Automated Theorem Proving*

Dr. Uwe Waldmann

with modifications by Prof. Dr. Jasmin Blanchette

Winter Term 2025/26

Contents

*This document contains the text of the lecture slides (almost verbatim) plus some additional information. 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.
Parts of this document are based on lecture notes by Harald Ganzinger and Christoph Weidenbach.

1 Preliminaries

Literature:

Franz Baader and Tobias Nipkow: *Term Rewriting and All That*,
Cambridge Univ. Press, 1998, Chapter 2.

Before we start with the main subjects of the lecture, we repeat some prerequisites from mathematics and computer science and introduce some tools that we will need throughout the lecture.

1.1 Mathematical Prerequisites

$\mathbb{N} = \{0, 1, 2, \ldots\}$ is the set of natural numbers (including 0).

$\mathbb{Z}$, $\mathbb{Q}$, $\mathbb{R}$ denote the integers, rational numbers and the real numbers, respectively.

$\emptyset$ is the empty set.

If M and M' are sets, then $M \cap M'$, $M \cup M'$, and $M \setminus M'$ denote the intersection, union, and set difference of M and M'.

The subset relation is denoted by $\subseteq$. The strict subset relation is denoted by $\subset$ (i.e., $M \subset M'$ if and only if $M \subseteq M'$ and $M \neq M'$).

Relations

Let M be a set, let $n \geq 2$. We write M^n for the n-fold cartesian product $M \times \cdots \times M$.

To handle the cases $n \geq 2$, $n = 1$, and $n = 0$ simultaneously, we also define $M^1 = M$ and $M^0 = \{()\}$. (We do not distinguish between an element m of M and a 1-tuple (m) of an element of M.)

An n-ary *relation* R over some set M is a subset of M^n: $R \subseteq M^n$.

We often use predicate notation for relations:

Instead of $(m_1, \ldots, m_n) \in R$ we write $R(m_1, \ldots, m_n)$, and say that $R(m_1, \ldots, m_n)$ holds or is true.

For binary relations, we often use infix notation, so
$(m, m') \in < \;\Leftrightarrow\; <(m, m') \;\Leftrightarrow\; m < m'$.

Since relations are sets, we can use the usual set operations for them.

Example: Let $R = \{(0, 2), (1, 2), (2, 2), (3, 2)\} \subseteq \mathbb{N} \times \mathbb{N}$.
Then $R \cap < \;=\; R \cap \{(n, m) \in \mathbb{N} \times \mathbb{N} \mid n < m\} \;=\; \{(0, 2), (1, 2)\}$.

A relation Q is a *subrelation* of a relation R if $Q \subseteq R$.

Words

Given a nonempty set (also called *alphabet*) Σ, the set Σ^* of *finite words* over Σ is defined inductively by

(i) the empty word ε is in Σ^*,

(ii) if $u \in \Sigma^*$ and $a \in \Sigma$ then ua is in Σ^*.

The set of *nonempty finite words* Σ^+ is $\Sigma^* \setminus \{\varepsilon\}$.

The *concatenation* of two words $u, v \in \Sigma^*$ is denoted by uv.

The length $|u|$ of a word $u \in \Sigma^*$ is defined by

(i) $|\varepsilon| := 0$,

(ii) $|ua| := |u| + 1$ for any $u \in \Sigma^*$ and $a \in \Sigma$.

1.2 Abstract Reduction Systems

Throughout the lecture, we will have to work with reduction systems.

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

A is a nonempty set,

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

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

Let $\to' \subseteq A \times A$ and $\to'' \subseteq A \times A$ be two binary relations. Then the *composition of* $\to'$ *and* $\to''$ is the binary relation $(\to' \circ \to'') \subseteq A \times A$ defined by

$a \ (\to' \circ \to'') \ c$ if and only if there exists some $b \in A$ such that $a \to' b$ and $b \to'' c$.

