Automated Reasoning*

Uwe Waldmann

Summer Term 2011

Topics of the Course

Preliminaries

abstract reduction systems
well-founded orderings

Propositional logic

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

First-order predicate logic

syntax, semantics, model theory, ...
calculi: resolution, tableaux, ...
implementation: sharing, indexing

First-order predicate logic with equality

term rewriting systems
calculi: Knuth-Bendix completion, dependency pairs

*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.

1 Preliminaries

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

Througout the lecture, we will have to work with reduction systems,

on the object level, in particular in the section on equality,

and on the meta level, i. e., to describe deduction calculi.

1.1 Abstract Reduction Systems

An *abstract reduction system* is a pair $(A, \rightarrow)$, where

A is a set,

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

The relation $\rightarrow$ is usually written in infix notation, i. e., $a \rightarrow b$ instead of $(a, b) \in \rightarrow$.

Let $\rightarrow' \subseteq A \times B$ and $\rightarrow'' \subseteq B \times C$ be two binary relations. Then the binary relation $(\rightarrow' \circ \rightarrow'') \subseteq A \times C$ is defined by

$$a\ (\rightarrow' \circ \rightarrow'')\ c \quad \text{if and only if} \quad a \rightarrow' b \text{ and } b \rightarrow'' c \text{ for some } b \in B.$$

$\rightarrow^0$	$= \{\, (a, a) \mid a \in A \,\}$	*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).

A relation $\rightarrow$ is called

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.

Lemma 1.1 *If $\rightarrow$ is terminating, then it is normalizing.*

Note: The reverse implication does not hold.

1.2 Well-Founded Orderings

Termination of reduction systems is strongly related to 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 $\succ$ on M is called *well-founded (Noetherian)*, if there is no infinite descending chain $a_0 \succ a_1 \succ a_2 \succ \ldots$ with $a_i \in M$.

Well-Foundedness and Termination

Lemma 1.2 *If $>$ is a well-founded partial ordering and $\rightarrow \subseteq >$, then $\rightarrow$ is terminating.*

Lemma 1.3 *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 $\rightarrow$. $\square$

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

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

(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

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

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

Counterexamples:

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

Basic Properties of Well-Founded Orderings

Lemma 1.4 *$(M, \succ)$ is well-founded if and only if every $\emptyset \subset M' \subseteq M$ has a minimal element.*

Lemma 1.5 *$(M_1, \succ_1)$ and $(M_2, \succ_2)$ are well-founded 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. $\square$

Monotone Mappings

Let $(M_1, >_1)$ and $(M_2, >_2)$ be strict partial orderings. A mapping $\varphi : M_1 \to M_2$ is called *monotone*, if $a >_1 b$ implies $\varphi(a) >_2 \varphi(b)$ for all $a, b \in M_1$.

Lemma 1.6 *If φ is a monotone mapping from $(M_1, >_1)$ to $(M_2, >_2)$ and $(M_2, >_2)$ is well-founded, then $(M_1, >_1)$ is well-founded.*

Noetherian Induction

Theorem 1.7 (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 $m \in M$.

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.* □

Multisets

Let M be a set. A *multiset* 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 multiset S.

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

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 multisets, e. g.,

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

---

[1]induction hypothesis
[2]induction step

A multiset S is called *finite*, if

$$|\{\, m \in M \mid S(m) > 0 \,\}| < \infty.$$

From now on we only consider finite multisets.

Multiset Orderings

Lemma 1.8 (König's Lemma) *Every finitely branching tree with infinitely many nodes contains an infinite path.*

Let $(M, \succ)$ be a strict partial ordering. The *multiset extension* of $\succ$ to multisets over M is defined by

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

Theorem 1.9
(a) $\succ_{\mathrm{mul}}$ *is a strict partial ordering.*
(b) $\succ$ *well-founded* $\Rightarrow$ $\succ_{\mathrm{mul}}$ *well-founded.*
(c) $\succ$ *total* $\Rightarrow$ $\succ_{\mathrm{mul}}$ *total.*

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

2 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)

2.1 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:

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

Notational Conventions

- We may 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 left-associative,
    i. e., $F \vee G \vee H$ means $(F \vee G) \vee H$.
  - $\rightarrow$ is right-associative,
    i. e., $F \rightarrow G \rightarrow H$ means $F \rightarrow (G \rightarrow H)$.

2.2 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 $\mathcal{A}^* : \Sigma\text{-formulas} \rightarrow \{0, 1\}$ is defined inductively over the structure of F as follows:

$$\begin{aligned}
\mathcal{A}^*(\bot) &= 0 \\
\mathcal{A}^*(\top) &= 1 \\
\mathcal{A}^*(P) &= \mathcal{A}(P) \\
\mathcal{A}^*(\neg F) &= \mathsf{B}_\neg(\mathcal{A}^*(F)) \\
\mathcal{A}^*(F \,\rho\, G) &= \mathsf{B}_\rho(\mathcal{A}^*(F), \mathcal{A}^*(G)) \quad \text{for} \quad \rho \in \{\wedge, \vee, \rightarrow, \leftrightarrow\}
\end{aligned}$$

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.)

2.3 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*):

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

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 $\mathcal{A}$ we have $\mathcal{A} \models F \;\Rightarrow\; \mathcal{A} \models G$.

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

Proposition 2.1 *$F \models G$ if and only if $\models (F \rightarrow G)$.*

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 2.2 *$F \models\!\!\!\!\!\dashv G$ if and only if $\models (F \leftrightarrow G)$.*

Proof. Analogously to Prop. 2.1. $\square$

Entailment is extended 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 $\mathcal{A} \models F$.

Note: formulas are always finite objects; but sets of formulas may be infinite. Therefore, it is in general not possible to replace a set of formulas by the conjunction of its elements.

Validity vs. Unsatisfiability

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

Proposition 2.3 *F is valid if and only if $\neg F$ is unsatisfiable.*

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 2.4 *$N \models F$ 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 2.5 *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. 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 2.6 *The following equivalences are valid for all formulas F, G, H:*

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

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

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

2.4 Normal Forms

We define *conjunctions* of formulas as follows:

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

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

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

and analogously *disjunctions:*

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

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

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

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 2.7 *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 steps 2, 4, and 6; steps 1, 3, and 5 are a bit more complicated.

For step 1, we can prove termination in the following way: We define a function ϕ from formulas to positive integers such that $\phi(\bot) = \phi(\top) = \phi(P) = 1$, $\phi(\neg F) = \phi(F)$, $\phi(F \wedge G) = \phi(F \vee G) = \phi(F \rightarrow G) = \phi(F) + \phi(G)$, and $\phi(F \leftrightarrow G) = 2\phi(F) + 2\phi(G) + 1$. Observe that ϕ is constructed in such a way that $\phi(F) > \phi(G)$ implies $\phi(H[F]) > \phi(H[G])$ for all formulas F, G, and H. Using this property, we can show that whenever a formula H' is the result of applying the rule of step 1 to a formula H, then $\phi(H) > \phi(H')$. Since ϕ takes only positive integer values, step 1 must terminate.

Termination of steps 3 and 5 is proved similarly. For step 3, we use 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 positive integer values, step 3 must terminate.

For step 5, we use 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 positive integer values, 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 conjunctions have to be pushed downward in step 5. $\Box$

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 2.8 *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. □

2.5 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.

2.6 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).

In particular, after learning a unit clause a restart is done.

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:

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

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

Decide:

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

if L is undefined under M and contained in N.

Fail:

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

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

Backjump:

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

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 $M\, L^{\mathrm{d}}\, M'$. (But usually there are better choices.)

Lemma 2.9 *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 2.10 *Every derivation starting from $\emptyset \parallel N$ terminates.*

Proof. (Idea) Consider a DPLL derivation step $M \parallel N \;\Rightarrow_{\mathrm{DPLL}}\; M' \parallel N'$ and a decomposition $M_0 L_1^d M_1 \ldots L_k^d M_k$ 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 or equal to n. We define $f(M) = n - \mathrm{length}(M)$ and finally

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

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

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

Lemma 2.11 *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. (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 2.9, $N \cup \{L_1, \dots, L_n\} \models \neg D$. Since $D \in N$, this is a contradiction, so $N \cup \{L_1, \dots, L_n\}$ is unsatisfiable. Consequently, $N \models \overline{L_1} \vee \cdots \vee \overline{L_n}$. 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. $\square$

Theorem 2.12 *(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. (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 2.9, every literal in M follows from N. On the other hand, $C \in N$, so N must be unsatisfiable. $\square$

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.

Then $N \models D \vee C'$ and $D \vee C'$ is also false under M. ($D \vee C'$ is a *resolvent* of $C' \vee \overline{L}$ and $D \vee L$.)

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.

Preprocessing

Some transformations are not performed during the DPLL search, but only in a preprocessing step:

(i) Subsumption

$N \cup \{C\} \cup \{D\} \ \Rightarrow \ N \cup \{C\}$

if $C \subseteq D$ considering C, D as multisets of literals.

(ii) Purity Deletion

(iii) Merging Replacement Resolution

$N \cup \{C \vee L\} \cup \{D \vee \overline{L}\} \ \Rightarrow \ N \cup \{C \vee L\} \cup \{D\}$

if $C \subseteq D$ considering C, D as multisets of literals.

(iv) Tautology Deletion

(v) Literal Elimination: do all possible resolution steps on a literal and throw away the parent clauses

Further Information

The ideas described so far heve been implemented in all modern SAT solvers: *zChaff*, *miniSAT*,*picoSAT*. Because of clause learning the algorithm is now called CDCL: Conflict Driven Clause Learning.

It has been shown in 2009 that CDCL can polynomially simulate resolution, a long standing open question:

Knot Pipatsrisawat, Adnan Darwiche: On the Power of Clause-Learning SAT Solvers with Restarts. CP 2009, 654-668

Literature

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: Solving 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.

Armin Biere, Marijn Heule, Hans van Maaren, Toby Walsh (eds.): Handbook of Satisfiability; IOS Press, 2009

Daniel Le Berre's slides at VTSA'09: `http://www.mpi-inf.mpg.de/vtsa09/`.

2.7 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 square (i, j) we generate $p^1_{i,j} \lor \ldots \lor p^9_{i,j}$
- For every square (i, j) and pair of values $d < d'$ we generate $\neg p^d_{i,j} \lor \neg p^{d'}_{i,j}$
- For every value d and column i we generate $p^d_{i,1} \lor \ldots \lor p^d_{i,9}$
  (Analogously for rows and 3×3 boxes)
- For every value d, column i, and pair of rows $j < j'$ we generate $\neg p^d_{i,j} \lor \neg p^d_{i,j'}$
  (Analogously for 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 $\neg p^3_{1,7} \lor \neg p^3_{5,7}$ and $p^3_{1,7}$ we obtain by unit propagating $\neg p^3_{5,7}$ and further from $p^1_{5,7} \lor p^2_{5,7} \lor p^3_{5,7} \lor p^4_{5,7} \lor \ldots \lor p^9_{5,7}$ we get $p^1_{5,7} \lor p^2_{5,7} \lor p^4_{5,7} \lor \ldots \lor p^9_{5,7}$ (and finally $p^7_{5,7}$).

2.8 Other Calculi

OBDDs (Ordered Binary Decision Diagrams):

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

⇒ canonical representation of formulas.

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.

⇒ semi-canonical representation of formulas.

Implementation needs DPLL (and OBDDs) as subroutines.

Ordered resolution
Tableau calculus
Hilbert calculus
Sequent calculus
Natural deduction

see next chapter

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)
  $\Rightarrow$ terms, atomic formulas
- logical symbols (domain-independent)
  $\Rightarrow$ 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 $\text{arity}(f) = n$,
- Π is a set of *predicate symbols* P with arity $m \geq 0$, written $\text{arity}(P) = m$.

Function symbols are also called *operator symbols*.
If $n = 0$ then f is also called a *constant (symbol)*.
If $m = 0$ then P is also called a *propositional variable*.

We will usually use

b, c, d for constant symbols,

f, g, h for non-constant function symbols,

P, Q, R, S for predicate symbols.

Convention: We will usually write $f/n \in \Omega$ instead of $f \in \Omega$, $\text{arity}(f) = n$ (analogously for predicate symbols).

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 to denote *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 and X (Σ-terms) are formed according to these syntactic rules:

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

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:

$$\begin{array}{lllr} A, B & ::= & P(s_1, \ldots, s_m) \ , \ P/m \in \Pi & \text{(non-equational atom)} \\ & \Big[\ \mid & (s \approx t) & \text{(equation)} \ \Big] \end{array}$$

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 are much more efficient.

