Sigma = (\Omega, \Pi)$ be a finite signature, let $\succ$ be a strict partial ordering *("precedence")* on Ω. The *lexicographic path ordering* $\succ_{\text{lpo}}$ on $\mathrm{T}_\Sigma(X)$ induced by $\succ$ is defined by: $s \succ_{\text{lpo}} t$ iff

(1) $t \in \mathrm{var}(s)$ and $t \neq s$, or

(2) $s = f(s_1, \ldots, s_m)$, $t = g(t_1, \ldots, t_n)$, and

  (a) $s_i \succeq_{\text{lpo}} t$ for some i, or

  (b) $f \succ g$ and $s \succ_{\text{lpo}} t_j$ for all j, or

  (c) $f = g$, $s \succ_{\text{lpo}} t_j$ for all j, and $(s_1, \ldots, s_m)\ (\succ_{\text{lpo}})_{\text{lex}}\ (t_1, \ldots, t_n)$.

There are several possibilities to compare subterms in (2)(c):

compare list of subterms lexicographically left-to-right (*"lexicographic path ordering (lpo)"*, Kamin and Lévy)

compare list of subterms lexicographically right-to-left (or according to some permutation π)

compare multiset of subterms using the multiset extension (*"multiset path ordering (mpo)"*, Dershowitz)

to each function symbol f with $\mathsf{arity}(n) \geq 1$ associate a status $\in \{mul\} \cup \{\, lex_\pi \mid \pi : \{1, \ldots, n\} \to \{1, \ldots, n\}\,\}$ and compare according to that status (*"recursive path ordering (rpo) with status"*)

The Knuth-Bendix Ordering

Let $\Sigma = (\Omega, \Pi)$ be a finite signature, let $\succ$ be a strict partial ordering *("precedence")* on Ω, let $w : \Omega \cup X \to \mathbb{R}_0^+$ be a *weight function*, such that the following admissibility conditions are satisfied:

$w(x) = w_0 \in \mathbb{R}^+$ for all variables $x \in X$; $w(c) \geq w_0$ for all constants $c \in \Omega$.

If $w(f) = 0$ for some $f \in \Omega$ with $\mathsf{arity}(f) = 1$, then $f \succeq g$ for all $g \in \Omega$.

The weight function w can be extended to terms as follows:

$$w(t) = \sum_{x \in \mathrm{var}(t)} w(x) \cdot \#(x, t) + \sum_{f \in \Omega} w(f) \cdot \#(f, t).$$

The *Knuth-Bendix ordering* $\succ_{\mathrm{kbo}}$ on $\mathrm{T}_\Sigma(X)$ induced by $\succ$ and w is defined by: $s \succ_{\mathrm{kbo}} t$ iff

(1) $\#(x, s) \geq \#(x, t)$ for all variables x and $w(s) > w(t)$, or

(2) $\#(x, s) \geq \#(x, t)$ for all variables x, $w(s) = w(t)$, and

  (a) $t = x$, $s = f^n(x)$ for some $n \geq 1$, or

  (b) $s = f(s_1, \dots, s_m)$, $t = g(t_1, \dots, t_n)$, and $f \succ g$, or

  (c) $s = f(s_1, \dots, s_m)$, $t = f(t_1, \dots, t_m)$, and $(s_1, \dots, s_m)\ (\succ_{\mathrm{kbo}})_{\mathrm{lex}}\ (t_1, \dots, t_m)$.

Theorem 4.37 *The Knuth-Bendix ordering induced by $\succ$ and w is a simplification ordering on $\mathrm{T}_\Sigma(X)$.*

Proof. Baader and Nipkow, pages 125–129. □

4.6 Knuth-Bendix Completion

Completion:

Goal: Given a set E of equations, transform E into an equivalent convergent set R of rewrite rules.
(If R is finite: decision procedure for E.)

How to ensure termination?

Fix a reduction ordering $\succ$ and construct R in such a way that $\rightarrow_R \subseteq \succ$ (i. e., $l \succ r$ for every $l \rightarrow r \in R$).

How to ensure confluence?

Check that all critical pairs are joinable.

Knuth-Bendix Completion: Inference Rules

The completion procedure is presented as a set of inference rules working on a set of equations E and a set of rules R: $E_0, R_0 \vdash E_1, R_1 \vdash E_2, R_2 \vdash \ldots$

At the beginning, $E = E_0$ is the input set and $R = R_0$ is empty. At the end, E should be empty; then R is the result.

For each step $E, R \vdash E', R'$, the equational theories of $E \cup R$ and $E' \cup R'$ agree: $\approx_{E \cup R} = \approx_{E' \cup R'}$.

Notations:

The formula $s \mathrel{\dot{\approx}} t$ denotes either $s \approx t$ or $t \approx s$.

$\mathrm{CP}(R)$ denotes the set of all critical pairs between rules in R.

Orient:

$$\frac{E \cup \{s \mathrel{\dot{\approx}} t\}, \quad R}{E, \quad R \cup \{s \rightarrow t\}} \qquad \text{if } s \succ t$$

Note: There are equations $s \approx t$ that cannot be oriented, i. e., neither $s \succ t$ nor $t \succ s$.

Trivial equations cannot be oriented – but we don't need them anyway:

Delete:

$$\frac{E \cup \{s \approx s\}, \quad R}{E, \quad R}$$

Critical pairs between rules in R are turned into additional equations:

Deduce:

$$\frac{E, \quad R}{E \cup \{s \approx t\}, \quad R} \qquad \text{if } \langle s, t\rangle \in \mathrm{CP}(R).$$

Note: If $\langle s, t\rangle \in \mathrm{CP}(R)$ then $s \leftarrow_R u \rightarrow_R t$ and hence $R \models s \approx t$.

The following inference rules are not absolutely necessary, but very useful (e. g., to get rid of joinable critical pairs and to deal with equations that cannot be oriented):

Simplify-Eq:

$$\frac{E \cup \{s \mathrel{\dot{\approx}} t\}, \quad R}{E \cup \{u \approx t\}, \quad R} \qquad \text{if } s \rightarrow_R u.$$

Simplification of the right-hand side of a rule is unproblematic.

R-Simplify-Rule:

$$\frac{E, \quad R \cup \{s \rightarrow t\}}{E, \quad R \cup \{s \rightarrow u\}} \qquad \text{if } t \rightarrow_R u.$$

Simplification of the left-hand side may influence orientability and orientation. Therefore, it yields an *equation*:

L-Simplify-Rule:

$$\frac{E, \quad R \cup \{s \rightarrow t\}}{E \cup \{u \approx t\}, \quad R} \qquad \begin{array}{l}\text{if } s \rightarrow_R u \text{ using a rule } l \rightarrow r \in R\\ \text{such that } s \sqsupset l \text{ (see next slide).}\end{array}$$

For technical reasons, the lhs of $s \rightarrow t$ may only be simplified using a rule $l \rightarrow r$, if $l \rightarrow r$ *cannot* be simplified using $s \rightarrow t$, that is, if $s \sqsupset l$, where the *encompassment quasi-ordering* $\mathrel{\underset{\sim}{\sqsupset}}$ is defined by

$$s \mathrel{\underset{\sim}{\sqsupset}} l \ \text{ if } \ s/p = l\sigma \text{ for some } p \text{ and } \sigma$$

and $\sqsupset \, = \, \mathrel{\underset{\sim}{\sqsupset}} \setminus \mathrel{\underset{\sim}{\sqsubset}}$ is the strict part of $\mathrel{\underset{\sim}{\sqsupset}}$.

Lemma 4.38 $\sqsupset$ *is a well-founded strict partial ordering.*

Lemma 4.39 *If* $E, R \vdash E', R'$, *then* $\approx_{E \cup R} \, = \, \approx_{E' \cup R'}$.

Lemma 4.40 *If* $E, R \vdash E', R'$ *and* $\rightarrow_R \, \subseteq \, \succ$, *then* $\rightarrow_{R'} \, \subseteq \, \succ$.

Knuth-Bendix Completion: Correctness Proof

If we run the completion procedure on a set E of equations, different things can happen:

(1) We reach a state where no more inference rules are applicable and E is not empty.
$\Rightarrow$ Failure (try again with another ordering?)

(2) We reach a state where E is empty and all critical pairs between the rules in the current R have been checked.

(3) The procedure runs forever.

In order to treat these cases simultaneously, we need some definitions.

A (finite or infinite sequence) $E_0, R_0 \vdash E_1, R_1 \vdash E_2, R_2 \vdash \dots$ with $R_0 = \emptyset$ is called a *run* of the completion procedure with input E_0 and $\succ$.

For a run, $E_\infty = \bigcup_{i \geq 0} E_i$ and $R_\infty = \bigcup_{i \geq 0} R_i$.

The sets of *persistent equations or rules* of the run are $E_* = \bigcup_{i \geq 0} \bigcap_{j \geq i} E_j$ and $R_* = \bigcup_{i \geq 0} \bigcap_{j \geq i} R_j$.

Note: If the run is finite and ends with E_n, R_n, then $E_* = E_n$ and $R_* = R_n$.