For a binary relation $\to \subseteq A \times A$, we define:

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

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

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

c is a *normal form of* b if $b \to^* c$ and c is in normal form.
Notation: $b{\downarrow}$ denotes the normal form of b if it is unique.

A relation $\to$ is called

terminating if there is no infinite descending chain $b_0 \to b_1 \to b_2 \to \cdots$.

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

Lemma 1.2.1 *If $\to$ is terminating, then it is normalizing.*

Note: The reverse implication does not hold (see exercise).

1.3 Orderings

Important properties of binary relations:

Let $M \neq \emptyset$. A binary relation $R \subseteq M \times M$ is called

reflexive if $R(x,x)$ for all $x \in M$,

irreflexive if $\neg R(x,x)$ for all $x \in M$,

antisymmetric if $R(x,y)$ and $R(y,x)$ imply $x = y$ for all $x, y \in M$,

transitive if $R(x,y)$ and $R(y,z)$ imply $R(x,z)$ for all $x, y, z \in M$,

total if $R(x,y)$ or $R(y,x)$ or $x = y$ for all $x, y \in M$.

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

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

Let $\succ$ be a strict partial ordering on M; let $M' \subseteq M$.

$a \in M'$ is called *minimal in* M' if there is no $b \in M'$ with $a \succ b$.

$a \in M'$ is called *smallest in* M' if $b \succ a$ for all $b \in M' \setminus \{a\}$.

Analogously:

$a \in M'$ is called *maximal in* M' if there is no $b \in M'$ with $a \prec b$.

$a \in M'$ is called *largest in* M' if $b \prec a$ for all $b \in M' \setminus \{a\}$.

Notation:
$M^{\prec x} = \{y \in M \mid y \prec x\}$,
$M^{\preceq x} = \{y \in M \mid y \preceq x\}$.

A subset $M' \subseteq M$ is called *downward-closed* if $x \in M'$ and $x \succ y$ implies $y \in M'$.

Well-Foundedness

Termination of reduction systems is strongly related to the concept of well-founded orderings.

A strict partial ordering $\succ$ on M is called *well-founded (or Noetherian)* if there is no infinite descending chain $a_0 \succ a_1 \succ a_2 \succ \cdots$ with $a_i \in M$ for every $i \in \mathbb{N}$.

Well-Foundedness and Termination

Lemma 1.3.1 *If $\succ$ is a well-founded partial ordering and $\to \subseteq \succ$, then $\to$ is terminating.*

Proof. Suppose that $\to \subseteq \succ$ for some partial ordering $\succ$ and that $\to$ is not terminating. Then there exists an infinite descending chain $b_0 \to b_1 \to b_2 \to \cdots$. Since $\to \subseteq \succ$, we have an infinite descending chain $b_0 \succ b_1 \succ b_2 \succ \cdots$, hence $\succ$ is not well-founded. □

Lemma 1.3.2 *If $\to$ is a terminating binary relation over A, then $\to^+$ is a well-founded partial ordering.*

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

Well-Founded Orderings: Examples

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

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

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

on $M_1 \times M_2$ by

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

(analogously for more than two orderings). This again yields a well-founded ordering.

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

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

is a well-founded ordering on the set Σ^* of finite words over the alphabet Σ.

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

Basic Properties of Well-Founded Orderings

Lemma 1.3.3 *$(M, \succ)$ is well-founded if and only if every nonempty $M' \subseteq M$ has a minimal element.*

Proof. "$\Leftarrow$": Suppose that $(M, \succ)$ is not well-founded. Then there is an infinite descending chain $a_0 \succ a_1 \succ a_2 \succ \cdots$ with $a_i \in M$. Consequently, the subset $M' = \{a_i \mid i \in \mathbb{N}\}$, does not have a minimal element.