Literals

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

Clauses

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

General First-Order Formulas

$\mathrm{F}_\Sigma(X)$ is the set of first-order formulas over Σ defined as follows:

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

Notational Conventions

We omit brackets according to the following rules:

- $Qx \quad >_p \quad \neg \quad >_p \quad \vee \quad >_p \quad \wedge \quad >_p \quad \rightarrow \quad >_p \quad \leftrightarrow$
  (binding precedences)
- $\vee$ and $\wedge$ are left-associative
- $\rightarrow$ 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.

Positions in Terms and Formulas

Positions of a term s (formula F):

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

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

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

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

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

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

Analogously for formulas F/p.

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

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

Analogously for formulas $F[G]_p$, $F[t]_p$.

Size of a term s:

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

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

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

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

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

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

Substitutions are often written as $\{x_1 \mapsto s_1, \dots, x_n \mapsto s_n\}$, with x_i pairwise distinct, and then denote the mapping

$$\{x_1 \mapsto s_1, \dots, x_n \mapsto s_n\}(y) = \begin{cases} s_i, & \text{if } y = x_i \\ y, & \text{otherwise} \end{cases}$$

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

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

with $\mathit{apply}(F, \sigma)$ 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:

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

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

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

$$\mathcal{A} = (U_\mathcal{A},\ (f_\mathcal{A} : U_\mathcal{A}^n \to U_\mathcal{A})_{f/n\in\Omega},\ (P_\mathcal{A} \subseteq U_\mathcal{A}^m)_{P/m\in\Pi})$$

where $U_\mathcal{A} \neq \emptyset$ 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:

$$\begin{aligned} \mathcal{A}(\beta)(x) &= \beta(x), & x \in X \\ \mathcal{A}(\beta)(f(s_1,\dots,s_n)) &= f_\mathcal{A}(\mathcal{A}(\beta)(s_1),\dots,\mathcal{A}(\beta)(s_n)), & f/n \in \Omega \end{aligned}$$

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:

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

Example

The "Standard" Interpretation for Peano Arithmetic:

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

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

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

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*):

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

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 $\mathcal{A} \in \Sigma$-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 $\mathcal{A} \in \Sigma$-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 $\mathcal{A} \in \Sigma$-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 $F \land \neg G$ 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 $\mathcal{A} \in \Sigma$-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

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 $\Rightarrow_P$:

$$\begin{array}{rcll}
(F \leftrightarrow G) & \Rightarrow_P & (F \rightarrow G) \wedge (G \rightarrow F) & \\
\neg Q x F & \Rightarrow_P & \overline{Q} x \neg F & (\neg Q) \\
((Q x F) \; \rho \; G) & \Rightarrow_P & Q y (F\{x \mapsto y\} \; \rho \; G), \;\; \rho \in \{\wedge, \vee\} & \\
((Q x F) \rightarrow G) & \Rightarrow_P & \overline{Q} y (F\{x \mapsto y\} \rightarrow G), & \\
(F \; \rho \; (Q x G)) & \Rightarrow_P & Q y (F \; \rho \; G\{x \mapsto y\}), \;\; \rho \in \{\wedge, \vee, \rightarrow\} &
\end{array}$$

Here y is always assumed to be some fresh variable and $\overline{Q}$ 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):

$$\forall x_1, \dots, x_n \exists y F \quad \Rightarrow_S \quad \forall x_1, \dots, x_n F\{y \mapsto f(x_1, \dots, x_n)\}$$

where f/n is a new function symbol (*Skolem function*).

Together: $F \Rightarrow_P^* \underbrace{G}_{\text{prenex}} \Rightarrow_S^* \underbrace{H}_{\text{prenex, no } \exists}$

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)

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

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

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

$N = \{C_1, \dots, C_k\}$ 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

The normal form algorithm described so far leaves lots of room for optimization. Note that we only can preserve satisfiability anyway due to Skolemization.

- size of the CNF is exponential when done naively; the transformations we introduced already for propositional logic avoid this exponential growth;
- we want to preserve the original formula structure;
- we want small arity of Skolem functions (see next section).

3.6 Getting Small Skolem Functions

A clause set that is better suited for automated theorem proving can be obtained using the following steps:

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

Negation Normal Form (NNF)

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

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

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:

$$
\begin{array}{rll}
Qx\,(G \land F) & \Rightarrow_{MS} & Qx\, G \land F \\
Qx\,(G \lor F) & \Rightarrow_{MS} & Qx\, G \lor F \\
\forall x\,(G \land H) & \Rightarrow_{MS} & \forall x\, G \land \forall x\, H \\
\exists x\,(G \lor H) & \Rightarrow_{MS} & \exists x\, G \lor \exists x\, H
\end{array}
$$

Variable Renaming

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

Standard Skolemization

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

$$\begin{aligned} \exists x\, H \quad \Rightarrow_{SK} \quad & H\{x \mapsto f(y_1, \ldots, y_n)\} \\ & \text{if } F/p = \exists x\, H \text{ and } p \text{ has minimal length,} \\ & \{y_1, \ldots, y_n\} \text{ are the free variables in } \exists x\, H, \\ & f/n \text{ is a new function symbol} \end{aligned}$$

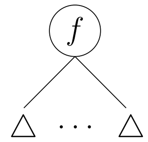
3.7 Herbrand Interpretations

From now on we shall consider FOL without equality. We assume that Ω 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)
- $f_{\mathcal{A}} : (s_1, \ldots, s_n) \mapsto f(s_1, \ldots, s_n),\ f/n \in \Omega$

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



In other words, *values are fixed* to be ground terms and *functions are fixed* to be the *term constructors*. Only predicate symbols $P/m \in \Pi$ 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*

$$(s_1, \ldots, s_n) \in P_{\mathcal{A}} \quad :\Leftrightarrow \quad P(s_1, \ldots, s_n) \in I$$

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}:

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

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.*

$$
\begin{aligned}
N \textit{ satisfiable} \quad & \Leftrightarrow \quad N \textit{ has a Herbrand model (over } \Sigma) \\
& \Leftrightarrow \quad G_\Sigma(N) \textit{ has a Herbrand model (over } \Sigma)
\end{aligned}
$$

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*, 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

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 finitely many clauses $F_1, \ldots, F_n \in N$ such that $F_1, \ldots, F_n \vdash_\Gamma F$*

Proofs as Trees

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

$$
\dfrac{P(f(c)) \qquad \dfrac{\dfrac{\dfrac{P(f(c)) \vee Q(b) \qquad \dfrac{\dfrac{P(f(c)) \vee Q(b) \quad \neg P(f(c)) \vee \neg P(f(c)) \vee Q(b)}{\neg P(f(c)) \vee Q(b) \vee Q(b)}}{\neg P(f(c)) \vee Q(b)}}{Q(b) \vee Q(b)}}{Q(b)} \qquad \neg P(f(c)) \vee \neg Q(b)}{\neg P(f(c))}}{\bot}
$$

3.9 Propositional Resolution

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

In this section we only deal with ground clauses.

The Resolution Calculus *Res*

Resolution inference rule:

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

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

(Positive) factorisation inference rule:

$$\frac{C \vee A \vee A}{C \vee A}$$

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

We treat "$\vee$" as associative and commutative, hence A and $\neg A$ can occur anywhere in the clauses; moreover, when we write $C \vee A$, etc., this includes unit clauses, that is, $C = \bot$.

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.	$\bot$	(Res. 4. into 9.)

Resolution with Implicit Factorization RIF

Factorization can be included in the resolution rule:

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

Sample refutation for RIF:

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.	$\bot$	(Res. 4. into 7.)

Soundness of Resolution

Theorem 3.15 *Propositional resolution is sound.*

Proof. Let $\mathcal{B} \in \Sigma$-Alg. To be shown:

(i) for resolution: $\mathcal{B} \models D \vee A,\ \mathcal{B} \models C \vee \neg A \ \Rightarrow\ \mathcal{B} \models D \vee C$

(ii) for factorization: $\mathcal{B} \models C \vee A \vee A \ \Rightarrow\ \mathcal{B} \models C \vee A$

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

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

1. $\mathcal{B} \models L_1 \vee \ldots \vee L_n \ \Leftrightarrow\ $ there exists i: $\mathcal{B} \models L_i$.
2. $\mathcal{B} \models A$ or $\mathcal{B} \models \neg A$.

This does not hold for formulas with variables!

3.10 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 multiset 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:

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

Properties of the Clause Ordering

Proposition 3.16

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

$$\begin{array}{cr|l|l}
 & & & \\ \hline
 & B\left\{\begin{array}{l} \\ \\ \\ \end{array}\right. & \begin{array}{l} \neg B \vee \ldots \\ \ldots \vee B \vee B \\ \ldots \vee B \end{array} & \text{all clauses } D \text{ with } \max(D) = B \\ \hline
\curlyvee & & \quad\vdots & \\ \hline
 & A\left\{\begin{array}{l} \\ \\ \\ \end{array}\right. & \begin{array}{l} \neg A \vee \ldots \\ \ldots \vee A \vee A \\ \ldots \vee A \end{array} & \text{all clauses } C \text{ with } \max(C) = A \\ \hline
 & & &
\end{array}$$

Closure of Clause Sets under Res

$$\begin{aligned}
Res(N) &= \{\, C \mid C \text{ is conclusion of an inference in } Res \\
&\qquad\qquad \text{with premises in } N \,\} \\
Res^0(N) &= N \\
Res^{n+1}(N) &= Res(Res^n(N)) \cup Res^n(N), \text{ for } n \geq 0 \\
Res^*(N) &= \textstyle\bigcup_{n \geq 0} Res^n(N)
\end{aligned}$$

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

Proposition 3.17

(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.

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*).

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 $\succ$:

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

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

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.)

Example

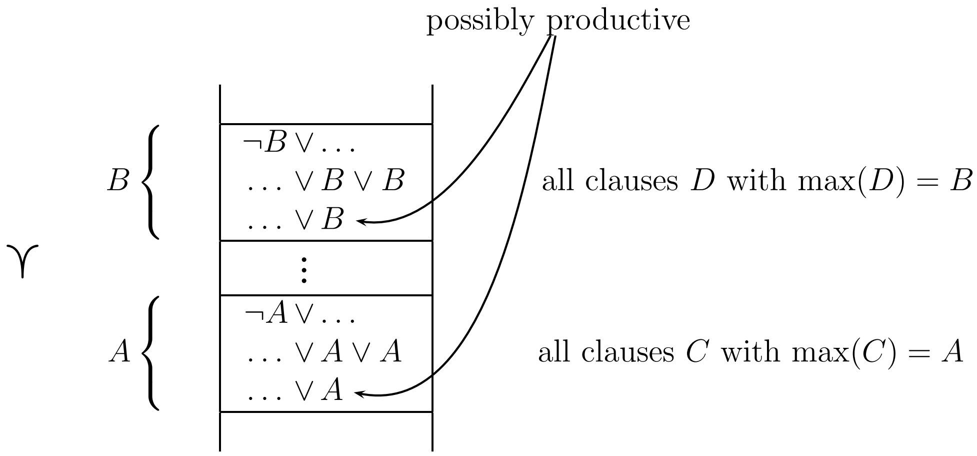
Let $A_5 \succ A_4 \succ A_3 \succ A_2 \succ A_1 \succ A_0$ (max. literals in red)

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

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

Structure of $N, \succ$

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



Some Properties of the Construction

Proposition 3.18

(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 $A \Rightarrow I_N \not\models C$.

Resolution Reduces Counterexamples

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

Construction of I for the extended clause set:

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

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

Factorization Reduces Counterexamples

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

Construction of I for the extended clause set:

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

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

Model Existence Theorem

Theorem 3.19 (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 3.20 *Let N be saturated w. r. t. Res. Then $N \models \bot \Leftrightarrow \bot \in N$.*

Proof of Theorem 3.19. 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. Since there is an inference

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

we infer that $D' \vee C' \in N$, and $C \succ D' \vee C'$ and $I_N \not\models D' \vee C'$. This contradicts the minimality of C.

Case 2: $C = C' \vee A \vee A$. There is an inference

$$\frac{C' \vee A \vee A}{C' \vee A}$$

that yields a smaller counterexample $C' \vee A \in N$. This contradicts the minimality of C.
$\square$

Compactness of Propositional Logic

Theorem 3.21 (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.
$\Rightarrow$ $\mathit{Res}^*(N)$ unsatisfiable
$\Rightarrow$ $\bot \in \mathit{Res}^*(N)$ by refutational completeness of resolution
$\Rightarrow$ $\exists n \geq 0 : \bot \in \mathit{Res}^n(N)$
$\Rightarrow$ $\bot$ has a finite resolution proof P;
choose M as the set of assumptions in P. $\square$

3.11 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)

inferior to the DPLL procedure.