A run is called *fair*, if $CP(R_*) \subseteq E_\infty$ (i. e., if every critical pair between persisting rules is computed at some step of the derivation).

Goal:

Show: If a run is fair and E_* is empty, then R_* is convergent and equivalent to E_0.

In particular: If a run is fair and E_* is empty, then $\approx_{E_0} = \approx_{E_\infty \cup R_\infty} = \leftrightarrow^*_{E_\infty \cup R_\infty} = \downarrow_{R_*}$.

General assumptions from now on:

$E_0, R_0 \vdash E_1, R_1 \vdash E_2, R_2 \vdash \dots$ is a fair run.

R_0 and E_* are empty.

A *proof* of $s \approx t$ in $E_\infty \cup R_\infty$ is a finite sequence $(s_0, \dots, s_n)$ such that $s = s_0$, $t = s_n$, and for all $i \in \{1, \dots, n\}$:

(1) $s_{i-1} \leftrightarrow_{E_\infty} s_i$, or

(2) $s_{i-1} \rightarrow_{R_\infty} s_i$, or

(3) $s_{i-1} \leftarrow_{R_\infty} s_i$.

The pairs (s_{i-1}, s_i) are called *proof steps.*

A proof is called a *rewrite proof in* R_*, if there is a $k \in \{0, \dots, n\}$ such that $s_{i-1} \rightarrow_{R_*} s_i$ for $1 \leq i \leq k$ and $s_{i-1} \leftarrow_{R_*} s_i$ for $k+1 \leq i \leq n$

Idea (Bachmair, Dershowitz, Hsiang):

> Define a well-founded ordering on proofs, such that for every proof that is not a rewrite proof in R_* there is an equivalent smaller proof.
>
> Consequence: For every proof there is an equivalent rewrite proof in R_*.

We associate a *cost* $c(s_{i-1}, s_i)$ with every proof step as follows:

(1) If $s_{i-1} \leftrightarrow_{E_\infty} s_i$, then $c(s_{i-1}, s_i) = (\{s_{i-1}, s_i\}, -, -)$, where the first component is a multiset of terms and $-$ denotes an arbitrary (irrelevant) term.

(2) If $s_{i-1} \rightarrow_{R_\infty} s_i$ using $l \rightarrow r$, then $c(s_{i-1}, s_i) = (\{s_{i-1}\}, l, s_i)$.

(3) If $s_{i-1} \leftarrow_{R_\infty} s_i$ using $l \rightarrow r$, then $c(s_{i-1}, s_i) = (\{s_i\}, l, s_{i-1})$.

Proof steps are compared using the lexicographic combination of the multiset extension of the reduction ordering $\succ$, the encompassment ordering $\sqsupset$, and the reduction ordering $\succ$.

The cost $c(P)$ of a proof P is the multiset of the costs of its proof steps.

The *proof ordering* $\succ_C$ compares the costs of proofs using the multiset extension of the proof step ordering.

Lemma 4.41 *$\succ_C$ is a well-founded ordering.*

Lemma 4.42 *Let P be a proof in $E_\infty \cup R_\infty$. If P is not a rewrite proof in R_*, then there exists an equivalent proof P' in $E_\infty \cup R_\infty$ such that $P \succ_C P'$.*

Proof. If P is not a rewrite proof in R_*, then it contains

(a) a proof step that is in E_∞, or
(b) a proof step that is in $R_\infty \setminus R_*$, or
(c) a subproof $s_{i-1} \leftarrow_{R_*} s_i \rightarrow_{R_*} s_{i+1}$ (peak).

We show that in all three cases the proof step or subproof can be replaced by a smaller subproof:

Case (a): A proof step using an equation $s \dot{\approx} t$ is in E_∞. This equation must be deleted during the run.

If $s \dot{\approx} t$ is deleted using *Orient*:
$\dots s_{i-1} \leftrightarrow_{E_\infty} s_i \dots \quad \Longrightarrow \quad \dots s_{i-1} \rightarrow_{R_\infty} s_i \dots$

If $s \dot{\approx} t$ is deleted using *Delete*:
$\dots s_{i-1} \leftrightarrow_{E_\infty} s_{i-1} \dots \quad \Longrightarrow \quad \dots s_{i-1} \dots$

If $s \dot{\approx} t$ is deleted using *Simplify-Eq*:
$\dots s_{i-1} \leftrightarrow_{E_\infty} s_i \dots \quad \Longrightarrow \quad \dots s_{i-1} \rightarrow_{R_\infty} s' \leftrightarrow_{E_\infty} s_i \dots$

Case (b): A proof step using a rule $s \rightarrow t$ is in $R_\infty \setminus R_*$. This rule must be deleted during the run.

If $s \rightarrow t$ is deleted using *R-Simplify-Rule*:
$\dots s_{i-1} \rightarrow_{R_\infty} s_i \dots \quad \Longrightarrow \quad \dots s_{i-1} \rightarrow_{R_\infty} s' \leftarrow_{R_\infty} s_i \dots$

If $s \rightarrow t$ is deleted using *L-Simplify-Rule*:
$\dots s_{i-1} \rightarrow_{R_\infty} s_i \dots \quad \Longrightarrow \quad \dots s_{i-1} \rightarrow_{R_\infty} s' \leftrightarrow_{E_\infty} s_i \dots$

Case (c): A subproof has the form $s_{i-1} \leftarrow_{R_*} s_i \rightarrow_{R_*} s_{i+1}$.

If there is no overlap or a non-critical overlap:
$\dots s_{i-1} \leftarrow_{R_*} s_i \rightarrow_{R_*} s_{i+1} \dots \quad \Longrightarrow \quad \dots s_{i-1} \rightarrow^*_{R_*} s' \leftarrow^*_{R_*} s_{i+1} \dots$

If there is a critical pair that has been added using *Deduce*:
$\dots s_{i-1} \leftarrow_{R_*} s_i \rightarrow_{R_*} s_{i+1} \dots \quad \Longrightarrow \quad \dots s_{i-1} \leftrightarrow_{E_\infty} s_{i+1} \dots$

In all cases, checking that the replacement subproof is smaller than the replaced subproof is routine. □

Theorem 4.43 *Let $E_0, R_0 \vdash E_1, R_1 \vdash E_2, R_2 \vdash \ldots$ be a fair run and let R_0 and E_* be empty. Then*

(1) every proof in $E_\infty \cup R_\infty$ is equivalent to a rewrite proof in R_*,

(2) R_* is equivalent to E_0, and

(3) R_* is convergent.

Proof. (1) By well-founded induction on $\succ_C$ using the previous lemma.

(2) Clearly $\approx_{E_\infty \cup R_\infty} = \approx_{E_0}$. Since $R_* \subseteq R_\infty$, we get $\approx_{R_*} \subseteq \approx_{E_\infty \cup R_\infty}$. On the other hand, by (1), $\approx_{E_\infty \cup R_\infty} \subseteq \approx_{R_*}$.

(3) Since $\rightarrow_{R_*} \subseteq \succ$, R_* is terminating. By (1), R_* is confluent. □

Knuth-Bendix Completion: Outlook

Classical completion:

Tries to transform a set E of equations into an equivalent convergent term rewrite system.

Fails, if an equation can neither be oriented nor deleted.

Unfailing completion:

Use an ordering $\succ$ that is total on ground terms.

If an equation cannot be oriented, use it in both directions for rewriting (except if that would yield a larger term). In other words, consider the relation $\leftrightarrow_E \cap \not\prec$.

Special case of superposition (see next chapter).

4.7 Superposition

Goal:

Combine the ideas of ordered resolution (overlap maximal literals in a clause) and Knuth-Bendix completion (overlap maximal sides of equations) to get a calculus for equational clauses.

Recapitulation: Equational Clauses

Atom: either $p(s_1, \ldots, s_m)$ with $p \in \Pi$ or $s \approx t$.

Literal: Atom or negated atom.

Clause: (possibly empty) disjunction of literals (all variables implicitly universally quantified).

For refutational theorem proving, it is sufficient to consider sets of clauses: every first-order formula F can be translated into a set of clauses N such that F is unsatisfiable if and only if N is unsatisfiable.

In the non-equational case, unsatisfiability can for instance be checked using the (ordered) resolution calculus.

Recapitulation: Ordered Resolution

(Ordered) resolution: inference rules:

	Ground case:	Non-ground case:
Resolution:	$\dfrac{D' \lor A \qquad C' \lor \neg A}{D' \lor C'}$	$\dfrac{D' \lor A \qquad C' \lor \neg A'}{(D' \lor C')\sigma}$ where $\sigma = \mathrm{mgu}(A, A')$.
Factoring:	$\dfrac{C' \lor A \lor A}{C' \lor A}$	$\dfrac{C' \lor A \lor A'}{(C' \lor A)\sigma}$ where $\sigma = \mathrm{mgu}(A, A')$.

Ordering restrictions:

Let $\succ$ be a well-founded and total ordering on ground atoms.

Literal ordering $\succ_L$: compares literals by comparing lexicographically first the respective atoms using $\succ$ and then their polarities (negative $>$ positive).

Clause ordering $\succ_C$: compares clauses by comparing their multisets of literals using the multiset extension of $\succ_L$.