"$\Rightarrow$": Suppose that the nonempty subset $M' \subseteq M$ does not have a minimal element. Choose $a_0 \in M'$ arbitrarily. Since for every $a_i \in M'$ there is a smaller $a_{i+1} \in M'$ (otherwise a_i would be minimal in M'), there is an infinite descending chain $a_0 \succ a_1 \succ a_2 \succ \cdots$ □

Lemma 1.3.4 *$(M_1, \succ_1)$ and $(M_2, \succ_2)$ are well-founded if and only if $(M_1 \times M_2, \succ)$ with $\succ = (\succ_1, \succ_2)_{\text{lex}}$ is well-founded.*

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

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 $(M_2, \succ_2)$.

"⇐": obvious. □

Monotone Mappings

Let $(M, \succ)$ and $(M', \succ')$ be strict partial orderings. A mapping $\varphi : M \to M'$ is called *monotone* if $a \succ b$ implies $\varphi(a) \succ' \varphi(b)$ for all $a, b \in M$.

Lemma 1.3.5 *If φ is a monotone mapping from $(M, \succ)$ to $(M', \succ')$ and $(M', \succ')$ is well-founded, then $(M, \succ)$ is well-founded.*

Proof. Suppose that $(M, \succ)$ is not well-founded, then there exists an infinite descending chain $a_0 \succ a_1 \succ a_2 \succ \cdots$. Since $a_i \succ a_{i+1}$ implies $\varphi(a_i) \succ' \varphi(a_{i+1})$, we obtain an infinite descending chain $\varphi(a_0) \succ' \varphi(a_1) \succ' \varphi(a_2) \succ' \ldots$, contradicting the well-foundedness of $(M', \succ')$. □

Well-Founded Induction

Well-founded induction generalizes the usual induction over natural numbers or data structures.

Theorem 1.3.6 (Well-Founded (or 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 $m \succ m'$,*
> *then $Q(m)$.*

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 that $X \neq \emptyset$. Since $(M, \succ)$ is well-founded, X has a minimal element m_0. Hence for all $m' \in M$ with $m' \prec m_0$ the property $Q(m')$ holds. On the other hand, the implication which is presupposed for this theorem holds in particular also for m_0, hence $Q(m_0)$ must be true. Therefore m_0 cannot be in X, contradicting the assumption. □

Well-Founded Recursion

Similarly, well-founded recursion generalizes the usual recursion over natural numbers or data structures. We will need this concept only once in this lecture, but in one of the main theorems.

Let M and S be sets, let $N \subseteq M$, and let $f : M \to S$ be a function. Then the *restriction* of f to N, denoted by $f|_N$, is a function from N to S with $f|_N(x) = f(x)$ for all $x \in N$.

Theorem 1.3.7 (Well-Founded (or Noetherian) Recursion) *Let $(M, \succ)$ be a well-founded ordering, let S be a set. Let ϕ be a binary function that takes two arguments x and g and maps them to an element of S, where $x \in M$ and g is a function from $M^{\prec x}$ to S.*

Then there exists exactly one function $f : M \to S$ such that for all $x \in M$

$$f(x) = \phi(x, f|_{M^{\prec x}})$$

Proof. The proof consists of four parts.

Part 1: For every downward-closed subset $N \subseteq M$ there is *at most one* function $f : N \to S$ such that $f(x) = \phi(x, f|_{N^{\prec x}}) = \phi(x, f|_{M^{\prec x}})$.

Proof: First observe that if $N \subseteq M$ is downward-closed and $x \in N$, then $N^{\prec x} = M^{\prec x}$. Assume that there exist a downward-closed subset $N \subseteq M$ and two *different* functions f_1 and f_2 from N to S with the property. Therefore, the set $N' := \{x \in N \mid f_1(x) \neq f_2(x)\}$ is nonempty. By well-foundedness, N' has a minimal element y. By minimality of y, $f_1|_{M^{\prec y}} = f_2|_{M^{\prec y}}$. Therefore $f_1(y) = \phi(y, f_1|_{M^{\prec y}}) = \phi(y, f_2|_{M^{\prec y}}) = f_2(y)$, contradicting the assumption.