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

General Resolution through Instantiation

Idea: instantiate clauses appropriately:

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

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.

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

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 (mgu).

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:

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

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

General resolution RIF with implicit factorization:

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

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 multiset 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 $\mathrm{mgu}(E)$.

Proposition 3.22

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

Rule-Based Naive Standard Unification

$$
\begin{array}{rcl}
t \doteq t, E & \Rightarrow_{SU} & E \\
f(s_1, \ldots, s_n) \doteq f(t_1, \ldots, t_n), E & \Rightarrow_{SU} & s_1 \doteq t_1, \ldots, s_n \doteq t_n, E \\
f(\ldots) \doteq g(\ldots), E & \Rightarrow_{SU} & \bot \\
x \doteq t, E & \Rightarrow_{SU} & x \doteq t, E\{t \mapsto x\} \\
 & & \text{if } x \in var(E), x \notin var(t) \\
x \doteq t, E & \Rightarrow_{SU} & \bot \\
 & & \text{if } x \neq t, x \in var(t) \\
t \doteq x, E & \Rightarrow_{SU} & x \doteq t, E \\
 & & \text{if } t \notin X
\end{array}
$$

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 $\sigma_E = \{x_1 \mapsto u_1, \ldots, x_k \mapsto u_k\}$.

Proposition 3.24 *If E is a solved form then σ_E is an mgu of E.*

Theorem 3.25

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, $\sigma \circ \{x \mapsto t\} = \sigma[x \mapsto t\sigma] = \sigma[x \mapsto x\sigma] = \sigma$. Therefore, for any equation $u \doteq v$ in E: $u\sigma = v\sigma$, iff $u\{x \mapsto t\}\sigma = v\{x \mapsto t\}\sigma$. (2) and (3) follow by induction from (1) using Proposition 3.24. □

Main Unification Theorem

Theorem 3.26 *E is unifiable if and only if there is a most general unifier σ of E, such that σ is idempotent and $dom(\sigma) \cup codom(\sigma) \subseteq var(E)$.*

Proof.

- $\Rightarrow_{SU}$ 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 multiset 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 $t \notin X$.

- A system E that is irreducible w. r. t. $\Rightarrow_{SU}$ is either $\bot$ 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 3.25, Proposition 3.24).
- 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

Problem: using $\Rightarrow_{SU}$, an *exponential growth* of terms is possible.

The following unification algorithm avoids this problem, at least if the final solved form is represented as a DAG.

$$
\begin{array}{rcl}
t \doteq t, E & \Rightarrow_{PU} & E \\
f(s_1, \ldots, s_n) \doteq f(t_1, \ldots, t_n), E & \Rightarrow_{PU} & s_1 \doteq t_1, \ldots, s_n \doteq t_n, E \\
f(\ldots) \doteq g(\ldots), E & \Rightarrow_{PU} & \bot \\
x \doteq y, E & \Rightarrow_{PU} & x \doteq y, E\{x \mapsto y\} \\
 & & \text{if } x \in var(E), x \neq y \\
x_1 \doteq t_1, \ldots, x_n \doteq t_n, E & \Rightarrow_{PU} & \bot \\
 & & \text{if there are positions } p_i \text{ with} \\
 & & t_i/p_i = x_{i+1}, t_n/p_n = x_1 \\
 & & \text{and some } p_i \neq \epsilon \\
x \doteq t, E & \Rightarrow_{PU} & \bot \\
 & & \text{if } x \neq t, x \in var(t) \\
t \doteq x, E & \Rightarrow_{PU} & x \doteq t, E \\
 & & \text{if } t \notin X \\
x \doteq t, x \doteq s, E & \Rightarrow_{PU} & x \doteq t, t \doteq s, E \\
 & & \text{if } t, s \notin X \text{ and } |t| \leq |s|
\end{array}
$$

Properties of PU

Theorem 3.27

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

Note: The solved form of $\Rightarrow_{PU}$ is different form the solved form obtained from $\Rightarrow_{SU}$. In order to obtain the unifier $\sigma_{E'}$, we have to sort the list of equality problems $x_i \doteq t_i$ in such a way that x_i does not occur in t_j for $j < i$, and then we have to compose the substitutions $\{x_1 \mapsto t_1\} \circ \cdots \circ \{x_k \mapsto t_k\}$.

Lifting Lemma

Lemma 3.28 *Let C and D be variable-disjoint clauses. If*

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

then there exists a substitution τ such that

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

An analogous lifting lemma holds for factorization.

Saturation of Sets of General Clauses

Corollary 3.29 *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 3.30 *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 3.31 *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 3.32 (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 $N \not\models \bot$.

$$
\begin{aligned}
N \not\models \bot &\Rightarrow \bot \notin Res^*(N) \qquad \text{(resolution is sound)} \\
&\Rightarrow \bot \notin G_\Sigma(Res^*(N)) \\
&\Rightarrow I_{G_\Sigma(Res^*(N))} \models G_\Sigma(Res^*(N)) \qquad \text{(Thm. 3.19; Cor. 3.29)} \\
&\Rightarrow I_{G_\Sigma(Res^*(N))} \models Res^*(N) \qquad \text{(Lemma 3.31)} \\
&\Rightarrow I_{G_\Sigma(Res^*(N))} \models N \qquad (N \subseteq Res^*(N)) \quad \square
\end{aligned}
$$

The Theorem of Löwenheim-Skolem

Theorem 3.33 (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 3.32. $\square$

Refutational Completeness of General Resolution

Theorem 3.34 *Let N be a set of general clauses where $Res(N) \subseteq N$. Then*

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

Proof. Let $Res(N) \subseteq N$. By Corollary 3.29: $Res(G_\Sigma(N)) \subseteq G_\Sigma(N)$

$$\begin{aligned} N \models \bot &\Leftrightarrow G_\Sigma(N) \models \bot && \text{(Lemma 3.30/3.31; Theorem 3.32)} \\ &\Leftrightarrow \bot \in G_\Sigma(N) && \text{(propositional resolution sound and complete)} \\ &\Leftrightarrow \bot \in N \quad \square \end{aligned}$$

Compactness of Predicate Logic

Theorem 3.35 (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 3.34, $\bot \in Res^*(N)$, 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. $\square$

3.12 Ordered Resolution with Selection

Motivation: Search space for *Res* *very* large.

Ideas for improvement:

1. In the completeness proof (Model Existence Theorem 3.19) one only needs to resolve and factor maximal atoms
   $\Rightarrow$ if the calculus is restricted to inferences involving maximal atoms, the proof remains correct
   $\Rightarrow$ *ordering 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 $\boxed{X}$:

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

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

Intuition:

- If a clause has at least one selected literal, compute only inferences that involve a selected literal.
- If a clause has no selected literals, compute only inferences that involve a maximal literal.

Resolution Calculus $Res_S^{\succ}$

The resolution calculus $Res_S^{\succ}$ is parameterized by

- a selection function S
- and a total and well-founded atom ordering $\succ$.

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:

A literal L is called *[strictly] maximal* in a clause C if and only if there exists a ground substitution σ such that $L\sigma$ is [strictly] maximal in $C\sigma$ (i.e., if for no other L' in C: $L\sigma \prec L'\sigma$ $[L\sigma \preceq L'\sigma]$).

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

if the following conditions are satisfied:

(i) $\sigma = \mathrm{mgu}(A, B)$;

(ii) $B\sigma$ strictly maximal in $D\sigma \vee B\sigma$;

(iii) nothing is selected in $D \vee B$ by S;

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

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

if the following conditions are satisfied:

(i) $\sigma = \mathrm{mgu}(A, B)$;

(ii) $A\sigma$ is maximal in $C\sigma \vee A\sigma \vee B\sigma$;

(iii) nothing is selected in $C \vee A \vee B$ by S.

Special Case: Propositional Logic

For ground clauses the resolution inference rule simplifies to

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

if the following conditions are satisfied:

(i) $A \succ D$;

(ii) nothing is selected in $D \vee A$ 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 $A \succ \max(D)$.

Analogously, the factorization rule simplifies to

$$\frac{C \vee A \vee A}{C \vee A}$$

if the following conditions are satisfied:

(i) A is the largest literal in $C \vee A \vee A$;

(ii) nothing is selected in $C \vee A \vee A$ by S.

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	$\perp$	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

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

we can obtain by *rotation*

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

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 rotation-equivalent proofs.

Lifting Lemma for $Res_S^{\succ}$

Lemma 3.36 *Let D and C be variable-disjoint clauses. If*

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

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

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

An analogous lifting lemma holds for factorization.

Saturation of General Clause Sets

Corollary 3.37 *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 Cor. 3.29 using the above lifting lemma. □

Soundness and Refutational Completeness

Theorem 3.38 *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 3.37. □

Craig-Interpolation

A theoretical application of ordered resolution is Craig-Interpolation:

Theorem 3.39 (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. $\square$

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 3.40 *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 3.41 *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 3.38. □

Monotonicity Properties of Redundancy

Theorem 3.42

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

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

We conclude that redundancy is preserved when, during a theorem proving process, one adds (derives) new clauses or deletes redundant clauses. Recall that $Red(N)$ may include clauses that are not in N.

3.13 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 3.41 and 3.42 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*

$$N \cup \{C \vee A \vee \neg A\} \;\Rightarrow\; N$$

- *Deletion of subsumed clauses*

$$N \cup \{C, D\} \;\Rightarrow\; N \cup \{C\}$$

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

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

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

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

$N \cup \{C\} \mid P \mid O \quad \Rightarrow_{RP} \quad N \mid P \mid O$

if C is a tautology

Forward subsumption

$N \cup \{C\} \mid P \mid O \quad \Rightarrow_{RP} \quad N \mid P \mid O$

if some $D \in P \cup O$ subsumes C

Backward subsumption

$N \cup \{C\} \mid P \cup \{D\} \mid O \quad \Rightarrow_{RP} \quad N \cup \{C\} \mid P \mid O$

$N \cup \{C\} \mid P \mid O \cup \{D\} \quad \Rightarrow_{RP} \quad N \cup \{C\} \mid P \mid O$

if C strictly subsumes D

Transition Rules for RP (II)

Forward reduction

$N \cup \{C \vee L\} \mid P \mid O \quad \Rightarrow_{RP} \quad N \cup \{C\} \mid P \mid O$

if there exists $D \vee L' \in P \cup O$

such that $\overline{L} = L'\sigma$ and $D\sigma \subseteq C$

Backward reduction

$N \mid P \cup \{C \vee L\} \mid O \quad \Rightarrow_{RP} \quad N \mid P \cup \{C\} \mid O$

$N \mid P \mid O \cup \{C \vee L\} \quad \Rightarrow_{RP} \quad N \mid P \cup \{C\} \mid O$

if there exists $D \vee L' \in N$

such that $\overline{L} = L'\sigma$ and $D\sigma \subseteq C$

Transition Rules for RP (III)

Clause processing

$N \cup \{C\} \mid P \mid O \quad \Rightarrow_{RP} \quad N \mid P \cup \{C\} \mid O$

Inference computation

$\emptyset \mid P \cup \{C\} \mid O \quad \Rightarrow_{RP} \quad N \mid P \mid O \cup \{C\}$,

where N is the set of conclusions of $Res_S^{\succ}$-inferences from clauses in $O \cup \{C\}$ where C is one of the premises

Soundness and Completeness

Theorem 3.43

$$N \models \bot \quad \Leftrightarrow \quad N \mid \emptyset \mid \emptyset \quad \stackrel{*}{\Rightarrow}_{RP} \quad N' \cup \{\bot\} \mid \_ \mid \_$$

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

Fairness

Problem:

If N is inconsistent, then $N \mid \emptyset \mid \emptyset \stackrel{*}{\Rightarrow}_{RP} N' \cup \{\bot\} \mid \_ \mid \_$.

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

No: a clause could be kept in 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 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 $Res^{\succ}_{S}$, 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 factorization 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 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 Semantic Tableaux

Literature:

M. Fitting: First-Order Logic and Automated Theorem Proving, Springer-Verlag, New York, 1996, chapters 3, 6, 7.

R. M. Smullyan: First-Order Logic, Dover Publ., New York, 1968, revised 1995.

Like resolution, semantic tableaux were developed in the sixties, independently by Zbigniew Lis and Raymond Smullyan on the basis of work by Gentzen in the 30s and of Beth in the 50s.