Ordering restrictions (ground case):

Inference are necessary only if the following conditions are satisfied:

- The left premise of a Resolution inference is not larger than or equal to the right premise.
- The literals that are involved in the inferences ($[\neg]\, A$) are maximal in the respective clauses (strictly maximal for the left premise of Resolution).

Ordering restrictions (non-ground case):

Lift the ground ordering to non-ground literals: A literal L is called [strictly] maximal in a clause C if and only if there exists a ground substitution σ such that for all other literals L' in C: $L\sigma \not\prec L'\sigma$ [$L\sigma \not\preceq L'\sigma$].

Recapitulation: Refutational Completeness

Resolution is (even with ordering restrictions) refutationally complete:

Dynamic view of refutational completeness:

If N is unsatisfiable ($N \models \bot$) then *fair* derivations from N produce $\bot$.

Static view of refutational completeness:

If N is *saturated*, then N is unsatisfiable if and only if $\bot \in N$.

Proving refutational completeness for the ground case:

We have to show:

If N is saturated (i. e., if sufficiently many inferences have been computed), and $\bot \notin N$, then N is satisfiable (i. e., has a model).

Model construction:

Suppose that N be saturated and $\bot \notin N$. We inspect all clauses in N in ascending order and construct a sequence of Herbrand interpretations (starting with the empty interpretation: all atoms are false).

If a clause C is false in the current interpretation, and has a positive and strictly maximal literal A, then extend the current interpretation such that C becomes true: add A to the current interpretation. (Then C is called *productive*.)

Otherwise, leave the current interpretation unchanged.

The sequence of interpretations has the following properties:

(1) If an atom is true in some interpretation, then it remains true in all future interpretations.

(2) If a clause is true at the time where it is inspected, then it remains true in all future interpretations.

(3) If a clause $C = C' \vee A$ is productive, then C remains true and C' remains false in all future interpretations.

Show by induction: if N is saturated and $\bot \notin N$, then every clause in N is either true at the time where it is inspected or productive.

Note:
For the induction proof, it is not necessary that the conclusion of an inference is contained in N. It is sufficient that it is redundant w. r. t. N.

N is called *saturated up to redundancy* if the conclusion of every inference from clauses in $N \setminus \mathit{Red}(N)$ is contained in $N \cup \mathit{Red}(N)$.

Proving refutational completeness for the non-ground case:

If $C_i\theta$ is a ground instance of the clause C_i for $i \in \{0, \dots, n\}$ and

$$\frac{C_n, \dots, C_1}{C_0}$$

and

$$\frac{C_n\theta, \dots, C_1\theta}{C_0\theta}$$

are inferences, then the latter inference is called a *ground instance* of the former.

For a set N of clauses, let $G_\Sigma(N)$ be the set of all ground instances of clauses in N.

Construct the interpretation from the set $G_\Sigma(N)$ of all ground instances of clauses in N:

N is saturated and does not contain $\bot$
$\Rightarrow$ $G_\Sigma(N)$ is saturated and does not contain $\bot$
$\Rightarrow$ $G_\Sigma(N)$ has a Herbrand model I
$\Rightarrow$ I is a model of N.

Observation

It is possible to encode an arbitrary predicate p using a function f_p and a new constant tt:

$$\begin{array}{rcr} p(t_1, \dots, t_n) & \rightsquigarrow & f_p(t_1, \dots, t_n) \approx tt \\ \neg\, p(t_1, \dots, t_n) & \rightsquigarrow & \neg\, f_p(t_1, \dots, t_n) \approx tt \end{array}$$

In equational logic it is therefore sufficient to consider the case that $\Pi = \emptyset$, i. e., equality is the only predicate symbol.

Abbreviation: $s \not\approx t$ instead of $\neg\, s \approx t$.

The Superposition Calculus – Informally

Conventions:

From now on: $\Pi = \emptyset$ (equality is the only predicate).

Inference rules are to be read modulo symmetry of the equality symbol.

We will first explain the ideas and motivations behind the superposition calculus and its completeness proof. Precise definitions will be given later.

Ground inference rules:

Pos. Superposition: $$\frac{D' \vee t \approx t' \qquad C' \vee s[t] \approx s'}{D' \vee C' \vee s[t'] \approx s'}$$

Neg. Superposition: $$\frac{D' \vee t \approx t' \qquad C' \vee s[t] \not\approx s'}{D' \vee C' \vee s[t'] \not\approx s'}$$

Equality Resolution: $$\frac{C' \vee s \not\approx s}{C'}$$

(Note: We will need one further inference rule.)

Ordering restrictions:

Some considerations:

The literal ordering must depend primarily on the larger term of an equation.

As in the resolution case, negative literals must be a bit larger than the corresponding positive literals.

Additionally, we need the following property: If $s \succ t \succ u$, then $s \not\approx u$ must be larger than $s \approx t$. In other words, we must compare first the larger term, then the polarity, and finally the smaller term.

The following construction has the required properties:

Let $\succ$ be a *reduction ordering that is total on ground terms.*

To a positive literal $s \approx t$, we assign the multiset $\{s, t\}$, to a negative literal $s \not\approx t$ the multiset $\{s, s, t, t\}$. The *literal ordering* $\succ_L$ compares these multisets using the multiset extension of $\succ$.

The *clause ordering* $\succ_C$ compares clauses by comparing their multisets of literals using the multiset extension of $\succ_L$.

Ordering restrictions:

Ground inferences are necessary only if the following conditions are satisfied:

- In superposition inferences, the left premise is smaller than the right premise.
- The literals that are involved in the inferences are maximal in the respective clauses (strictly maximal for positive literals in superposition inferences).
- In these literals, the lhs is greater than or equal to the rhs (in superposition inferences: greater than the rhs).

Model construction:

We want to use roughly the same ideas as in the completeness proof for resolution.

But: a Herbrand interpretation does not work for equality: The equality symbol $\approx$ must be interpreted by equality in the interpretation.

Solution: Define a set E of ground equations and take $\mathrm{T}_\Sigma(\emptyset)/E = \mathrm{T}_\Sigma(\emptyset)/{\approx_E}$ as the universe.

Then two ground terms s and t are equal in the interpretation, if and only if $s \approx_E t$.

If E is a terminating and confluent rewrite system R, then two ground terms s and t are equal in the interpretation, if and only if $s \downarrow_R t$.

One problem:

In the completeness proof for the resolution calculus, the following property holds:

If $C = C' \lor A$ with a strictly maximal and positive literal A is false in the current interpretation, then adding A to the current interpretation cannot make any literal of C' true.

This does not hold for superposition:

Let $b \succ c \succ d$. Assume that the current rewrite system (representing the current interpretation) contains the rule $c \to d$. Now consider the clause $b \approx c \lor b \approx d$.

We need a further inference rule to deal with clauses of this kind, either the "Merging Paramodulation" rule of Bachmair and Ganzinger or the following "Equality Factoring" rule due to Nieuwenhuis:

Equality Factoring: $$\frac{C' \lor s \approx t' \lor s \approx t}{C' \lor t \not\approx t' \lor s \approx t'}$$

Note: This inference rule subsumes the usual factoring rule.

How do the non-ground versions of the inference rules for superposition look like?

Main idea as in the resolution calculus:

Replace identity by unifiability. Apply the mgu to the resulting clause. In the ordering restrictions, replace $\succ$ by $\not\preceq$.

However:

As in Knuth-Bendix completion, we do not want to consider overlaps at or below a variable position.

Consequence: there are inferences between ground instances $D\theta$ and $C\theta$ of clauses D and C which are *not* ground instances of inferences between D and C.

Such inferences have to be treated in a special way in the completeness proof.

The Superposition Calculus – Formally

Until now, we have seen most of the ideas behind the superposition calculus and its completeness proof.

We will now start again from the beginning giving precise definitions and proofs.

Inference rules are applied with respect to the commutativity of equality $\approx$.

Inference rules:

Pos. Superposition:

$$\frac{D' \lor t \approx t' \qquad C' \lor s[u] \approx s'}{(D' \lor C' \lor s[t'] \approx s')\sigma}$$

where $\sigma = \mathrm{mgu}(t, u)$ and u is not a variable.

Neg. Superposition:

$$\frac{D' \lor t \approx t' \qquad C' \lor s[u] \not\approx s'}{(D' \lor C' \lor s[t'] \not\approx s')\sigma}$$

where $\sigma = \mathrm{mgu}(t, u)$ and u is not a variable.

Equality Resolution:

$$\frac{C' \lor s \not\approx s'}{C'\sigma}$$

where $\sigma = \mathrm{mgu}(s, s')$.

Equality Factoring:

$$\frac{C' \lor s' \approx t' \lor s \approx t}{(C' \lor t \not\approx t' \lor s \approx t')\sigma}$$

where $\sigma = \mathrm{mgu}(s, s')$.

Theorem 4.44 *All inference rules of the superposition calculus are* correct, *i. e., for every rule*

$$\frac{C_n, \ldots, C_1}{C_0}$$

we have $\{C_1, \ldots, C_n\} \models C_0$.