Part 2: If N_1 and N_2 are downward-closed subsets of M and the functions $f_1 : N_1 \to S$ and $f_2 : N_2 \to S$ satisfy $f_i(x) = \phi(x, f_i|_{M^{\prec x}})$ for all $x \in N_i$ $(i = 1, 2)$, then $f_1(x) = f_2(x)$ for all $x \in N_1 \cap N_2$.

Proof: Define $N_0 := N_1 \cap N_2$ and $f_i' = f_i|_{N_0}$ for $i = 1, 2$. Clearly N_0 is downward-closed and for all $x \in N_0$ and $i = 1, 2$ we have $f_i'(x) = f_i(x) = \phi(x, f_i|_{M^{\prec x}}) = \phi(x, f_i'|_{M^{\prec x}})$. By part 1, there is at most one function from N_0 to S with this property, so $f_1' = f_2'$, and therefore $f_1(x) = f_2(x)$ for all $x \in N_1 \cap N_2$.

Part 3: For every $y \in M$ there exists a function $f_y : M^{\preceq y} \to S$ such that $f_y(x) = \phi(x, f_y|_{M^{\prec x}})$ for all $x \in M^{\preceq y}$.

Proof: We use well-founded induction over $\succ$. Let $y \in M$. By the induction hypothesis, for every $z \prec y$ there exists a function $f_z : M^{\preceq z} \to S$ such that $f_z(x) = \phi(x, f_z|_{M^{\prec x}})$ for all $x \in M^{\preceq z}$. By part 2, all functions f_z agree on the intersections of their domains. Define the function $f_y : M^{\preceq y} \to S$ by $f_y(x) = f_x(x)$ for $x \prec y$ and by $f_y(y) = \phi(y, f_y|_{M^{\prec y}})$. The

function f_y has the desired property for $x = y$ by construction and for all $x \prec y$ by the induction hypothesis (since $f_y(x) = f_x(x)$ for $x \prec y$ and f_x has the desired property).

Part 4: There exists a function $f : M \to S$ such that $f(x) = \phi(x, f|_{M^{\prec x}})$ for all $x \in M$.

Proof: Define $f : M \to S$ by $f(x) = f_x(x)$.

The claim of the theorem follows now from part 1 (for $N := M$) and part 4. □

The well-founded recursion scheme generalizes terminating recursive programs.

Note that functions defined by well-founded recursion need *not* be computable, in particular since for many well-founded orderings the sets $M^{\prec x}$ may be infinite.

1.4 Multisets

Let M be a set. A *multiset* S over M is a mapping $S : M \to \mathbb{N}$. We interpret $S(m)$ as 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 ($\in$, $\subseteq$, $\cup$, $\cap$, etc.) with analogous meaning also for multisets, e.g.,

$$
\begin{aligned}
m \in S \quad &:\Leftrightarrow \quad S(m) > 0 \\
(S_1 \cup S_2)(m) \quad &:= \quad S_1(m) + S_2(m) \\
(S_1 \cap S_2)(m) \quad &:= \quad \min\{S_1(m), S_2(m)\} \\
(S_1 - S_2)(m) \quad &:= \quad \begin{cases} S_1(m) - S_2(m) & \text{if } S_1(m) \geq S_2(m) \\ 0 & \text{otherwise} \end{cases} \\
S_1 \subseteq S_2 \quad &:\Leftrightarrow \quad S_1(m) \leq S_2(m) \text{ for all } m \in M
\end{aligned}
$$

A multiset S is called *finite* if the set $\{m \in M \mid S(m) > 0\}$ is finite.

From now on we only consider finite multisets.

Multiset Orderings

Let $(M, \succ)$ be an abstract reduction system. The *multiset extension* of $\succ$ to multisets over M is defined by

$S_1 \succ_{\text{mul}} S_2$ if and only if
there exist multisets X and Y over M such that
$\emptyset \neq X \subseteq S_1$,
$S_2 = (S_1 - X) \cup Y$,
$\forall y \in Y\ \exists x \in X\colon x \succ y$

Theorem 1.4.1
(a) If $\succ$ is transitive, then $\succ_{\text{mul}}$ is transitive.