Idea

Idea (for the propositional case):

A set $\{F \wedge G\} \cup N$ of formulas has a model if and only if $\{F \wedge G,\ F,\ G\} \cup N$ has a model.

A set $\{F \vee G\} \cup N$ of formulas has a model if and only if $\{F \vee G,\ F\} \cup N$ or $\{F \vee G,\ G\} \cup N$ has a model.

(and similarly for other connectives).

To avoid duplication, represent sets as paths of a tree.

Continue splitting until two complementary formulas are found $\Rightarrow$ inconsistency detected.

A Tableau for $\{P \wedge \neg(Q \vee \neg R),\ \neg Q \vee \neg R\}$

1. $P \wedge \neg(Q \vee \neg R)$
2. $\neg Q \vee \neg R$

3. $\neg Q$	4. $\neg R$
5. P	10. P
6. $\neg(Q \vee \neg R)$	11. $\neg(Q \vee \neg R)$
7. $\neg Q$	
8. $\neg\neg R$	
9. R	

This tableau is not "maximal", however the first "path" is. This path is not "closed", hence the set $\{1, 2\}$ is satisfiable. (These notions will all be defined below.)

Properties

Properties of tableau calculi:

analytic: inferences according to the logical content of the symbols.

goal oriented: inferences operate directly on the goal to be proved (unlike, e. g., ordered resolution).

global: some inferences affect the entire proof state (set of formulas), as we will see later.

Propositional Expansion Rules

Expansion rules are applied to the formulas in a tableau and expand the tableau at a leaf. We append the conclusions of a rule (horizontally or vertically) at a *leaf*, whenever the premise of the expansion rule matches a formula appearing *anywhere* on the path from the root to that leaf.

Negation Elimination

$$\frac{\neg\neg F}{F} \qquad \frac{\neg\top}{\bot} \qquad \frac{\neg\bot}{\top}$$

α-Expansion

(for formulas that are essentially conjunctions: append subformulas α_1 and α_2 one on top of the other)

$$\frac{\alpha}{\begin{array}{l}\alpha_1\\ \alpha_2\end{array}}$$

β-Expansion

(for formulas that are essentially disjunctions:
append β_1 and β_2 horizontally, i. e., branch into β_1 and β_2)

$$\frac{\beta}{\beta_1 \mid \beta_2}$$

Classification of Formulas

conjunctive			disjunctive		
α	α_1	α_2	β	β_1	β_2
$X \wedge Y$	X	Y	$\neg(X \wedge Y)$	$\neg X$	$\neg Y$
$\neg(X \vee Y)$	$\neg X$	$\neg Y$	$X \vee Y$	X	Y
$\neg(X \rightarrow Y)$	X	$\neg Y$	$X \rightarrow Y$	$\neg X$	Y

We assume that the binary connective $\leftrightarrow$ has been eliminated in advance.

Tableaux: Notions

A *semantic tableau* is a marked (by formulas), finite, unordered tree and inductively defined as follows: Let $\{F_1, \ldots, F_n\}$ be a set of formulas.

(i) The tree consisting of a single path

$$\begin{array}{c} F_1 \\ \vdots \\ F_n \end{array}$$

is a tableau for $\{F_1, \ldots, F_n\}$. (We do not draw edges if nodes have only one successor.)

(ii) If T is a tableau for $\{F_1, \ldots, F_n\}$ and if T' results from T by applying an expansion rule then T' is also a tableau for $\{F_1, \ldots, F_n\}$.

A *path* (from the root to a leaf) in a tableau is called *closed*, if it either contains $\perp$, or else it contains both some formula F and its negation $\neg F$. Otherwise the path is called *open*.

A tableau is called *closed*, if all paths are closed.

A *tableau proof* for F is a closed tableau for $\{\neg F\}$.

A path P in a tableau is called *maximal*, if for each non-atomic formula F on P there exists a node in P at which the expansion rule for F has been applied.

In that case, if F is a formula on P, P also contains:

(i) F_1 and F_2, if F is a α-formula,

(ii) F_1 or F_2, if F is a β-formula, and

(iii) F', if F is a negation formula, and F' the conclusion of the corresponding elimination rule.

A tableau is called *maximal*, if each path is closed or maximal.

A tableau is called *strict*, if for each formula the corresponding expansion rule has been applied at most once on each path containing that formula.

A tableau is called *clausal*, if each of its formulas is a clause.

A Sample Proof

One starts out from the negation of the formula to be proved.

$$
\begin{array}{lcr}
1. & \neg[(P \rightarrow (Q \rightarrow R)) \rightarrow ((P \vee S) \rightarrow ((Q \rightarrow R) \vee S))] & \\
2. & (P \rightarrow (Q \rightarrow R)) & [1_1] \\
3. & \neg((P \vee S) \rightarrow ((Q \rightarrow R) \vee S)) & [1_2] \\
4. & P \vee S & [3_1] \\
5. & \neg((Q \rightarrow R) \vee S)) & [3_2] \\
6. & \neg(Q \rightarrow R) & [5_1] \\
7. & \neg S & [5_2]
\end{array}
$$

$$8. \ \neg P \quad [2_1] \qquad\qquad 9. \ Q \rightarrow R \quad [2_2]$$

$$10. \ P \quad [4_1] \qquad\qquad 11. \ S \quad [4_2]$$

There are three paths, each of them closed.

Properties of Propositional Tableaux

We assume that T is a tableau for $\{F_1, \ldots, F_n\}$.

Theorem 3.44 *$\{F_1, \ldots, F_n\}$ satisfiable $\Leftrightarrow$ some path (i. e., the set of its formulas) in T is satisfiable.*

Proof. By induction over the structure of T. □

Corollary 3.45 *T closed $\Rightarrow$ $\{F_1, \ldots, F_n\}$ unsatisfiable*

Theorem 3.46 *Let T be a strict propositional tableau. Then T is finite.*

Proof. New formulas resulting from expansion are either $\bot$, $\top$ or subformulas of the expanded formula. By strictness, on each path a formula can be expanded at most once. Therefore, each path is finite, and a finitely branching tree with finite paths is finite by Lemma 1.8. □

Conclusion: Strict and maximal tableaux can be effectively constructed.

Refutational Completeness

Theorem 3.47 *Let P be a maximal, open path in a tableau. Then set of formulas on P is satisfiable.*

Proof. (The theorem holds for arbitrary tableaux, but in this proof we consider only the case of a clausal tableau. The full proof can be found, e.g., in Fitting 1996.)

Let N be the set of formulas on P. As P is open, $\bot$ is not in N. Let $C \vee A$ and $D \vee \neg A$ be two resolvable clauses in N. One of the two subclauses C or D, C say, is not empty, as otherwise P would be closed. Since P is maximal, in P the β-rule was applied on $C \vee A$. Therefore, P (and N) contains a proper subclause of $C \vee A$, and hence $C \vee A$ is redundant w. r. t. N. By the same reasoning, if N contains a clause that can be factored, that clause must be redundant w. r. t. N. In other words, N is saturated up to redundancy w. r. t. *Res*(olution). Now apply Theorem 3.19 to prove satisfiability of N. □

Theorem 3.48 $\{F_1, \ldots, F_n\}$ *satisfiable* $\Leftrightarrow$ *there exists no closed strict tableau for* $\{F_1, \ldots, F_n\}$.

Proof. One direction is clear by Theorem 3.44. For the reverse direction, let T be a strict, maximal tableau for $\{F_1, \ldots, F_n\}$ and let P be an open path in T. By the previous theorem, the set of formulas on P, and hence by Theorem 3.44 the set $\{F_1, \ldots, F_n\}$, is satisfiable. □

Consequences

The validity of a propositional formula F can be established by constructing a strict, maximal tableau for $\{\neg F\}$:

- T closed $\Leftrightarrow$ F valid.
- It suffices to test complementarity of paths w. r. t. atomic formulas (cf. reasoning in the proof of Theorem 3.47).
- Which of the potentially many strict, maximal tableaux one computes does not matter. In other words, tableau expansion rules can be applied don't-care non-deterministically ("*proof confluence*").
- The expansion strategy, however, can have a dramatic impact on tableau size.
- Since it is sufficient to saturate paths w. r. t. ordered resolution (up to redundancy), tableau expansion rules can be even more restricted, in particular by certain ordering constraints.

Semantic Tableaux for First-Order Logic

Additional classification of quantified formulas:

universal		existential	
γ	$\gamma(t)$	δ	$\delta(t)$
$\forall x F$	$F[t/x]$	$\exists x F$	$F[t/x]$
$\neg\exists x F$	$\neg F[t/x]$	$\neg\forall x F$	$\neg F[t/x]$

Moreover we assume that the set of variables X is partitioned into 2 disjoint infinite subsets X_g and X_f, so that bound [free] variables variables can be chosen from X_g [X_f]. (This avoids the variable capturing problem.)

Additional Expansion Rules

γ-expansion

$$\frac{\gamma}{\gamma(x)} \quad \text{where } x \text{ is a variable in } X_f$$

δ-expansion

$$\frac{\delta}{\delta(f(x_1, \ldots, x_n))}$$

where f is a *new* Skolem function, and the x_i are the free variables in δ

Skolemization becomes part of the calculus and needs not necessarily be applied in a preprocessing step. Of course, one could do Skolemization beforehand, and then the δ-rule would not be needed.

Note that the rules are parametric, instantiated by the choices for x and f, respectively. Strictness here means that only one instance of the rule is applied on each path to any formula on the path.

In this form the rules go back to Hähnle and Schmitt: The liberalized δ-rule in free variable semantic tableaux, J. Automated Reasoning 13,2, 1994, 211–221.

Free-Variable Tableaux

Let $\{F_1, \ldots, F_n\}$ be a set of *closed formulas*.

(i) The tree consisting of a single path

$$\begin{array}{c} F_1 \\ \vdots \\ F_n \end{array}$$

is a tableau for $\{F_1, \ldots, F_n\}$.

(ii) If T is a tableau for $\{F_1, \ldots, F_n\}$ and if T' results by applying an expansion rule to T, then T' is also a tableau for $\{F_1, \ldots, F_n\}$.

(iii) If T is a tableau for $\{F_1, \ldots, F_n\}$ and if σ is a substitution, then $T\sigma$ is also a tableau for $\{F_1, \ldots, F_n\}$.

The *substitution rule* (iii) may, potentially, modify all the formulas of a tableau. This feature is what is makes the tableau method a *global proof method*. (Resolution, by comparison, is a local method.)

If one took (iii) literally, by repeated application of γ-rule one could enumerate all substitution instances of the universally quantified formulas. That would be a major drawback compared with resolution. Fortunately, we can improve on this.

Example

$$\begin{array}{rll}
1. & \neg[\exists w \forall x\ p(x, w, f(x, w)) \rightarrow \exists w \forall x \exists y\ p(x, w, y)] & \\
2. & \exists w \forall x\ p(x, w, f(x, w)) & 1_1\ [\alpha] \\
3. & \neg \exists w \forall x \exists y\ p(x, w, y) & 1_2\ [\alpha] \\
4. & \forall x\ p(x, c, f(x, c)) & 2(c)\ [\delta] \\
5. & \neg \forall x \exists y\ p(x, v_1, y) & 3(v_1)\ [\gamma] \\
6. & \neg \exists y\ p(b(v_1), v_1, y) & 5(b(v_1))\ [\delta] \\
7. & p(v_2, c, f(v_2, c)) & 4(v_2)\ [\gamma] \\
8. & \neg p(b(v_1), v_1, v_3) & 6(v_3)\ [\gamma]
\end{array}$$

7. and 8. are complementary (modulo unification):

$$v_2 \doteq b(v_1),\ c \doteq v_1,\ f(v_2, c) \doteq v_3$$

is solvable with an mgu $\sigma = [c/v_1, b(c)/v_2, f(b(c), c)/v_3]$, and hence, $T\sigma$ is a closed (linear) tableau for the formula in 1.

AMGU-Tableaux

Idea: Restrict the substitution rule to unifiers of complementary formulas.

We speak of an *AMGU-Tableau*, whenever the substitution rule is only applied for substitutions σ for which there is a path in T containing two *literals* $\neg A$ and B such that $\sigma = \mathrm{mgu}(A, B)$.

Correctness

Given an signature Σ, by Σ^{sko} we denote the result of adding infinitely many new Skolem function symbols which we may use in the δ-rule.

Let $\mathcal{A}$ be a Σ^{sko}-interpretation, T a tableau, and β a variable assignment over $\mathcal{A}$.

T is called $(\mathcal{A}, \beta)$-*valid*, if there is a path P_β in T such that $\mathcal{A}, \beta \models F$, for each formula F on P_β.