Proof. Exercise. □

Orderings:

Let $\succ$ be a *reduction ordering that is total on ground terms.*

To a positive literal $s \approx t$, we assign the multiset $\{s, t\}$, to a negative literal $s \not\approx t$ the multiset $\{s, s, t, t\}$. The *literal ordering* $\succ_L$ compares these multisets using the multiset extension of $\succ$.

The *clause ordering* $\succ_C$ compares clauses by comparing their multisets of literals using the multiset extension of $\succ_L$.

Inferences have to be computed only if the following ordering restrictions are satisfied:

– In superposition inferences, after applying the unifier to both premises, the left premise is not greater than or equal to the right one.

– The last literal in each premise is maximal in the respective premise, i. e., there exists no greater literal (strictly maximal for positive literals in superposition inferences, i. e., there exists no greater or equal literal).

– In these literals, the lhs is not smaller than the rhs (in superposition inferences: neither smaller nor equal).

A ground clause C is called *redundant w. r. t. a set of ground clauses* N, if it follows from clauses in N that are smaller than C.

A clause is *redundant w. r. t. a set of clauses* N, if all its ground instances are redundant w. r. t. $G_\Sigma(N)$.

The set of all clauses that are redundant w. r. t. N is denoted by $Red(N)$.

N is called *saturated up to redundancy*, if the conclusion of every inference from clauses in $N \setminus Red(N)$ is contained in $N \cup Red(N)$.

Superposition: Refutational Completeness

For a set E of ground equations, $\mathrm{T}_\Sigma(\emptyset)/E$ is an E-interpretation (or E-algebra) with universe $\{ [t] \mid t \in \mathrm{T}_\Sigma(\emptyset) \}$.

One can show (similar to the proof of Birkhoff's Theorem) that for every *ground* equation $s \approx t$ we have $\mathrm{T}_\Sigma(\emptyset)/E \models s \approx t$ if and only if $s \leftrightarrow_E^* t$.

In particular, if E is a convergent set of rewrite rules R and $s \approx t$ is a ground equation, then $\mathrm{T}_\Sigma(\emptyset)/R \models s \approx t$ if and only if $s \downarrow_R t$. By abuse of terminology, we say that an equation or clause is valid (or true) in R if and only if it is true in $\mathrm{T}_\Sigma(\emptyset)/R$.

Construction of candidate interpretations (Bachmair & Ganzinger 1990):

Let N be a set of clauses not containing $\bot$. Using induction on the clause ordering we define sets of rewrite rules E_C and R_C for all $C \in G_\Sigma(N)$ as follows:

Assume that E_D has already been defined for all $D \in G_\Sigma(N)$ with $D \prec_C C$. Then $R_C = \bigcup_{D \prec_C C} E_D$.

The set E_C contains the rewrite rule $s \to t$, if

(a) $C = C' \vee s \approx t$.

(b) $s \approx t$ is strictly maximal in C.

(c) $s \succ t$.

(d) C is false in R_C.

(e) C' is false in $R_C \cup \{s \to t\}$.

(f) s is irreducible w. r. t. R_C.

In this case, C is called *productive*. Otherwise $E_C = \emptyset$.

Finally, $R_\infty = \bigcup_{D \in G_\Sigma(N)} E_D$.

Lemma 4.45 *If $E_C = \{s \to t\}$ and $E_D = \{u \to v\}$, then $s \succ u$ if and only if $C \succ_C D$.*

Corollary 4.46 *The rewrite systems R_C and R_∞ are convergent.*

Proof. Obviously, $s \succ t$ for all rules $s \to t$ in R_C and R_∞.

Furthermore, it is easy to check that there are no critical pairs between any two rules: Assume that there are rules $u \to v$ in E_D and $s \to t$ in E_C such that u is a subterm of s. As $\succ$ is a reduction ordering that is total on ground terms, we get $u \prec s$ and therefore $D \prec_C C$ and $E_D \subseteq R_C$. But then s would be reducible by R_C, contradicting condition (f). □

Lemma 4.47 *If $D \preceq_C C$ and $E_C = \{s \to t\}$, then $s \succ u$ for every term u occurring in a negative literal in D and $s \succeq v$ for every term v occurring in a positive literal in D.*

Corollary 4.48 *If $D \in G_\Sigma(N)$ is true in R_D, then D is true in R_∞ and R_C for all $C \succ_C D$.*

Proof. If a positive literal of D is true in R_D, then this is obvious.

Otherwise, some negative literal $s \not\approx t$ of D must be true in R_D, hence $s \not\downarrow_{R_D} t$. As the rules in $R_\infty \setminus R_D$ have left-hand sides that are larger than s and t, they cannot be used in a rewrite proof of $s \downarrow t$, hence $s \not\downarrow_{R_C} t$ and $s \not\downarrow_{R_\infty} t$. □

Corollary 4.49 *If $D = D' \vee u \approx v$ is productive, then D' is false and D is true in R_∞ and R_C for all $C \succ_C D$.*

Proof. Obviously, D is true in R_∞ and R_C for all $C \succ_C D$.

Since all negative literals of D' are false in R_D, it is clear that they are false in R_∞ and R_C. For the positive literals $u' \approx v'$ of D', condition (e) ensures that they are false in $R_D \cup \{u \rightarrow v\}$. Since $u' \preceq u$ and $v' \preceq u$ and all rules in $R_\infty \setminus R_D$ have left-hand sides that are larger than u, these rules cannot be used in a rewrite proof of $u' \downarrow v'$, hence $u' \not\downarrow_{R_C} v'$ and $u' \not\downarrow_{R_\infty} v'$. $\square$

Lemma 4.50 ("Lifting Lemma") *Let C be a clause and let θ be a substitution such that $C\theta$ is ground. Then every equality resolution or equality factoring inference from $C\theta$ is a ground instance of an inference from C.*

Proof. Exercise. $\square$

Lemma 4.51 ("Lifting Lemma") *Let $D = D' \vee u \approx v$ and $C = C' \vee [\neg]\ s \approx t$ be two clauses (without common variables) and let θ be a substitution such that $D\theta$ and $C\theta$ are ground.*

If there is a superposition inference between $D\theta$ and $C\theta$ where $u\theta$ and some subterm of $s\theta$ are overlapped, and $u\theta$ does not occur in $s\theta$ at or below a variable position of s, then the inference is a ground instance of a superposition inference from D and C.

Proof. Exercise. $\square$

Theorem 4.52 ("Model Construction") *Let N be a set of clauses that is saturated up to redundancy and does not contain the empty clause. Then we have for every ground clause $C\theta \in G_\Sigma(N)$:*

(i) $E_{C\theta} = \emptyset$ if and only if $C\theta$ is true in $R_{C\theta}$.

(ii) If $C\theta$ is redundant w. r. t. $G_\Sigma(N)$, then it is true in $R_{C\theta}$.

(iii) $C\theta$ is true in R_∞ and in R_D for every $D \in G_\Sigma(N)$ with $D \succ_C C\theta$.

Proof. We use induction on the clause ordering $\succ_C$ and assume that (i)–(iii) are already satisfied for all clauses in $G_\Sigma(N)$ that are smaller than $C\theta$. Note that the "if" part of (i) is obvious from the construction and that condition (iii) follows immediately from (i) and Corollaries 4.48 and 4.49. So it remains to show (ii) and the "only if" part of (i).

Case 1: $C\theta$ is redundant w. r. t. $G_\Sigma(N)$.

If $C\theta$ is redundant w. r. t. $G_\Sigma(N)$, then it follows from clauses in $G_\Sigma(N)$ that are smaller than $C\theta$. By part (iii) of the induction hypothesis, these clauses are true in $R_{C\theta}$. Hence $C\theta$ is true in $R_{C\theta}$.

Case 2: $x\theta$ is reducible by $R_{C\theta}$.

Suppose there is a variable x occurring in C such that $x\theta$ is reducible by $R_{C\theta}$, say $x\theta \rightarrow_{R_{C\theta}} w$. Let the substitution θ' be defined by $x\theta' = w$ and $y\theta' = y\theta$ for every variable $y \neq x$. The clause $C\theta'$ is smaller than $C\theta$. By part (iii) of the induction hypothesis, it is true in $R_{C\theta}$. By congruence, every literal of $C\theta$ is true in $R_{C\theta}$ if and only if the corresponding literal of $C\theta'$ is true in $R_{C\theta}$; hence $C\theta$ is true in $R_{C\theta}$.

Case 3: $C\theta$ contains a maximal negative literal.

Suppose that $C\theta$ does not fall into Case 1 or 2 and that $C\theta = C'\theta \;\vee\; s\theta \not\approx s'\theta$, where $s\theta \not\approx s'\theta$ is maximal in $C\theta$. If $s\theta \approx s'\theta$ is false in $R_{C\theta}$, then $C\theta$ is clearly true in $R_{C\theta}$ and we are done. So assume that $s\theta \approx s'\theta$ is true in $R_{C\theta}$, that is, $s\theta \downarrow_{R_{C\theta}} s'\theta$. Without loss of generality, $s\theta \succeq s'\theta$.

Case 3.1: $s\theta = s'\theta$.