(b) If $\succ$ is irreflexive and transitive, then $\succ_{\text{mul}}$ is irreflexive.

(c) If $\succ$ is a well-founded ordering, then $\succ_{\text{mul}}$ is a well-founded ordering.

(d) If $\succ$ is a strict total ordering, then $\succ_{\text{mul}}$ is a strict total ordering.

Proof. See Baader and Nipkow, pages 22–24. □

The multiset extension as defined above is due to Dershowitz and Manna (1979).

There are several other ways to characterize the multiset extension of a binary relation. The following one is due to Huet and Oppen (1980):

Let $(M, \succ)$ be an abstract reduction system. The *(Huet–Oppen) multiset extension* of $\succ$ to multisets over M is defined by

$S_1 \succ_{\text{mul}}^{\text{HO}} S_2$ if and only if
$S_1 \neq S_2$ and
$\forall m \in M\colon \big(S_2(m) > S_1(m)$
$\Rightarrow\ \exists m' \in M\colon m' \succ m$ and $S_1(m') > S_2(m')\big)$

A third way to characterize the multiset extension of a binary relation $\succ$ is to define it as the transitive closure of the relation $\succ_{\text{mul}}^{1}$ given by

$S_1 \succ_{\text{mul}}^{1} S_2$ if and only if
there exists $x \in S_1$ and a multiset Y over M such that
$S_2 = (S_1 - \{x\}) \cup Y$,
$\forall y \in Y\colon x \succ y$

For strict partial orderings all three characterizations of $\succ_{\text{mul}}$ are equivalent:

Theorem 1.4.2 *If $\succ$ is a strict partial ordering, then*
(a) $\succ_{\text{mul}} = \succ^{\text{HO}}_{\text{mul}}$,
(b) $\succ_{\text{mul}} = (\succ^{1}_{\text{mul}})^{+}$.

Proof. (a) See Baader and Nipkow, pages 24–26. (b) Exercise. □

Note, however, that for an arbitrary binary relation $\succ$ all three relations $\succ_{\text{mul}}$, $\succ^{\text{HO}}_{\text{mul}}$, and $(\succ^{1}_{\text{mul}})^{+}$ may be different.

2 Propositional Logic

Propositional logic

- logic of truth values,
- decidable (but NP-complete),
- can be used to describe functions over a finite domain,
- industry standard for many analysis/verification tasks (e.g., model checking).

2.1 Syntax

When we define a logic, we must define what formulas of the logic look like (syntax), and what they mean (semantics). We start with the syntax.

Propositional formulas are built from

- propositional variables,
- logical connectives (e.g., $\wedge$, $\vee$).

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 inductively as follows:

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

Sometimes further connectives are used, for instance

$$
\begin{array}{ll}
(F \leftarrow G) & \text{(reverse implication)} \\
(F \oplus G) & \text{(exclusive or)} \\
(\text{if } F \text{ then } G_1 \text{ else } G_0) & \text{(if-then-else)}
\end{array}
$$

Notational Conventions

As a notational convention we assume that $\neg$ binds strongest, and we remove outermost parentheses, so $\neg P \vee Q$ is actually a shorthand for $((\neg P) \vee Q)$.

Instead of $((P \wedge Q) \wedge R)$ we simply write $P \wedge Q \wedge R$ (analogously for $\vee$).

For all other logical connectives, we will use parentheses when needed.

Formula Manipulation

Automated theorem proving is very much formula manipulation. We perform syntactic operations on formulas to show semantic properties of formulas.

To precisely describe the manipulation of a formula, we introduce positions.