T is called *satisfiable* if there exists a structure $\mathcal{A}$ such that for each assignment β the tableau T is $(\mathcal{A}, \beta)$-valid. (This implies that we may choose P_β depending on β.)

Theorem 3.49 *Let T be a tableau for $\{F_1, \ldots, F_n\}$, where the F_i are closed Σ-formulas. Then $\{F_1, \ldots, F_n\}$ is satisfiable $\Leftrightarrow$ T is satisfiable.*

Proof. Proof of "$\Rightarrow$" by induction over the depth of T. For δ one needs to reuse the ideas for proving that Skolemization preserves [un-]satisfiability. $\square$

Incompleteness of Strictness

Strictness for γ is incomplete:

$$
\begin{array}{lcr}
1. & \neg[\forall x\; p(x) \rightarrow (p(c) \wedge p(b))] & \\
2. & \forall x\; p(x) & 1_1 \\
3. & \neg(p(c) \wedge p(b)) & 1_2 \\
4. & p(v_1) & 2(v_1)
\end{array}
$$

$$
5.\ \neg p(c) \quad 3_1 \qquad\qquad 6.\ \neg p(b) \quad 3_2
$$

If we placed a strictness requirement also on applications of γ, the tableau would only be expandable by the substitution rule. However, there is no substitution (for v_1) that can close both paths simultaneously.

Multiple Application of γ Solves the Problem

$$
\begin{array}{lcr}
1. & \neg[\forall x\, p(x) \rightarrow (p(c) \wedge p(b))] & \\
2. & \forall x\, p(x) & 1_1 \\
3. & \neg(p(c) \wedge p(b)) & 1_2 \\
4. & p(v_1) & 2_{v_1}
\end{array}
$$

$$
\begin{array}{lll}
5. & \neg p(c) & 3_1
\end{array}
\qquad\qquad
\begin{array}{lll}
6. & \neg p(b) & 3_2 \\
7. & p(v_2) & 2_{v_2}
\end{array}
$$

The point is that different applications of γ to $\forall x\, p(x)$ may employ different free variables for x.

Now, by two applications of the AMGU-rule, we obtain the substitution $[c/v_1, b/v_2]$ closing the tableau.

Therefore *strictness for* γ should from now on mean that each *instance* of γ (depending on the choice of the free variable) is applied at most once to each γ-formula on any path.

Refutational Completeness

Theorem 3.50 $\{F_1, \ldots, F_n\}$ *satisfiable* $\Leftrightarrow$ *there exists no closed, strict AMGU-Tableau for* $\{F_1, \ldots, F_n\}$.

For the proof one defines a fair tableau expansion process converging against an infinite tableau where on each path each γ-formula is expanded into all its variants (modulo the choice of the free variable).

One may then again show that each path in that tableau is saturated (up to redundancy) by resolution. This requires to apply the lifting lemma for resolution in order to show completeness of the AMGU-restriction.

How Often Do we Have to Apply γ?

Theorem 3.51 *There is no recursive function $f : F_\Sigma \times F_\Sigma \to \mathbb{N}$ such that, if the closed formula F is unsatisfiable, then there exists a closed tableau for F where to all formulas $\forall x G$ appearing in T the γ-rule is applied at most $f(F, \forall x G)$ times on each path containing $\forall x G$.*

Otherwise unsatisfiability or, respectively, validity for first-order logic would be decidable. In fact, one would be able to enumerate in finite time all tableaux bounded in depth as indicated by f. In other words, free-variable tableaux are not recursively bounded in their depth.

Again $\forall$ is treated like an infinite conjunction. By repeatedly applying γ, together with the substitution rule, one can enumerate all instances $F[t/x]$ vertically, that is, conjunctively, in each path containing $\forall x F$.

Semantic Tableaux vs. Resolution

- Tableaux: global, goal-oriented, "backward".
- Resolution: local, "forward".
- Goal-orientation is a clear advantage if only a small subset of a large set of formulas is necessary for a proof. (Note that resolution provers saturate also those parts of the clause set that are irrelevant for proving the goal.)
- Resolution can be combined with more powerful redundancy elimination methods; because of its global nature this is more difficult for the tableau method.
- Resolution can be refined to work well with equality; for tableaux this seems to be impossible.
- On the other hand tableau calculi can be easily extended to other logics; in particular tableau provers are very successful in modal and description logics.

3.16 Other Inference Systems

- Instantiation-based methods
  Resolution-based instance generation
  Disconnection calculus
  ...
- Natural deduction
- Sequent calculus/Gentzen calculus
- Hilbert calculus

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].

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

$$F_1, \dots, F_m \vdash G_1, \dots, G_k$$

meaning

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

A kind of mixture between natural deduction and semantic tableaux.

Inferences rules, e.g.:

$$\frac{\Gamma \vdash \Delta}{\Gamma, F \vdash \Delta}\ (WL) \qquad \frac{\Gamma, F \vdash \Delta \qquad \Sigma, G \vdash \Pi}{\Gamma, \Sigma, F \vee G \vdash \Delta, \Pi}\ (\vee L)$$

$$\frac{\Gamma \vdash \Delta}{\Gamma \vdash F, \Delta}\ (WR) \qquad \frac{\Gamma \vdash F, \Delta \qquad \Sigma \vdash G, \Pi}{\Gamma, \Sigma \vdash F \wedge G, \Delta, \Pi}\ (\wedge R)$$

Perfect symmetry between the handling of assumptions and their consequences.

Can be used both backwards and forwards.

Hilbert Calculus

Hilbert calculus:

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

Axiom schemes, e. g.,

$$F \rightarrow (G \rightarrow F)$$
$$(F \rightarrow (G \rightarrow H)) \rightarrow ((F \rightarrow G) \rightarrow (F \rightarrow H))$$

plus Modus ponens:

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

Unsuitable for finding or reading proofs, but sometimes used for *specifying* (e.g. modal) logics.

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 any prover for first-order logic without equality:

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*

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

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 $\mathcal{B} = (U_{\mathcal{B}},\ (f_{\mathcal{B}} : U^n \rightarrow U)_{f \in \Omega},\ (p_{\mathcal{B}} \subseteq U_{\mathcal{B}}^m)_{p \in \Pi \cup \{\sim\}})$ 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 $1 = \mathcal{B}(\beta[x \mapsto a, y \mapsto a'])(x \sim y \rightarrow y \sim x) = (a \sim_{\mathcal{B}} a' \rightarrow a' \sim_{\mathcal{B}} a)$, 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. □

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).

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

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

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

Roadmap

How to proceed:

- This semester: Equations (unit clauses with equality)

  Term rewrite systems
  Expressing semantic consequence syntactically
  Knuth-Bendix-Completion
  Entailment for equations

- Next semester: Equational clauses

  Combining resolution and KB-completion → Superposition
  Entailment for clauses with equality

4.2 Rewrite Systems

Let E be a set of (implicitly universally quantified) equations.

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

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

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 universally quantified equations. A model of E is also called an *E-algebra*.

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:

$E \vdash t \approx t$ *(Reflexivity)*

$$\frac{E \vdash t \approx t'}{E \vdash t' \approx t} \quad \textit{(Symmetry)}$$

$$\frac{E \vdash t \approx t' \qquad E \vdash t' \approx t''}{E \vdash t \approx t''} \quad \textit{(Transitivity)}$$

$$\frac{E \vdash t_1 \approx t_1' \quad \ldots \quad E \vdash t_n \approx t_n'}{E \vdash f(t_1,\ldots,t_n) \approx f(t_1',\ldots,t_n')} \quad \textit{(Congruence)}$$

$E \vdash t\sigma \approx t'\sigma$ *(Instance)*
if $(t \approx t') \in E$ and $\sigma : X \to \mathrm{T}_\Sigma(X)$

Lemma 4.2 *The following properties are equivalent:*

(i) $s \leftrightarrow_E^* t$

(ii) $E \vdash s \approx t$ *is derivable.*

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 4.3 $f_\mathcal{T}$ *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 4.4 $\mathcal{T} = \mathrm{T}_\Sigma(X)/E$ *is an* E*-algebra.*

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 $\sigma = \{x_1 \mapsto t_1, \dots, x_n \mapsto t_n\}$, 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 $\mathcal{T}(\gamma)(s) = [s\sigma] = [t\sigma] = \mathcal{T}(\gamma)(t)$. □

Lemma 4.5 *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. 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 4.6 ("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)⇔(ii): Lemma 4.2.

(ii)⇒(iii): By induction on the size of the derivation for $E \vdash s \approx t$.

(iii)⇒(iv): Obvious, since $\mathcal{T} = \mathrm{T}_\Sigma(X)/E$ is an E-algebra.

(iv)⇒(ii): Lemma 4.5. $\square$

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 $x + y \not\approx_E y + x$.

4.3 Confluence

Let $(A, \rightarrow)$ be an abstract reduction system.

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

The 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 $b \downarrow c$.

convergent, if it is confluent and terminating.

Theorem 4.7 *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 4.8 *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. □

Corollary 4.9 *If* $\rightarrow$ *is normalizing and confluent, then every element* b *has a unique normal form.*

Proposition 4.10 *If* $\rightarrow$ *is normalizing and confluent, then* $b \leftrightarrow^* c$ *if and only if* $b\!\downarrow = c\!\downarrow$.

Proof. Either using Thm. 4.7 or directly by induction on the length of the derivation of $b \leftrightarrow^* c$. □

Confluence and Local Confluence

Theorem 4.11 ("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. □

Rewrite Relations

Corollary 4.12 *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 4.13 *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 $\text{var}(l_1) \cap \text{var}(l_2) = \emptyset$. (Remember that $\text{var}(l_i) \supseteq \text{var}(r_i)$.)

Let $p \in \text{pos}(l_1)$ 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 4.14 ("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 $p_i \in \text{pos}(s)$, 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: $p_1 \,||\, p_2$.

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: $p_2 = p_1 q_1 q_2$, 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 $p_1 q' q_2$, 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 $p_1 q q_2$, 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: $p_2 = p_1 p$, 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 4.15 *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 4.16 *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 4.17 *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 4.18 *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 $p \in \mathrm{pos}(t)$.*

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 4.19 *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": $s \to_R s'$ 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 $\to_R \subseteq \succ$. Since $\succ$ is a well-founded ordering, $\to_R$ is terminating.

"only if": Define $\succ = \to_R^+$. If $\to_R$ is terminating, then $\succ$ is a reduction ordering. □

Two Different Scenarios

Depending on the application, the TRS whose termination we want to show can be

(i) fixed and known in advance, or

(ii) evolving (e.g., generated by some saturation process).

Methods for case (ii) are also usable for case (i).
Many methods for case (i) are not usable for case (ii).

We will first consider case (ii);
additional techniques for case (i) will be considered later.

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 $\beta : X \to U_{\mathcal{A}}$.

Is $\succ_{\mathcal{A}}$ a reduction ordering?

Lemma 4.20 *$\succ_{\mathcal{A}}$ 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 $\beta : X \to U_{\mathcal{A}}$. Let σ be a substitution. We have to show that $\mathcal{A}(\gamma)(s\sigma) \succ \mathcal{A}(\gamma)(s'\sigma)$ for all assignments $\gamma : X \to U_{\mathcal{A}}$. 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 $f : U_{\mathcal{A}}^n \to U_{\mathcal{A}}$ 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 4.21 *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 $\beta : X \to U_{\mathcal{A}}$. Let $\beta : X \to U_{\mathcal{A}}$ 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 4.22 *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 $\mathbb{N}$ or some subset of $\mathbb{N}$.

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:

$U_{\mathcal{A}} = \{ n \in \mathbb{N} \mid n \geq 1 \}$.

If arity$(f) = 0$, then P_f is a constant ≥ 1.

If arity$(f) = n \geq 1$, then P_f is a polynomial $P(X_1, \ldots, X_n)$, 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 $x_1, \ldots, x_n$ yields a polynomial P_t with indeterminates $X_1, \ldots, X_n$ (where X_i corresponds to $\beta(x_i)$).

Example:

$\Omega = \{b/0,\ f/1,\ g/3\}$
$P_b = 3, \quad P_f(X_1) = X_1^2, \quad P_g(X_1, X_2, X_3) = X_1 + X_2 X_3.$

Let $t = g(f(b), f(x), y)$, then $P_t(X, Y) = 9 + X^2 Y$.

If P, Q are polynomials in $\mathbb{N}[X_1, \ldots, X_n]$, we write $P > Q$ if $P(a_1, \ldots, a_n) > Q(a_1, \ldots, a_n)$ for all $a_1, \ldots, a_n \in U_{\mathcal{A}}$.