If $s\theta = s'\theta$, then there is an *equality resolution* inference

$$\frac{C'\theta \;\vee\; s\theta \not\approx s'\theta}{C'\theta}\;.$$

As shown in the Lifting Lemma, this is an instance of an *equality resolution* inference

$$\frac{C' \;\vee\; s \not\approx s'}{C'\sigma}$$

where $C = C' \;\vee\; s \not\approx s'$ is contained in N and $\theta = \rho \circ \sigma$. (Without loss of generality, σ is idempotent, therefore $C'\theta = C'\sigma\rho = C'\sigma\sigma\rho = C'\sigma\theta$, so $C'\theta$ is a ground instance of $C'\sigma$.) Since $C\theta$ is not redundant w. r. t. $G_\Sigma(N)$, C is not redundant w. r. t. N. As N is saturated up to redundancy, the conclusion $C'\sigma$ of the inference from C is contained in $N \cup \mathit{Red}(N)$. Therefore, $C'\theta$ is either contained in $G_\Sigma(N)$ and smaller than $C\theta$, or it follows from clauses in $G_\Sigma(N)$ that are smaller than itself (and therefore smaller than $C\theta$). By the induction hypothesis, clauses in $G_\Sigma(N)$ that are smaller than $C\theta$ are true in $R_{C\theta}$, thus $C'\theta$ and $C\theta$ are true in $R_{C\theta}$.

Case 3.2: $s\theta \succ s'\theta$.

If $s\theta \downarrow_{R_{C\theta}} s'\theta$ and $s\theta \succ s'\theta$, then $s\theta$ must be reducible by some rule in some $E_{D\theta} \subseteq R_{C\theta}$. (Without loss of generality we assume that C and D are variable disjoint; so we can use the same substitution θ.) Let $D\theta = D'\theta \;\vee\; t\theta \approx t'\theta$ with $E_{D\theta} = \{t\theta \to t'\theta\}$. Since $D\theta$ is productive, $D'\theta$ is false in $R_{C\theta}$. Besides, by part (ii) of the induction hypothesis, $D\theta$ is not redundant w.r.t. $G_\Sigma(N)$, so D is not redundant w.r.t. N. Note that $t\theta$ cannot occur in $s\theta$ at or below a variable position of s, say $x\theta = w[t\theta]$, since otherwise $C\theta$ would be subject to Case 2 above. Consequently, the *left superposition* inference

$$\frac{D'\theta \;\vee\; t\theta \approx t'\theta \qquad C'\theta \;\vee\; s\theta[t\theta] \not\approx s'\theta}{D'\theta \;\vee\; C'\theta \;\vee\; s\theta[t'\theta] \not\approx s'\theta}$$

is a ground instance of a *left superposition* inference from D and C. By saturation up to redundancy, its conclusion is either contained in $G_\Sigma(N)$ and smaller than $C\theta$, or it follows from clauses in $G_\Sigma(N)$ that are smaller than itself (and therefore smaller than $C\theta$). By the induction hypothesis, these clauses are true in $R_{C\theta}$, thus $D'\theta \;\vee\; C'\theta \;\vee\; s\theta[t'\theta] \not\approx s'\theta$ is true in $R_{C\theta}$. Since $D'\theta$ and $s\theta[t'\theta] \not\approx s'\theta$ are false in $R_{C\theta}$, both $C'\theta$ and $C\theta$ must be true.

Case 4: $C\theta$ does not contain a maximal negative literal.

Suppose that $C\theta$ does not fall into Cases 1 to 3. Then $C\theta$ can be written as $C'\theta \;\vee\; s\theta \approx s'\theta$, where $s\theta \approx s'\theta$ is a maximal literal of $C\theta$. If $E_{C\theta} = \{s\theta \to s'\theta\}$ or $C'\theta$ is true in $R_{C\theta}$ or $s\theta = s'\theta$, then there is nothing to show, so assume that $E_{C\theta} = \emptyset$ and that $C'\theta$ is false in $R_{C\theta}$. Without loss of generality, $s\theta \succ s'\theta$.

Case 4.1: $s\theta \approx s'\theta$ is maximal in $C\theta$, but not strictly maximal.

If $s\theta \approx s'\theta$ is maximal in $C\theta$, but not strictly maximal, then $C\theta$ can be written as $C''\theta \;\vee\; t\theta \approx t'\theta \;\vee\; s\theta \approx s'\theta$, where $t\theta = s\theta$ and $t'\theta = s'\theta$. In this case, there is a *equality factoring* inference

$$\frac{C''\theta \;\vee\; t\theta \approx t'\theta \;\vee\; s\theta \approx s'\theta}{C''\theta \;\vee\; t'\theta \not\approx s'\theta \;\vee\; t\theta \approx t'\theta}$$

This inference is a ground instance of an inference from C. By induction hypothesis, its conclusion is true in $R_{C\theta}$. Trivially, $t'\theta = s'\theta$ implies $t'\theta \downarrow_{R_{C\theta}} s'\theta$, so $t'\theta \not\approx s'\theta$ must be false and $C\theta$ must be true in $R_{C\theta}$.

Case 4.2: $s\theta \approx s'\theta$ is strictly maximal in $C\theta$ and $s\theta$ is reducible.

Suppose that $s\theta \approx s'\theta$ is strictly maximal in $C\theta$ and $s\theta$ is reducible by some rule in $E_{D\theta} \subseteq R_{C\theta}$. Let $D\theta = D'\theta \;\vee\; t\theta \approx t'\theta$ and $E_{D\theta} = \{t\theta \to t'\theta\}$. Since $D\theta$ is productive, $D\theta$ is not redundant and $D'\theta$ is false in $R_{C\theta}$. We can now proceed in essentially the same way as in Case 3.2: If $t\theta$ occurred in $s\theta$ at or below a variable position of s, say $x\theta = w[t\theta]$, then $C\theta$ would be subject to Case 2 above. Otherwise, the *right superposition* inference

$$\frac{D'\theta \ \vee \ t\theta \approx t'\theta \qquad C'\theta \ \vee \ s\theta[t\theta] \approx s'\theta}{D'\theta \ \vee \ C'\theta \ \vee \ s\theta[t'\theta] \approx s'\theta}$$

is a ground instance of a *right superposition* inference from D and C. By saturation up to redundancy, its conclusion is true in $R_{C\theta}$. Since $D'\theta$ and $C'\theta$ are false in $R_{C\theta}$, $s\theta[t'\theta] \approx s'\theta$ must be true in $R_{C\theta}$. On the other hand, $t\theta \approx t'\theta$ is true in $R_{C\theta}$, so by congruence, $s\theta[t\theta] \approx s'\theta$ and $C\theta$ are true in $R_{C\theta}$.

Case 4.3: $s\theta \approx s'\theta$ is strictly maximal in $C\theta$ and $s\theta$ is irreducible.

Suppose that $s\theta \approx s'\theta$ is strictly maximal in $C\theta$ and $s\theta$ is irreducible by $R_{C\theta}$. Then there are three possibilities: $C\theta$ can be true in $R_{C\theta}$, or $C'\theta$ can be true in $R_{C\theta} \cup \{s\theta \to s'\theta\}$, or $E_{C\theta} = \{s\theta \to s'\theta\}$. In the first and the third case, there is nothing to show. Let us therefore assume that $C\theta$ is false in $R_{C\theta}$ and $C'\theta$ is true in $R_{C\theta} \cup \{s\theta \to s'\theta\}$. Then $C'\theta = C''\theta \ \vee \ t\theta \approx t'\theta$, where the literal $t\theta \approx t'\theta$ is true in $R_{C\theta} \cup \{s\theta \to s'\theta\}$ and false in $R_{C\theta}$. In other words, $t\theta \downarrow_{R_{C\theta} \cup \{s\theta \to s'\theta\}} t'\theta$, but not $t\theta \downarrow_{R_{C\theta}} t'\theta$. Consequently, there is a rewrite proof of $t\theta \to^* u \leftarrow^* t'\theta$ by $R_{C\theta} \cup \{s\theta \to s'\theta\}$ in which the rule $s\theta \to s'\theta$ is used at least once. Without loss of generality we assume that $t\theta \succeq t'\theta$. Since $s\theta \approx s'\theta \succ_{\mathrm{L}} t\theta \approx t'\theta$ and $s\theta \succ s'\theta$ we can conclude that $s\theta \succeq t\theta \succ t'\theta$. But then there is only one possibility how the rule $s\theta \to s'\theta$ can be used in the rewrite proof: We must have $s\theta = t\theta$ and the rewrite proof must have the form $t\theta \to s'\theta \to^* u \leftarrow^* t'\theta$, where the first step uses $s\theta \to s'\theta$ and all other steps use rules from $R_{C\theta}$. Consequently, $s'\theta \approx t'\theta$ is true in $R_{C\theta}$. Now observe that there is an *equality factoring* inference

$$\frac{C''\theta \ \vee \ t\theta \approx t'\theta \ \vee \ s\theta \approx s'\theta}{C''\theta \ \vee \ t'\theta \not\approx s'\theta \ \vee \ t\theta \approx t'\theta}$$

whose conclusion is true in $R_{C\theta}$ by saturation. Since the literal $t'\theta \not\approx s'\theta$ must be false in $R_{C\theta}$, the rest of the clause must be true in $R_{C\theta}$, and therefore $C\theta$ must be true in $R_{C\theta}$, contradicting our assumption. This concludes the proof of the theorem. $\square$

A Σ-interpretation $\mathcal{A}$ is called *term-generated,* if for every $b \in U_{\mathcal{A}}$ there is a ground term $t \in \mathrm{T}_\Sigma(\emptyset)$ such that $b = \mathcal{A}(\beta)(t)$.