A *position* is a word over $\mathbb{N}$. The set of positions of a formula F is inductively defined by

$$
\begin{aligned}
\mathrm{pos}(F) &:= \{\varepsilon\} \text{ if } F \in \{\top, \bot\} \text{ or } F \in \Pi \\
\mathrm{pos}(\neg F) &:= \{\varepsilon\} \cup \{1p \mid p \in \mathrm{pos}(F)\} \\
\mathrm{pos}(F \circ G) &:= \{\varepsilon\} \cup \{1p \mid p \in \mathrm{pos}(F)\} \cup \{2p \mid p \in \mathrm{pos}(G)\} \\
&\quad \text{where } \circ \in \{\wedge, \vee, \rightarrow, \leftrightarrow\}.
\end{aligned}
$$

The prefix order $\leq$ on positions is defined by $p \leq q$ if there is some p' such that $pp' = q$.

Note that the prefix order is partial, e.g., the positions 12 and 21 are not comparable, they are "parallel," see below.

By $<$ we denote the strict part of $\leq$, that is, $p < q$ if $p \leq q$ but not $q \leq p$.

By $\parallel$ we denote incomparable positions, that is, $p \parallel q$ if neither $p \leq q$ nor $q \leq p$.

We say that p is *above* q if $p \leq q$, p is *strictly above* q if $p < q$, and p and q are *parallel* if $p \parallel q$.

The *size* of a formula F is given by the cardinality of $\mathrm{pos}(F)$: $|F| := |\mathrm{pos}(F)|$.

The *subformula* of F at position $p \in \mathrm{pos}(F)$ is recursively defined by

$$\begin{aligned} F|_\varepsilon &:= F \\ (\neg F)|_{1p} &:= F|_p \\ (F_1 \circ F_2)|_{ip} &:= F_i|_p \quad \text{where } i \in \{1,2\} \\ &\qquad\qquad \text{and } \circ \in \{\wedge, \vee, \to, \leftrightarrow\}. \end{aligned}$$

Finally, the *replacement* of a subformula at position $p \in \mathrm{pos}(F)$ by a formula G is recursively defined by

$$\begin{aligned} F[G]_\varepsilon &:= G \\ (\neg F)[G]_{1p} &:= \neg(F[G]_p) \\ (F_1 \circ F_2)[G]_{1p} &:= (F_1[G]_p \circ F_2) \\ (F_1 \circ F_2)[G]_{2p} &:= (F_1 \circ F_2[G]_p) \\ &\qquad\qquad \text{where } \circ \in \{\wedge, \vee, \to, \leftrightarrow\}. \end{aligned}$$

Example 2.1.1 *The set of positions for the formula* $F = (P \to Q) \to (P \wedge \neg R)$ *is* $\mathrm{pos}(F) = \{\varepsilon, 1, 11, 12, 2, 21, 22, 221\}$.

The subformula at position 22 *is* $F|_{22} = \neg R$ *and replacing this formula by* $P \leftrightarrow Q$ *results in* $F[P \leftrightarrow Q]_{22} = (P \to Q) \to (P \wedge (P \leftrightarrow Q))$.

2.2 Semantics

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

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

Valuations

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

A Π-*valuation* is a function $\mathcal{A} : \Pi \to \{0, 1\}$ where $\{0, 1\}$ is the set of *truth values*.

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

Given a Π-valuation $\mathcal{A} : \Pi \to \{0, 1\}$, its extension to formulas $\mathcal{A}^* : \mathrm{F}_\Pi \to \{0, 1\}$ is defined inductively as follows:

$$\begin{aligned}
\mathcal{A}^*(\bot) &= 0 \\
\mathcal{A}^*(\top) &= 1 \\
\mathcal{A}^*(P) &= \mathcal{A}(P) \\
\mathcal{A}^*(\neg F) &= 1 - \mathcal{A}^*(F) \\
\mathcal{A}^*(F \wedge G) &= \min(\mathcal{A}^*(F), \mathcal{A}^*(G)) \\
\mathcal{A}^*(F \vee G) &= \max(\mathcal{A}^*(F), \mathcal{A}^*(G)) \\
\mathcal{A}^*(F \to G) &= \max(1 - \mathcal{A}^*(F), \mathcal{A}^*(G)) \\
\mathcal{A}^*(F \leftrightarrow G) &= \text{if } \mathcal{A}^*(F) = \mathcal{A}^*(G) \text{ then 1 else 0}
\end{aligned}$$

For simplicity, the extension $\mathcal{A}^*$ of $\mathcal{A}$ is usually also denoted by $\mathcal{A}$.