Clearly, $l \succ_{\mathcal{A}} r$ iff $P_l > P_r$ iff $P_l - P_r > 0$.

Question: Can we check $P_l - P_r > 0$ automatically?

Hilbert's 10th Problem:

Given a polynomial $P \in \mathbb{Z}[X_1, \ldots, X_n]$ with integer coefficients, is $P = 0$ for some n-tuple of natural numbers?

Theorem 4.23 *Hilbert's 10th Problem is undecidable.*

Proposition 4.24 *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. $\square$

One easy case:

If we restrict to linear polynomials, deciding whether $P_l - P_r > 0$ is trivial:

$\sum k_i a_i + k > 0$ for all $a_1, \ldots, a_n \geq 1$ if and only if

$k_i \geq 0$ for all $i \in \{1, \ldots, n\}$,

and $\sum k_i + k > 0$

Another possible solution:

Test whether $P_l(a_1, \ldots, a_n) > P_r(a_1, \ldots, a_n)$ for all $a_1, \ldots, a_n \in \{ x \in \mathbb{R} \mid x \geq 1 \}$.

This is decidable (but hard). Since $U_{\mathcal{A}} \subseteq \{ x \in \mathbb{R} \mid x \geq 1 \}$, it implies $P_l > P_r$.

Alternatively:

Use fast overapproximations.

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 $R_{\text{emb}} = \{ f(x_1, \ldots, x_n) \to x_i \mid f \in \Omega,\, 1 \leq i \leq n = \operatorname{arity}(f) \}$.

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 4.25 *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, $s = s[l\sigma] \succ s[r\sigma] = t$. $\square$

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 4.26 ("Kruskal's Theorem") *Let Σ be a finite signature, let X be a finite set of variables. Then for every infinite sequence $t_1, t_2, t_3, \ldots$ there are indices $j > i$ such that $t_j \trianglerighteq_{\text{emb}} t_i$. ($\trianglerighteq_{\text{emb}}$ is called* a well-partial-ordering (wpo).*)*

Proof. See Baader and Nipkow, page 113–115. $\square$

Theorem 4.27 (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 $\sigma = \{x \mapsto t_i\}$, 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 $t_i \trianglelefteq_{\text{emb}} t_j$. Hence $t_i \preceq t_j$, contradicting $t_i \succ t_j$. $\square$

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 $f(g(f(x))) \trianglerighteq_{\text{emb}} f(f(x))$ implies $f(g(f(x))) \succeq f(f(x))$, hence $f(f(x)) \succ f(f(x))$.

Path Orderings

Let $\Sigma = (\Omega, \Pi)$ be a finite signature, let $\succ$ be a strict partial ordering *("precedence")* on Ω.

The *lexicographic path ordering* $\succ_{\text{lpo}}$ 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 $(s_1, \ldots, s_m)\ (\succ_{\text{lpo}})_{\text{lex}}\ (t_1, \ldots, t_n)$.

Lemma 4.28 *$s \succ_{\text{lpo}} t$ implies* $\text{var}(s) \supseteq \text{var}(t)$.

Proof. By induction on $|s| + |t|$ and case analysis. □

Theorem 4.29 *$\succ_{\text{lpo}}$ 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. □

Theorem 4.30 *If the precedence $\succ$ is total, then the lexicographic path ordering $\succ_{\text{lpo}}$ is total on ground terms, i. e., for all $s, t \in \mathrm{T}_\Sigma(\emptyset)$: $s \succ_{\text{lpo}} t \lor t \succ_{\text{lpo}} s \lor s = t$.*

Proof. By induction on $|s| + |t|$ and case analysis. □

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) $t \in \text{var}(s)$ and $t \neq s$, or

(2) $s = f(s_1, \dots, s_m)$, $t = g(t_1, \dots, t_n)$, 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 $(s_1, \dots, s_m)\ (\succ_{\text{lpo}})_{\text{lex}}\ (t_1, \dots, t_n)$.

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 arity$(n) \geq 1$ associate a status $\in \{mul\} \cup \{\, lex_\pi \mid \pi : \{1, \dots, n\} \to \{1, \dots, n\}\,\}$ 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 $\text{arity}(f) = 1$, then $f \succeq g$ for all $g \in \Omega$.

The weight function w can be extended to terms as follows:

$$w(t) = \sum_{x \in \text{var}(t)} w(x) \cdot \#(x, t) + \sum_{f \in \Omega} w(f) \cdot \#(f, t).$$

The *Knuth-Bendix ordering* $\succ_{\text{kbo}}$ on $\mathrm{T}_\Sigma(X)$ induced by $\succ$ and w is defined by: $s \succ_{\text{kbo}} t$ 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) $s = f(s_1, \ldots, s_m)$, $t = g(t_1, \ldots, t_n)$, and $f \succ g$, or

  (c) $s = f(s_1, \ldots, s_m)$, $t = f(t_1, \ldots, t_m)$, and $(s_1, \ldots, s_m)\ (\succ_{\text{kbo}})_{\text{lex}}\ (t_1, \ldots, t_m)$.

Theorem 4.31 *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. □

Remark

If $\Pi \neq \emptyset$, then all the term orderings described in this section can also be used to compare non-equational atoms by treating predicate symbols like function symbols.

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

$s \mathrel{\underset{\sim}{\sqsupset}} l$ if $s/p = l\sigma$ for some p and σ

and $\sqsupset \; = \; \mathrel{\underset{\sim}{\sqsupset}} \setminus \mathrel{\underset{\sim}{\sqsubset}}$ is the strict part of $\mathrel{\underset{\sim}{\sqsupset}}$.

Lemma 4.32 *$\sqsupset$ is a well-founded strict partial ordering.*

Lemma 4.33 *If $E, R \vdash E', R'$, then $\approx_{E \cup R} \, = \, \approx_{E' \cup R'}$.*

Lemma 4.34 *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) $E_0, R_0 \vdash E_1, R_1 \vdash E_2, R_2 \vdash \ldots$ 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:

$E_0, R_0 \vdash E_1, R_1 \vdash E_2, R_2 \vdash \ldots$ 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 $1 \le i \le k$ and $s_{i-1} \leftarrow_{R_*} s_i$ for $k+1 \le i \le n$

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 4.35 $\succ_C$ *is a well-founded ordering.*

Lemma 4.36 *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:
$\dots s_{i-1} \leftarrow_{R_*} s_i \rightarrow_{R_*} s_{i+1} \dots \Longrightarrow \dots s_{i-1} \rightarrow^*_{R_*} s' \leftarrow^*_{R_*} s_{i+1} \dots$

If there is a critical pair that has been added using *Deduce*:
$\dots s_{i-1} \leftarrow_{R_*} s_i \rightarrow_{R_*} s_{i+1} \dots \Longrightarrow \quad \dots s_{i-1} \leftrightarrow_{E_\infty} s_{i+1} \dots$

In all cases, checking that the replacement subproof is smaller than the replaced subproof is routine. $\square$

Theorem 4.37 *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. $\square$

4.7 Unfailing Completion

Classical completion:

Try to transform a set E of equations into an equivalent convergent TRS.

Fail, if an equation can neither be oriented nor deleted.

Unfailing completion (Bachmair, Dershowitz and Plaisted):

If an equation cannot be oriented, we can still use *orientable instances* for rewriting.

Note: If $\succ$ is total on ground terms, then every *ground instance* of an equation is trivial or can be oriented.

Goal: Derive a *ground convergent* set of equations.

Let E be a set of equations, let $\succ$ be a reduction ordering.

We define the relation $\rightarrow_{E^\succ}$ by

$$
\begin{array}{lll}
s \rightarrow_{E^\succ} t & \text{iff} & \text{there exist } (u \approx v) \in E \text{ or } (v \approx u) \in E, \\
 & & p \in \mathrm{pos}(s), \text{ and } \sigma : X \rightarrow \mathrm{T}_\Sigma(X), \\
 & & \text{such that } s/p = u\sigma \text{ and } t = s[v\sigma]_p \textit{ and } u\sigma \succ v\sigma.
\end{array}
$$

Note: $\rightarrow_{E^\succ}$ is terminating by construction.

From now on let $\succ$ be a reduction ordering that is total on ground terms.

E is called ground convergent w. r. t. $\succ$, if for all ground terms s and t with $s \leftrightarrow_E^* t$ there exists a ground term v such that $s \rightarrow_{E^\succ}^* v \leftarrow_{E^\succ}^* t$. (Analogously for $E \cup R$.)

As for standard completion, we establish ground convergence by computing critical pairs.

However, the ordering $\succ$ is not total on non-ground terms. Since $s\theta \succ t\theta$ implies $s \not\preceq t$, we approximate $\succ$ on ground terms by $\not\preceq$ on arbitrary terms.

Let $u_i \stackrel{.}{\approx} v_i$ $(i = 1, 2)$ be equations in E whose variables have been renamed such that $\mathrm{var}(u_1 \stackrel{.}{\approx} v_1) \cap \mathrm{var}(u_2 \stackrel{.}{\approx} v_2) = \emptyset$. Let $p \in \mathrm{pos}(u_1)$ be a position such that u_1/p is not a variable, σ is an mgu of u_1/p and u_2, and $u_i\sigma \not\preceq v_i\sigma$ $(i = 1, 2)$. Then $\langle v_1\sigma, (u_1\sigma)[v_2\sigma]_p\rangle$ is called a *semi-critical pair* of E with respect to $\succ$.

The set of all semi-critical pairs of E is denoted by $\mathrm{SP}_\succ(E)$.

Semi-critical pairs of $E \cup R$ are defined analogously. If $\rightarrow_R \subseteq \succ$, then $\mathrm{CP}(R)$ and $\mathrm{SP}_\succ(R)$ agree.

Note: In contrast to critical pairs, it may be necessary to consider overlaps of a rule with itself at the top. For instance, if $E = \{f(x) \approx g(y)\}$, then $\langle g(y), g(y')\rangle$ is a non-trivial semi-critical pair.

The *Deduce* rule takes now the following form:

Deduce:

$$\frac{E, \quad R}{E \cup \{s \approx t\}, \quad R} \qquad \text{if } \langle s, t\rangle \in \mathrm{SP}_{\succ}(E \cup R).$$

Moreover, the fairness criterion for runs is replaced by

$$\mathrm{SP}_{\succ}(E_* \cup R_*) \subseteq E_\infty$$

(i. e., if every semi-critical pair between persisting rules or equations is computed at some step of the derivation).

Analogously to Thm. 4.37 we obtain now the following theorem:

Theorem 4.38 *Let $E_0, R_0 \vdash E_1, R_1 \vdash E_2, R_2 \vdash \ldots$ be a fair run; let $R_0 = \emptyset$. Then*

(1) $E_* \cup R_*$ is equivalent to E_0, and

(2) $E_* \cup R_*$ is ground convergent.

Moreover one can show that, whenever there exists a *reduced* convergent R such that $\approx_{E_0} = \downarrow_R$ and $\rightarrow_R \in \succ$, then for every fair *and simplifying* run $E_* = \emptyset$ and $R_* = R$ up to variable renaming.

Here R is called reduced, if for every $l \rightarrow r \in R$, both l and r are irreducible w. r. t. $R \setminus \{l \rightarrow r\}$. A run is called simplifying, if R_* is reduced, and for all equations $u \approx v \in E_*$, u and v are incomparable w. r. t. $\succ$ and irreducible w. r. t. R_*.

Unfailing completion is refutationally complete for equational theories:

Theorem 4.39 *Let E be a set of equations, let $\succ$ be a reduction ordering that is total on ground terms. For any two terms s and t, let $\hat{s}$ and $\hat{t}$ be the terms obtained from s and t by replacing all variables by Skolem constants. Let $eq/2$, $true/0$ and $false/0$ be new operator symbols, such that $true$ and $false$ are smaller than all other terms. Let $E_0 = E \cup \{eq(\hat{s}, \hat{t}) \approx true,\ eq(x, x) \approx false\}$. If $E_0, \emptyset \vdash E_1, R_1 \vdash E_2, R_2 \vdash \ldots$ be a fair run of unfailing completion, then $s \approx_E t$ iff some $E_i \cup R_i$ contains $true \approx false$.*

Outlook:

Combine ordered resolution and unfailing completion to get a calculus for equational clauses:

compute inferences between (strictly) maximal literals as in ordered resolution,
compute overlaps between maximal sides of equations as in unfailing completion

$\Rightarrow$ Superposition calculus.

5 Implementing Saturation Procedures

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; $\mathit{New} = \mathit{Infer}(\{ D_i \mid i \in I \}, C)$; $U = U \cup \mathit{New}$; $W = W \cup \{C\}$

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?