Lemma 4.53 *Let N be a set of (universally quantified) Σ-clauses and let $\mathcal{A}$ be a term-generated Σ-interpretation. Then $\mathcal{A}$ is a model of $G_\Sigma(N)$ if and only if it is a model of N.*

Proof. ($\Rightarrow$): Let $\mathcal{A} \models G_\Sigma(N)$; let $(\forall \vec{x} C) \in N$. Then $\mathcal{A} \models \forall \vec{x} C$ iff $\mathcal{A}(\gamma[x_i \mapsto a_i])(C) = 1$ for all γ and a_i. Choose ground terms t_i such that $\mathcal{A}(\gamma)(t_i) = a_i$; define θ such that $x_i\theta = t_i$, then $\mathcal{A}(\gamma[x_i \mapsto a_i])(C) = \mathcal{A}(\gamma \circ \theta)(C) = \mathcal{A}(\gamma)(C\theta) = 1$ since $C\theta \in G_\Sigma(N)$.

($\Leftarrow$): Let $\mathcal{A}$ be a model of N; let $C \in N$ and $C\theta \in G_\Sigma(N)$. Then $\mathcal{A}(\gamma)(C\theta) = \mathcal{A}(\gamma \circ \theta)(C) = 1$ since $\mathcal{A} \models N$. $\square$

Theorem 4.54 (Refutational Completeness: Static View) *Let N be a set of clauses that is saturated up to redundancy. Then N has a model if and only if N does not contain the empty clause.*

Proof. If $\bot \in N$, then obviously N does not have a model. If $\bot \notin N$, then the interpretation R_∞ (that is, $\mathrm{T}_\Sigma(\emptyset)/R_\infty$) is a model of all ground instances in $G_\Sigma(N)$ according to part (iii) of the model construction theorem. As $\mathrm{T}_\Sigma(\emptyset)/R_\infty$ is term generated, it is a model of N. $\square$

So far, we have considered only inference rules that add new clauses to the current set of clauses (corresponding to the *Deduce* rule of Knuth-Bendix Completion).

In other words, we have derivations of the form $N_0 \vdash N_1 \vdash N_2 \vdash \dots$, where each N_{i+1} is obtained from N_i by adding the consequence of some inference from clauses in N_i.

Under which circumstances are we allowed to delete (or simplify) a clause during the derivation?

A *run* of the superposition calculus is a sequence $N_0 \vdash N_1 \vdash N_2 \vdash \dots$, such that
(i) $N_i \models N_{i+1}$, and
(ii) all clauses in $N_i \setminus N_{i+1}$ are redundant w. r. t. N_{i+1}.

In other words, during a run we may add a new clause if it follows from the old ones, and we may delete a clause, if it is redundant w. r. t. the remaining ones.

For a run, $N_\infty = \bigcup_{i \geq 0} N_i$ and $N_* = \bigcup_{i \geq 0} \bigcap_{j \geq i} N_j$. The set N_* of all *persistent* clauses is called the *limit* of the run.

Lemma 4.55 *If $N \subseteq N'$, then $Red(N) \subseteq Red(N')$.*

Proof. Obvious. $\square$

Lemma 4.56 *If $N' \subseteq Red(N)$, then $Red(N) \subseteq Red(N \setminus N')$.*

Proof. Follows from the compactness of first-order logic and the well-foundedness of the multiset extension of the clause ordering. $\square$

Lemma 4.57 *Let $N_0 \vdash N_1 \vdash N_2 \vdash \ldots$ be a run. Then $Red(N_i) \subseteq Red(N_\infty)$ and $Red(N_i) \subseteq Red(N_*)$ for every i.*

Proof. Exercise. □

Corollary 4.58 *$N_i \subseteq N_* \cup Red(N_*)$ for every i.*

Proof. If $C \in N_i \setminus N_*$, then there is a $k \geq i$ such that $C \in N_k \setminus N_{k+1}$, so C must be redundant w. r. t. N_{k+1}. Consequently, C is redundant w. r. t. N_*. □

A run is called *fair,* if the conclusion of every inference from clauses in $N_* \setminus Red(N_*)$ is contained in some $N_i \cup Red(N_i)$.

Lemma 4.59 *If a run is fair, then its limit is saturated up to redundancy.*

Proof. If the run is fair, then the conclusion of every inference from non-redundant clauses in N_* is contained in some $N_i \cup Red(N_i)$, and therefore contained in $N_* \cup Red(N_*)$. Hence N_* is saturated up to redundancy. □

Theorem 4.60 (Refutational Completeness: Dynamic View) *Let $N_0 \vdash N_1 \vdash N_2 \vdash \ldots$ be a fair run, let N_* be its limit. Then N_0 has a model if and only if $\bot \notin N_*$.*

Proof. ($\Leftarrow$): By fairness, N_* is saturated up to redundancy. If $\bot \notin N_*$, then it has a term-generated model. Since every clause in N_0 is contained in N_* or redundant w. r. t. N_*, this model is also a model of $G_\Sigma(N_0)$ and therefore a model of N_0.

($\Rightarrow$): Obvious, since $N_0 \models N_*$. □

Superposition: Extensions

Extensions and improvements:

simplification techniques,

selection functions (as for ordered resolution),

redundancy for inferences,

basic strategies,

constraint reasoning.

Theory Reasoning

Superposition vs. resolution + equality axioms:

specialized inference rules, thus no inferences with theory axioms,

computation modulo symmetry,

stronger ordering restrictions,

no variable overlaps,

stronger redundancy criterion.

Similar techniques can be used for other theories:

transitive relations,

dense total orderings without endpoints,

commutativity,

associativity and commutativity,

abelian monoids,

abelian groups,

divisible torsion-free abelian groups.

Observations:

no inferences with theory axioms:
yes, usually possible.

computation modulo theory axioms:
often possible, but requires unification and orderings modulo theory.

stronger ordering restrictions, no variable overlaps:
sometimes possible, but in many cases, certain variable overlaps remain necessary.

stronger redundancy criterion:
depends on the model construction.

In many cases, integrating more theory axioms simplifies matters.

Inefficient unification procedures may be replaced by constraints.

5 Implementation Issues

Problem:

Refutational completeness is nice in theory, but ...

... it guarantees only that proofs will be found eventually, not that they will be found quickly.

Even though orderings and selection functions reduce the number of possible inferences, the search space problem is enormous.

First-order provers "look for a needle in a haystack": It may be necessary to make some millions of inferences to find a proof that is only a few dozens of steps long.

Coping with Large Sets of Formulas

Consequently:

- We must deal with large sets of formulas.
- We must use efficient techniques to find formulas that can be used as partners in an inference.
- We must simplify/eliminate as many formulas as possible.
- We must use efficient techniques to check whether a formula can be simplified/eliminated.

Note:

Often there are several competing implementation techniques.

Design decisions are not independent of each other.

Design decisions are not independent of the particular class of problems we want to solve. (FOL without equality/FOL with equality/unit equations, size of the signature, special algebraic properties like AC, etc.)

5.1 The Main Loop

Standard approach:

Select one clause ("Given clause").

Find many partner clauses that can be used in inferences together with the "given clause" using an appropriate index data structure.

Compute the conclusions of these inferences; add them to the set of clauses.

Consequently: split the set of clauses into two subsets.

- W = "Worked-off" (or "active") clauses: Have already been selected as "given clause". (So all inferences between these clauses have already been computed.)
- U = "Usable" (or "passive") clauses: Have not yet been selected as "given clause".

During each iteration of the main loop:

Select a new given clause C from U; $U := U \setminus \{C\}$.

Find partner clauses D_i from W; $New = Infer(\{ D_i \mid i \in I \}, C)$; $U = U \cup New$; $W = W \cup \{C\}$

Additionally:

Try to simplify C using W. (Skip the remainder of the iteration, if C can be eliminated.)

Try to simplify (or even eliminate) clauses from W using C.

Design decision: should one also simplify U using W?

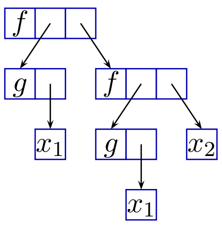
yes $\rightsquigarrow$ "Otter loop":
Advantage: simplifications of U may be useful to derive the empty clause.

no $\rightsquigarrow$ "Discount loop":
Advantage: clauses in U are really passive; only clauses in W have to be kept in index data structure. (Hence: can use index data structure for which retrieval is faster, even if update is slower and space consumption is higher.)