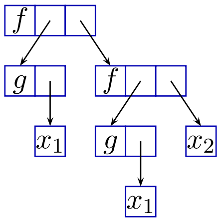
yes $\rightsquigarrow$ "Otter loop":
Advantage: simplifications of U may be useful to derive the empty clause.

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

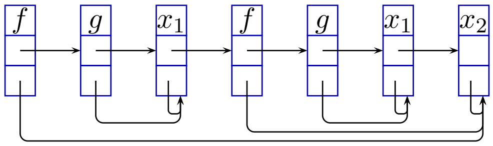
$f(g(x_1), f(g(x_1), x_2))$



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:

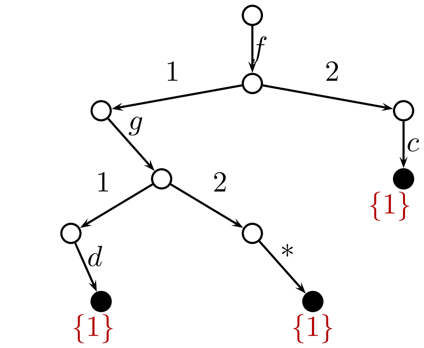
Paths of terms are encoded in a trie ("retrieval tree").

A star $*$ represents arbitrary variables.

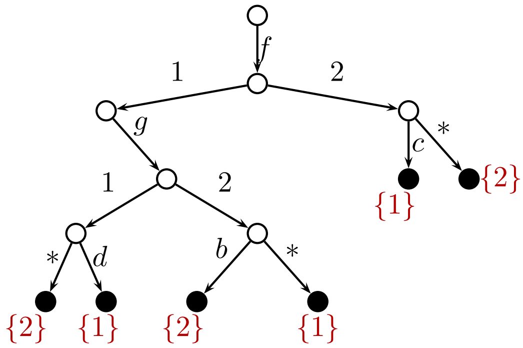
Example: Paths of $f(g(*, b), *)$: $f.1.g.1.*$
$f.1.g.2.b$
$f.2.*$

Each leaf of the trie contains the set of (pointers to) all terms that contain the respective path.

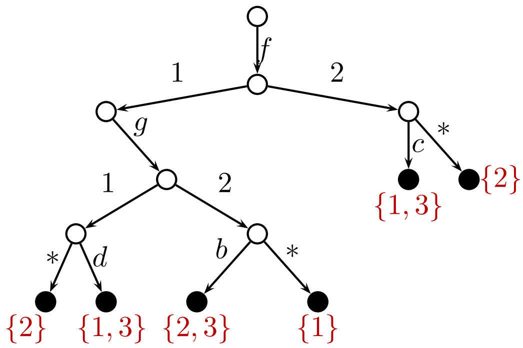
Example: Path index for $\{f(g(d,*),c)\}$



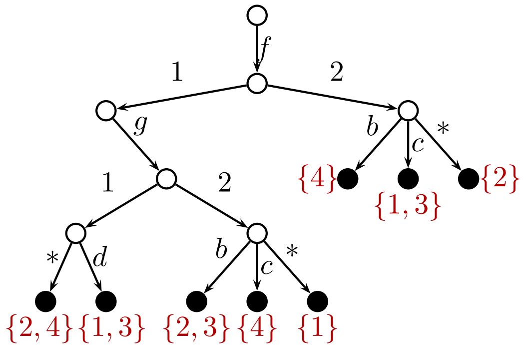
Example: Path index for $\{f(g(d,*),c),\ f(g(*,b),*)\}$



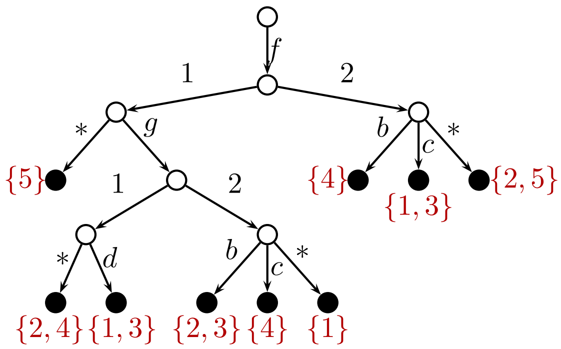
Example: Path index for $\{f(g(d,*),c),\ f(g(*,b),*),\ f(g(d,b),c)\}$



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:

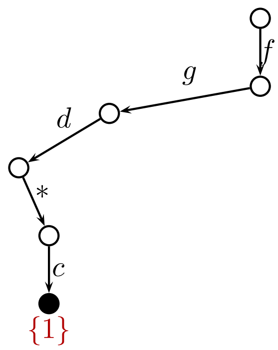
Preorder traversals of terms are encoded in a trie.

A star $*$ represents arbitrary variables.

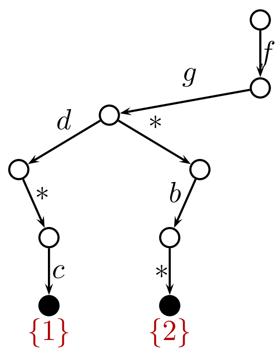
Example: String of $f(g(*,b),*)$: $\quad f.g.*.b.*$

Each leaf of the trie contains (a pointer to) the term that is represented by the path.

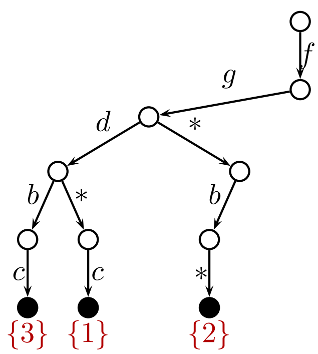
Example: Discrimination tree for $\{f(g(d,*),c)\}$



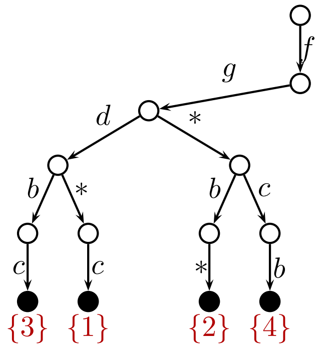
Example: Discrimination tree for $\{f(g(d,*),c),\ f(g(*,b),*)\}$



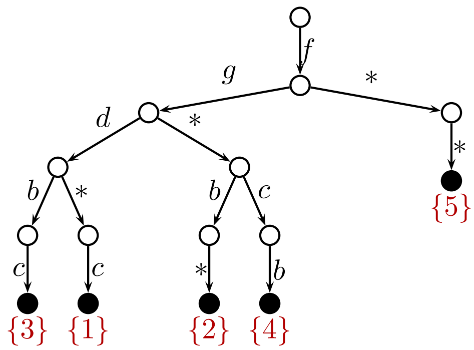
Example: Discrimination tree for $\{f(g(d,*),c),\ f(g(*,b),*),\ f(g(d,b),c)\}$



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.

Feature Vector Indexing

Goal:

C' is subsumed by C if $C' = C\sigma \vee D$.

Find all clauses C' for a given C or vice versa.

If C' is subsumed by C, then

- C' contains at least as many literals as C.
- C' contains at least as many positive literals as C.
- C' contains at least as many negative literals as C.
- C' contains at least as many function symbols as C.
- C' contains at least as many occurrences of f as C.
- C' contains at least as many occurrences of f in negative literals as C.
- the deepest occurrence of f in C' is at least as deep as in C.
- ...

Idea:

Select a list of these "features".

Compute the "feature vector" (a list of natural numbers) for each clause and store it in a trie.

When searching for a subsuming clause: Traverse the trie, check all clauses for which all features are smaller or equal. (Stop if a subsuming clause is found.)

When searching for subsumed clauses: Traverse the trie, check all clauses for which all features are larger or equal.

Advantages:

Works on the clause level, rather than on the term level.

Specialized for subsumption testing.

Disadvantages:

Needs to be complemented by other index structure for other operations.

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 Termination Revisited

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

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

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

TRS is fixed and known in advance.

Literature:

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

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

6.1 Dependency Pairs

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

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

$T_0 := \{\, t \in \mathrm{T}_\Sigma(X) \mid \text{there is an infinite derivation } t \rightarrow_R t_1 \rightarrow_R t_2 \rightarrow_R \ldots \}$.

$T_\infty := \{\, t \in T_0 \mid \forall p > \varepsilon : t/p \notin T_0 \,\} =$ minimal elements of T_0 w. r. t. $\rhd$.

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

R is non-terminating iff $T_0 \neq \emptyset$ iff $T_\infty \neq \emptyset$.

Assume that $T_\infty \neq \emptyset$ and consider some non-terminating derivation starting from $t \in T_\infty$. Since all subterms of t allow only finite derivations, at some point a rule $l \rightarrow r \in R$ must be applied at the root of t (possibly preceded by rewrite steps below the root):

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

In particular, $root(t) = root(l)$, so we see that the root symbol of any term in T_∞ must be contained in $D := \{\, root(l) \mid l \rightarrow r \in R \,\}$. D is called the set of *defined symbols* of R; $C := \Omega \setminus D$ is called the set of *constructor symbols* of R.

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

If v occurred in $r\sigma$ at or below a variable position of r, then $x\sigma/p = v$ for some $x \in var(r) \subseteq var(l)$, hence $s_i \trianglerighteq x\sigma$ and there would be an infinite derivation starting from some t_i. This contradicts $t \in T_\infty$, though.

Therefore, $v = u\sigma$ for some non-variable subterm u of r. As $v \in T_\infty$, we see that $root(u) = root(v) \in D$. Moreover, u cannot be a proper subterm of l, since otherwise again there would be an infinite derivation starting from some t_i.

Putting everything together, we obtain

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

where $r \trianglerighteq u$, $root(u) \in D$, $l \not\trianglerighteq u$, u is not a variable.

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

If we define $S := \{\, l \to u \mid l \to r \in R,\ r \trianglerighteq u,\ root(u) \in D,\ l \not\trianglerighteq u,\ u \notin X \,\}$, we can combine the rewrite step at the root and the subterm step and obtain

$$t \xrightarrow{>\varepsilon}{}^{*}_{R} l\sigma \xrightarrow{\varepsilon}_{S} u\sigma.$$

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

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

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

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

$$DP(R) := \{\, l^\sharp \to u^\sharp \mid l \to r \in R,\ r \trianglerighteq u,\ root(u) \in D,\ l \not\trianglerighteq u,\ u \notin X \,\}.$$

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

$$t^\sharp \to^{*}_{R} l^\sharp\sigma \to_{DP(R)} u^\sharp\sigma$$

where $t^\sharp \in T^\sharp_\infty$ and $u^\sharp\sigma \in T^\sharp_\infty$.

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

Since $u^\sharp\sigma$ is again in $T^\sharp_\infty$, we can continue the process in the same way. We obtain: R is non-terminating iff there is an infinite sequence

$$t_1 \to^{*}_{R} t_2 \to_{DP(R)} t_3 \to^{*}_{R} t_4 \to_{DP(R)} \dots$$

with $t_i \in T^\sharp_\infty$ for all i.

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

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

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

directed graph

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

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

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

The functions *cap* and *ren* are defined by:

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

$$
\begin{aligned}
\mathit{ren}(x) &= y, \quad y \text{ fresh} \\
\mathit{ren}(f(t_1,\dots,t_n)) &= f(\mathit{ren}(t_1),\dots,\mathit{ren}(t_n))
\end{aligned}
$$

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

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

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

$$t_1 \to_R^* t_2 \to_K t_3 \to_R^* t_4 \to_K \dots$$

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

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

6.2 Subterm Criterion

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

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

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

Theorem 6.1 (Hirokawa and Middeldorp) *Let K be a cycle in $DG(R)$. If there is a simple projection π for K such that $\pi(l) \trianglerighteq \pi(r)$ for every $l \to r \in K$ and $\pi(l) \triangleright \pi(r)$ for some $l \to r \in K$, then there are no K-minimal sequences.*

Proof. Suppose that

$$t_1 \to_R^* u_1 \to_K t_2 \to_R^* u_2 \to_K \ldots$$

is a K-minimal infinite rewrite sequence. Apply π to every t_i:

Case 1: $u_i \to_K t_{i+1}$. There is an $l \to r \in K$ such that $u_i = l\sigma$, $t_{i+1} = r\sigma$. Then $\pi(u_i) = \pi(l)\sigma$ and $\pi(t_{i+1}) = \pi(r)\sigma$. By assumption, $\pi(l) \trianglerighteq \pi(r)$. If $\pi(l) = \pi(r)$, then $\pi(u_i) = \pi(t_{i+1})$. If $\pi(l) \triangleright \pi(r)$, then $\pi(u_i) = \pi(l)\sigma \triangleright \pi(r)\sigma = \pi(t_{i+1})$. In particular, $\pi(u_i) \triangleright \pi(t_{i+1})$ for infinitely many i (since every DP is used infinitely often).