5.2 Term Representations

The obvious data structure for terms: Trees

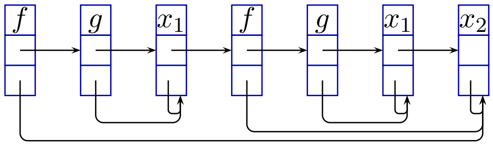
$f(g(x_1), f(g(x_1), x_2))$



optionally: (full) sharing

An alternative: Flatterms

$f(g(x_1), f(g(x_1), x_2))$



need more memory;
but: better suited for preorder term traversal
and easier memory management.

5.3 Index Data Structures

Problem:

For a term t, we want to find all terms s such that

- s is an instance of t,
- s is a generalization of t (i. e., t is an instance of s),
- s and t are unifiable,
- s is a generalization of some subterm of t,
- ...

Requirements:

fast insertion,

fast deletion,

fast retrieval,

small memory consumption.

Note: In applications like functional or logic programming, the requirements are different (insertion and deletion are much less important).

Many different approaches:

- Path indexing
- Discrimination trees
- Substitution trees
- Context trees
- Feature vector indexing

- ...

Perfect filtering:

The indexing technique returns exactly those terms satisfying the query.

Imperfect filtering:

The indexing technique returns some superset of the set of all terms satisfying the query.

Retrieval operations must be followed by an additional check, but the index can often be implemented more efficiently.

Frequently: All occurrences of variables are treated as different variables.

Path Indexing

Path indexing:

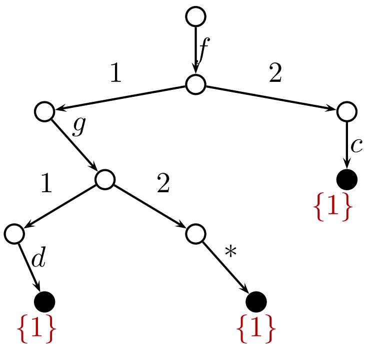
Paths of terms are encoded in a trie ("retrieval tree").

A star $*$ represents arbitrary variables.

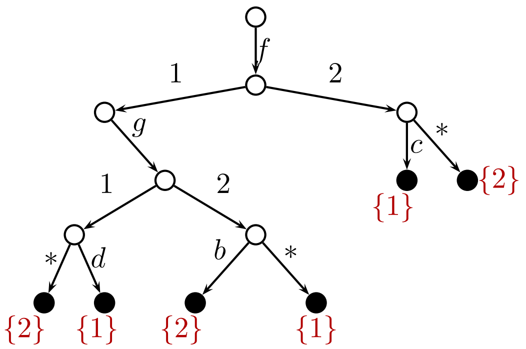
Example: Paths of $f(g(*, b), *)$: $f.1.g.1.*$
$f.1.g.2.b$
$f.2.*$

Each leaf of the trie contains the set of (pointers to) all terms that contain the respective path.

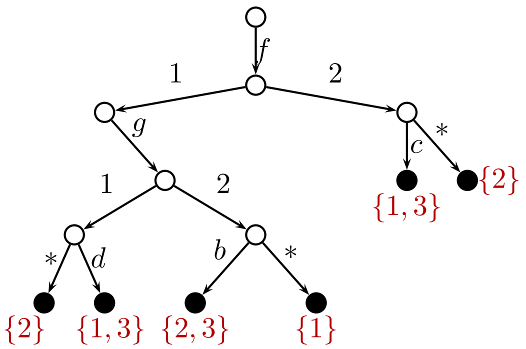
Example: Path index for $\{f(g(d, *), c)\}$



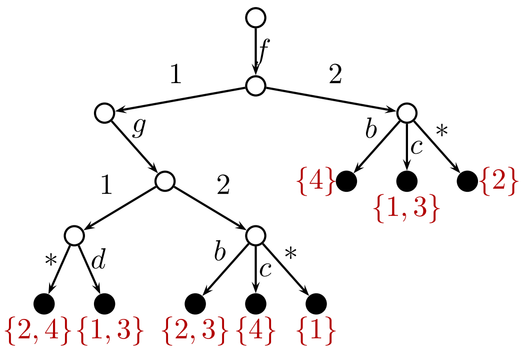
Example: Path index for $\{f(g(d, *), c),\ f(g(*, b), *)\}$



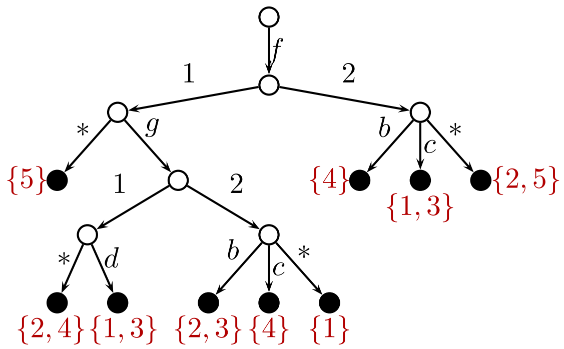
Example: Path index for $\{f(g(d,*),c),\ f(g(*,b),*),\ f(g(d,b),c)\}$



Example: Path index for $\{f(g(d,*),c),\ f(g(*,b),*),\ f(g(d,b),c),\ f(g(*,c),b)\}$



Example: Path index for $\{f(g(d,*),c),\ f(g(*,b),*),\ f(g(d,b),c),\ f(g(*,c),b),\ f(*,*)\}$



Advantages:

Uses little space.

No backtracking for retrieval.

Efficient insertion and deletion.

Good for finding instances.

Disadvantages:

Retrieval requires combining intermediate results for subterms.

Discrimination Trees

Discrimination trees:

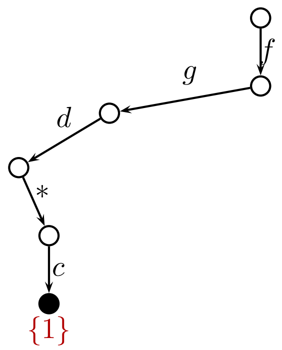
Preorder traversals of terms are encoded in a trie.

A star $*$ represents arbitrary variables.

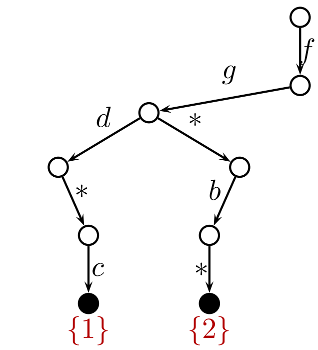
Example: String of $f(g(*, b), *)$: $f.g.*.b.*$

Each leaf of the trie contains (a pointer to) the term that is represented by the path.

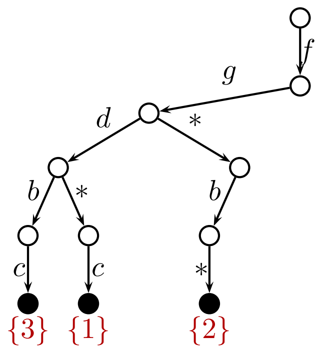
Example: Discrimination tree for $\{f(g(d, *), c)\}$



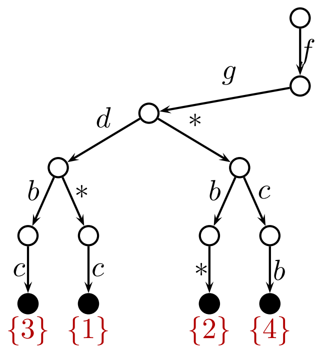
Example: Discrimination tree for $\{f(g(d,*),c),\ f(g(*,b),*)\}$



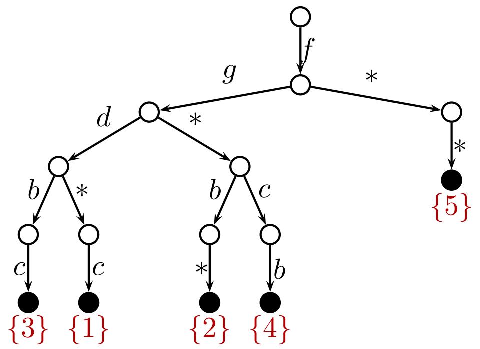
Example: Discrimination tree for $\{f(g(d,*),c),\ f(g(*,b),*),\ f(g(d,b),c)\}$



Example: Discrimination tree for $\{f(g(d,*),c),\ f(g(*,b),*),\ f(g(d,b),c),\ f(g(*,c),b)\}$



Example: Discrimination tree for $\{f(g(d,*),c),\ f(g(*,b),*),\ f(g(d,b),c),\ f(g(*,c),b),$ $f(*,*)\}$



Advantages:

Each leaf yields one term, hence retrieval does not require intersections of intermediate results for subterms.

Good for finding generalizations.

Disadvantages:

Uses more storage than path indexing (due to less sharing).

Uses still more storage, if jump lists are maintained to speed up the search for instances or unifiable terms.

Backtracking required for retrieval.

Literature

Literature:

R. Sekar, I. V. Ramakrishnan, and Andrei Voronkov: Term Indexing, Ch. 26 in Robinson and Voronkov (eds.), *Handbook of Automated Reasoning, Vol. II*, Elsevier, 2001.

Christoph Weidenbach: Combining Superposition, Sorts and Splitting, Ch. 27 in Robinson and Voronkov (eds.), *Handbook of Automated Reasoning, Vol. II*, Elsevier, 2001.

6 Many-Sorted First-Order Logic

Many-Sorted First-order logic

- generalization of first-order logic
- idea is to prohibit ill-defined statements, e.g., $\mathrm{cons}(3, \mathrm{nil}) + 2$
- identical proof theory
- sorts denote subsets of the domain

- variables come with a sort
- functions are declared over the sorts

Many-Sorted Signature

A signature

$$\Sigma_\Upsilon = (\Omega, \Pi, \Upsilon, \upsilon)$$

fixes an alphabet of non-logical symbols, where

- Ω, Π are the sets of function, predicate symbols
- Υ is a set of sort symbols
- υ is a function assigning sorts to function, predicate and variable symbols

Terms, Atoms,Formulae

Well-sorted Terms of sort $S \in \Upsilon$ over Σ_Υ (resp., $\mathrm{T}^S_{\Sigma_\Upsilon}(X)$-terms) are formed according to these syntactic rules:

$$\begin{array}{rcll} s, t, u, v & ::= & x & , x \in X, \upsilon(x) = S \text{ (variable)} \\ & \mid & f(t_1, ..., t_n) & , f \in \Omega, \mathsf{arity}(f) = n, \upsilon(f) = T_1 \ldots T_n S, \\ & & & t_i \in \mathrm{T}^{T_i}_{\Sigma_\Upsilon}(X) \text{ (functional term)} \end{array}$$

By $\mathrm{T}^S_{\Sigma_\Upsilon}$ we denote the set of Σ_Υ-ground terms of sort S, $\mathrm{T}_{\Sigma_\Upsilon}(X) = \bigcup_{S \in \Upsilon} \mathrm{T}^S_{\Sigma_\Upsilon}(X)$.

If $P \in \Pi$, $t_i \in \mathrm{T}^{T_i}_{\Sigma_\Upsilon}(X)$, $\upsilon(P) = T_1 \ldots T_n$ then $P(t_1, ..., t_n)$ is an atom. For any $t, s \in \mathrm{T}^S_{\Sigma_\Upsilon}(X)$, $s \approx t$ is an atom.

Formulae are build as for standard (unsorted) first-order logic.

For substitions we additionally require that if $x\sigma = t$ then $t \in \mathrm{T}^{\upsilon(x)}_{\Sigma_\Upsilon}(X)$ and call it *well-sorted*. Note that application of the standard unification algorithms to any two terms of the same sort yields a well-sorted unifier (if there exists a unifier at all).

Many-Sorted Structures

A Σ_Υ-*algebra* is a quadruple

$$\begin{array}{rl}\mathcal{A} = & (U_\mathcal{A},\ (f_\mathcal{A} : (T_1)_\mathcal{A} \times \ldots \times (T_n)_\mathcal{A} \to S_\mathcal{A})_{f\in\Omega},\\ & \quad (p_\mathcal{A} \subseteq (S_1)_\mathcal{A} \times \ldots \times (S_m)_\mathcal{A})_{p\in\Pi},\\ & \quad (T_\mathcal{A} \subseteq U_\mathcal{A})_{T\in\Upsilon})\end{array}$$

where $\mathsf{arity}(f) = n$, $\mathsf{arity}(p) = m$, $\upsilon(f) = T_1 \ldots T_n S$, $\upsilon(p) = S_1 \ldots S_m$, $T_\mathcal{A} \neq \emptyset$, $U_\mathcal{A} \neq \emptyset$ is a set, called the *universe* of $\mathcal{A}$.

The rest of the semantics is identical to the unsorted case, except that valuations respect the sort information.

7 SUP(LA)

Superposition Modulo Linear Arithmetic

- Consider the base specification $\mathrm{SP} = (\Sigma_{\mathrm{LA}}, \mathcal{A}_{\mathrm{LA}})$, where $\Sigma_{\mathrm{LA}} = (\mathbb{Q} \cup \{+, -, *\}, \{\geq, \leq, >, <\})$ see Section 2.
- The hierarchic extension of SP is $\mathrm{SP}' = (\Sigma', N')$, where $\Sigma_{\mathrm{LA}} \subseteq \Sigma'$ and N' is a set of Σ' clauses.
- We consider a many-sorted setting, consisting of a base sort, containing all terms of Σ_{LA} plus potentially extension terms from $\Sigma' \setminus \Sigma_{\mathrm{LA}}$, and a general sort containing all other terms.
- A term (a clause) consisting only of Σ_{LA} symbols and base sort variables, is called a *base term* (*base clause*).
- For the following results, we need that $\mathcal{A}_{\mathrm{LA}}$ is *term-generated*, i.e., for any $a \in U_{\mathrm{LA}}$ $(= \mathbb{Q})$ there is a ground term $t \in \mathrm{T}_{\Sigma_{\mathrm{LA}}}$ with $\mathcal{A}_{\mathrm{LA}}(t) = a$. This is obvious, because $\mathbb{Q} \subseteq \Sigma_{\mathrm{LA}}$.
- Furthermore, we need that $\mathrm{SP} = (\Sigma_{\mathrm{LA}}, \mathcal{A}_{\mathrm{LA}})$ is compact.
- A model of $\mathcal{A}'$ of SP', i.e., $\mathcal{A}' \models N'$, is called *hierarchic* if $\mathcal{A}' |_{\Sigma_{\mathrm{LA}}} = \mathcal{A}_{\mathrm{LA}}$.
- A substitution is called *simple* if it maps variables of the base sort to base terms.

Hierarchic Clauses

A clause $C = \Lambda \parallel C'$ is called *hierarchic* if Λ only contains base terms and base literals (Σ_{LA}) and all base terms in C' are variables. The semantics of C is $\bigwedge \Lambda \to C'$.

Any clause can be equivalently transformed into a hierarchic clause: whenever a subterm t whose top symbol is a base theory symbol occurs immediately below a non-base operator symbol, it is replaced by a new base sort variable x ("abstracted out") and the equation $x \approx t$ is added to Λ. Analogously, if a subterm t whose top symbol is not a base theory symbol occurs immediately below a base operator symbol, it is replaced by a general variable y and the disequation $y \not\approx t$ is added to C'. This transformation is repeated until the clause is hierarchic.

Superposition Modulo LA

Pos. Superposition:
$$\frac{\Lambda_1 \parallel D' \vee t \approx t' \qquad \Lambda_2 \parallel C' \vee s[u] \approx s'}{(\Lambda_1, \Lambda_2 \parallel D' \vee C' \vee s[t'] \approx s')\sigma}$$
where $\sigma = \mathrm{mgu}(t, u)$ and simple and u is not a variable.

Neg. Superposition:
$$\frac{\Lambda_1 \parallel D' \vee t \approx t' \qquad \Lambda_2 \parallel C' \vee s[u] \not\approx s'}{(\Lambda_1, \Lambda_2 \parallel D' \vee C' \vee s[t'] \not\approx s')\sigma}$$
where $\sigma = \mathrm{mgu}(t, u)$ and simple and u is not a variable.

Equality Resolution:
$$\frac{\Lambda \parallel C' \vee s \not\approx s'}{(\Lambda \parallel C')\sigma}$$
where $\sigma = \mathrm{mgu}(s, s')$ and simple.

Equality Factoring:
$$\frac{\Lambda \parallel C' \vee s' \approx t' \vee s \approx t}{(\Lambda \parallel C' \vee t \not\approx t' \vee s \approx t')\sigma}$$
where $\sigma = \mathrm{mgu}(s, s')$ and simple.

Constraint Refutation:
$$\frac{\Lambda_1 \parallel \Box \qquad \ldots \qquad \Lambda_n \parallel \Box}{\Box}$$
where $\neg(\bigwedge \Lambda_1) \wedge \ldots \wedge \neg(\bigwedge \Lambda_n)$ is inconsistent in $\mathcal{A}_{\mathrm{LA}}$.

Redundancy

A clause $C \in N$ is called *redundant* if for all simple ground instances C' of C there are simple ground instances $C'_1, \dots, C'_n$ from N such that $C'_1, \dots, C'_n \models C'$ and $C'_i \prec C'$ for all i.

A hierarchic clause $\Lambda \parallel C$ is called a *tautology* if C is a tautology or the existential closure of $\bigwedge \Lambda$ is unsatisfiable in $\mathcal{A}_{\mathrm{LA}}$.

A hierarchic clause $\Lambda_1 \parallel C_1$ *subsumes* a hierarchic clause $\Lambda_2 \parallel C_2$, if there is a simple matcher σ such that $C_1\sigma \subset C_2$ and the universal closure of $\bigwedge \Lambda_2 \rightarrow \bigwedge \Lambda_1\sigma$ holds in $\mathcal{A}_{\mathrm{LA}}$.

Sufficient Completeness

A set N of clauses is called *sufficiently complete with respect to simple instances*, if for every model $\mathcal{A}'$ of the set of simple ground instances from N and every ground non-base term t of the base sort there exists a ground base term t such that $t' \approx t$ is true in $\mathcal{A}'$.

Completeness of SUP(LA)

The hierarchic superposition calculus modulo LA is refutationally complete for all sets of clauses that are sufficiently complete with respect to simple instances.

The End