Case 2: $t_i \to_R^* u_i$. Then $\pi(t_i) \to \pi(u_i)$.

By applying π to every term in the K-minimal infinite rewrite sequence, we obtain an infinite $(\to_R \cup \triangleright)$-sequence containing infinitely many $\triangleright$-steps. Since $\triangleright$ is well-founded, there must also exist infinitely many $\to_R$-steps (otherwise the infinite sequence would have an infinite tail consisting only of $\triangleright$-steps, contradicting well-foundedness.)

Now note that $\triangleright \circ \to_R \subseteq \to_R \circ \triangleright$. Therefore we can commute $\triangleright$-steps and $\to_R$-steps and move all $\to_R$-steps to the front. We obtain an infinite $\to_R$-sequence that starts with $\pi(t_1)$. However $t_1 \triangleright \pi(t_1)$ and $t_1 \in T_\infty$, so there cannot be an infinite $\to_R$-sequence starting from $\pi(t_1)$. □

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

Better method: Analyze strongly connected components (SCCs).

SCC of a graph: maximal subgraph in which there is a non-empty path from every node to every node. (The two nodes can be identical.)[3]

Important property: Every cycle is contained in some SCC.

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

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

Example: See Ex. 13 from Hirokawa and Middeldorp.

[3]There are several definitions of SCCs that differ in the treatment of edges from a node to itself.

6.3 Reduction Pairs and Argument Filterings

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

$$t_1 \to_R^* u_1 \to_K t_2 \to_R^* u_2 \to_K \dots$$

using well-founded orderings.

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

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

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

This motivates the following definitions:

Rewrite quasi-ordering $\succsim$:

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

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

$\succsim$ is a rewrite quasi-ordering.

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

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

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

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

Argument filtering π:

$\pi : \Omega \cup \Omega^\sharp \to \mathbb{N} \cup \text{list of } \mathbb{N}$

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

Extension to terms:

$\pi(x) = x$

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

$\pi(f(t_1, \dots, t_n)) = f'(\pi(t_{i_1}), \dots, \pi(t_{i_k}))$, if $\pi(f) = [i_1, \dots, i_k]$, where f'/k is a new function symbol.

Let $\succ$ be a reduction ordering, let π be an argument filtering. Define $s \succ_\pi t$ iff $\pi(s) \succ \pi(t)$ and $s \succsim_\pi t$ iff $\pi(s) \succeq \pi(t)$.

Lemma 6.2 *$(\succsim_\pi, \succ_\pi)$ is a reduction pair.*

Proof. Follows from the following two properties:

$\pi(s\sigma) = \pi(s)\sigma_\pi$, where σ_π is the substitution that maps x to $\pi(\sigma(x))$.

$$\pi(s[u]_p) = \begin{cases} \pi(s), & \text{if } p \text{ does not correspond to any position in } \pi(s) \\ \pi(s)[\pi(u)]_q, & \text{if } p \text{ corresponds to } q \text{ in } \pi(s) \end{cases}$$ □

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

Reduction pairs by interpretation:

Let $\mathcal{A}$ be a Σ-algebra; let $\succ$ be a well-founded strict partial ordering on its universe.

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

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

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

For polynomial orderings, this definition permits interpretations of function symbols where some variable does not occur at all (e. g., $P_f(X, Y) = 2X + 1$ for a *binary* function symbol). It is no longer required that *every* variable must occur with some positive coefficient.

Theorem 6.3 (Arts and Giesl) *Let K be a cycle in the dependency graph of the TRS R. If there is a reduction pair $(\succsim, \succ)$ such that*

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

then there is no K-minimal infinite sequence.

Proof. Assume that

$$t_1 \to_R^* u_1 \to_K t_2 \to_R^* u_2 \to_K \dots$$

is a K-minimal infinite rewrite sequence.

As $l \succsim r$ for all $l \to r \in R$, we obtain $t_i \succsim u_i$ by stability under substitutions, compatibility with contexts, reflexivity and transitivity.

As $l \succsim r$ or $l \succ r$ for all $l \to r \in K$, we obtain $u_i \ (\succsim \cup \succ) \ t_{i+1}$ by stability under substitutions.

So we get an infinite $(\succsim \cup \succ)$-sequence containing infinitely many $\succ$-steps (since every DP in K, in particular the one for which $l \succ r$ holds, is used infinitely often).

By compatibility of $\succsim$ and $\succ$, we can transform this into an infinite $\succ$-sequence, contradicting well-foundedness. ◻

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

Search for a reduction pair $(\succsim, \succ)$ such that $l \succsim r$ for all $l \to r \in R$ and $l \succsim r$ or $l \succ r$ for all DPs $l \to r$ in the SCC. Delete all DPs in the SCC for which $l \succ r$. Then re-compute SCCs for the remaining graph and re-start.

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

$$\begin{aligned}
&minus(x, 0) \to x && (1)\\
&minus(s(x), s(y)) \to minus(x, y) && (2)\\
&quot(0, s(y)) \to 0 && (3)\\
&quot(s(x), s(y)) \to s(quot(minus(x, y), s(y))) && (4)
\end{aligned}$$

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

R has three dependency pairs:

$$\begin{aligned}
&minus^\sharp(s(x), s(y)) \to minus^\sharp(x, y) && (5)\\
&quot^\sharp(s(x), s(y)) \to quot^\sharp(minus(x, y), s(y)) && (6)\\
&quot^\sharp(s(x), s(y)) \to minus^\sharp(x, y) && (7)
\end{aligned}$$

The dependency graph of R is

$$\begin{tikzcd}
(5) \arrow[loop below] & (7) \arrow[l] & (6) \arrow[l] \arrow[loop below]
\end{tikzcd}$$

There are exactly two SCCs (and also two cycles). The cycle at (5) can be handled using the subterm criterion with $\pi(minus^\sharp) = 1$. For the cycle at (6) we can use an argument filtering π that maps *minus* to 1 and leaves all other function symbols unchanged (that is, $\pi(g) = [1, \ldots, arity(g)]$ for every g different from *minus*.) After applying the argument filtering, we compare left and right-hand sides using an LPO with precedence $quot > s$ (the precedence of other symbols is irrelevant). We obtain $l \succ r$ for (6) and $l \succsim r$ for (1), (2), (3), (4), so the previous theorem can be applied.

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

A DP processor

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

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

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

Examples:

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

where $SCC_1, \ldots, SCC_n$ are the strongly connected components of G.

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

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

(and analogously for reduction pairs)

The dependency method can also be used for proving termination of *innermost rewriting:* $s \xrightarrow{\text{i}}_R t$ if $s \to_R t$ at position p and no rule of R can be applied at a position strictly below p. (DP processors for innermost termination are more powerful than for ordinary termination, and for program analysis, innermost termination is usually sufficient.)

7 Outlook

Further topics in automated reasoning.

7.1 Satisfiability Modulo Theories (SMT)

DPLL checks satisfiability of propositional formulas.

DPLL can also be used for ground first-order formulas without equality:

Ground first-order atoms are treated like propositional variables.

Truth values of $P(a), Q(a), Q(f(a))$ are independent.

For ground formulas with equality, independence is lost:

If $b \approx c$ is true, then $f(b) \approx f(c)$ must also be true.

Similarly for other theories, e. g. linear arithmetic: $b > 5$ implies $b > 3$.

We can still use DPLL, but we must combine it with a decision procedure for the theory part T:

$M \models_T C$: M and the theory axioms T entail C.

New DPLL rules:

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.

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-Backjump:

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

if $M\ L^{\mathrm{d}}\ M' \models \neg C$
and 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 $M\ L^{\mathrm{d}}\ M'$.

7.2 Sorted Logics

So far, we have considered only unsorted first-order logic.

In practice, one often considers many-sorted logics:

$read/2$ becomes $read : array \times nat \to data.$

$write/3$ becomes $write : array \times nat \times data \to array.$

Variables: $x : data$

Only one declaration per function/predicate/variable symbol.

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

Algebras:

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

Interpretations of function and predicate symbols correspond to their declarations:

$read_{\mathcal{A}} : array_{\mathcal{A}} \times nat_{\mathcal{A}} \to data_{\mathcal{A}}$

Proof theory, calculi, etc.:

Essentially as in the unsorted case.

More difficult:

Subsorts

Overloading

7.3 Splitting

Tableau-like rule within resolution to eliminate variable-disjoint (positive) disjunctions:

$$\frac{N \cup \{C_1 \vee C_2\}}{N \cup \{C_1\} \;\mid\; N \cup \{C_2\}}$$

if $var(C_1) \cap var(C_2) = \emptyset$.

Split clauses are smaller and more likely to be usable for simplification.

Splitting tree is explored using intelligent backtracking.

7.4 Integrating Theories into Resolution

Certain kinds of axioms are

important in practice,

but difficult for theorem provers.

Most important case: equality

but also: orderings, (associativity and) commutativity, ...

Idea: Combine ordered resolution and critical pair computation.

Superposition (ground case):

$$\frac{D' \vee t \approx t' \qquad C' \vee s[t] \approx s'}{D' \vee C' \vee s[t'] \approx s'}$$

Superposition (non-ground case):

$$\frac{D' \vee t \approx t' \qquad C' \vee s[u] \approx s'}{(D' \vee C' \vee s[t'] \approx s')\sigma}$$

where $\sigma = \mathrm{mgu}(t, u)$ and u is not a variable.

Advantages:

No variable overlaps (as in KB-completion).

Stronger ordering restrictions:
Only overlaps of (strictly) maximal sides of (strictly) maximal literals are required.

Stronger redundancy criteria.

Similarly for orderings:

Ordered chaining:

$$\frac{D' \ \vee \ t' < t \qquad C' \ \vee \ s < s'}{(D' \ \vee \ C' \ \vee \ t' < s')\sigma}$$

where σ is a most general unifier of t and s.

Integrating other theories:

Black box:

Use external decision procedure.

Easy, but works only under certain restrictions.

White box:

Integrate using specialized inference rules and theory unification.

Hard work.

Often: integrating more theory axioms is better.

Errata

page 2, beginning of Sect. 1.1:

replace “A is a set” by “A is a non-empty set”

page 8, “Truth Value of a Formula in $\mathcal{A}$”:

replace “Σ-formulas” by “Π-formulas”

page 20, before Lemma 2.9:

replace “decision literals in $M\ L^{\mathrm{d}}\ M'$” by “decision literals in $M'\ L^{\mathrm{d}}\ M''$”

page 24, Sect. 2.7:

replace every $p^d_{i,j}$ by $P^d_{i,j}$

page 34, “Value of a Term in $\mathcal{A}$ with Respect to β”:

replace “$a \in \mathcal{A}$” by “$a \in U_{\mathcal{A}}$”

page 36, Lemma 3.3:

replace “$\beta \circ \sigma : X \to \mathcal{A}$” by “$\beta \circ \sigma : X \to U_{\mathcal{A}}$”

page 42, beginning of Sect 3.6:

replace “A clause set that is” by “Skolem functions that are”

page 42, “Miniscoping”:

add an additional rule

$$Qx\, F \quad \Rightarrow_{MS} \quad F \quad \text{if } x \text{ does not occur freely in } F$$

(this rule is only needed if we start with a formula in which x does not occur at all; otherwise the remaining rules are sufficient)

page 58, “Rule-Based Naive Standard Unification”:

in the fourth rule, replace “$E\{t \mapsto x\}$” by “$E\{x \mapsto t\}$”

page 58/59, Theorem 3.26, Proof:

replace the first item by

- $\Rightarrow_{SU}$ is Noetherian. A suitable lexicographic ordering on the multisets E (with $\bot$ minimal) shows this. Compare in this order:
  (1) the number of variables that occur in E below a function or predicate symbol, or on the right-hand side of an equation, or at least twice;
  (2) the multiset of the sizes (numbers of symbols) of all equations in E;
  (3) the number of non-variable left-hand sides of equations in E.

page 78–82:

everywhere in Sect. 3.15, replace the notation

$$[t_1/x_1, \ldots, t_n/x_n]$$

by

$$\{x_1 \mapsto t_1, \ldots, x_n \mapsto t_n\}$$

page 122:

in Thm. 6.1, Proof, Case 2, replace "$\pi(t_i) \to \pi(u_i)$" by "$\pi(t_i) \to_R^* \pi(u_i)$"