Note that formulas and truth values are disjoint classes of objects. Statements like $P = 1$ or $F \wedge G = 0$ that equate formulas and truth values are nonsensical. A formula is never *equal to* a truth value, but it *has* a truth value in some valuation $\mathcal{A}$.

2.3 Models, Validity, and Satisfiability

Let F be a Π-formula.

We say that F is *true* in $\mathcal{A}$ ($\mathcal{A}$ is a *model* of F; F *is valid* in $\mathcal{A}$; F *holds* in $\mathcal{A}$), written $\mathcal{A} \models F$, if $\mathcal{A}(F) = 1$.

We say that F is *valid* or that F is a *tautology*, written $\models F$, if $\mathcal{A} \models F$ for all Π-valuations $\mathcal{A}$.

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

Entailment and Equivalence

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

$$\text{if } \mathcal{A} \models F \text{ then } \mathcal{A} \models G,$$

or equivalently

$$\mathcal{A}(F) \leq \mathcal{A}(G).$$

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

$$\mathcal{A} \models F \text{ if and only if } \mathcal{A} \models G,$$

or equivalently

$$\mathcal{A}(F) = \mathcal{A}(G).$$

F and G are called *equisatisfiable* if either both F and G are satisfiable, or both F and G are unsatisfiable.

The notions defined above for formulas, such as satisfiability, validity, and entailment, are extended to sets of formulas N by treating sets of formulas analogously to conjunctions of formulas, e.g.:

$\mathcal{A} \models N$ if $\mathcal{A} \models G$ for all $G \in N$.

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

Proposition 2.3.1 *$F \models G$ if and only if $\models (F \to G)$.*

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

($\Leftarrow$) 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, $\mathcal{A}(F \to G) = \max(1 - \mathcal{A}(F), \mathcal{A}(G)) = \max(1 - 1, 0) = 0$, so $(F \to G)$ does not hold in $\mathcal{A}$. □

Proposition 2.3.2 *$F \models\!\models G$ if and only if $\models (F \leftrightarrow G)$.*

Proof. Analogously to Prop. 2.3.1. □

Validity vs. Unsatisfiability

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

Proposition 2.3.3 *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 $\mathcal{A}(\neg F) = 1 - \mathcal{A}(F) = 0$ for every valuation $\mathcal{A}$, so $\neg F$ is unsatisfiable.

($\Leftarrow$) Analogously. □

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

In a similar way, entailment can be reduced to unsatisfiability and vice versa:

Proposition 2.3.4 *$G \models F$ if and only if $G \wedge \neg F$ is unsatisfiable.*
$N \models F$ if and only if $N \cup \{\neg F\}$ is unsatisfiable.

Proposition 2.3.5 *$G \models \bot$ if and only if G is unsatisfiable.*
$N \models \bot$ if and only if N 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 in $\mathcal{A}$.

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 decidable (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.

Replacement Theorem

Proposition 2.3.6 *Let $\mathcal{A}$ be a valuation, let F and G be formulas, and let $H = H[F]_p$ be a formula in which F occurs as a subformula at position p.*

If $\mathcal{A}(F) = \mathcal{A}(G)$, then $\mathcal{A}(H[F]_p) = \mathcal{A}(H[G]_p)$.

Proof. The proof proceeds by induction over the length of p.

If $p = \varepsilon$, then $H[F]_\varepsilon = F$ and $H[G]_\varepsilon = G$, so $\mathcal{A}(H[F]_p) = \mathcal{A}(F) = \mathcal{A}(G) = \mathcal{A}(H[G]_p)$ by assumption.

If $p = 1q$ or $p = 2q$, then $H = \neg H_1$ or $H = H_1 \circ H_2$ for $\circ \in \{\wedge, \vee, \rightarrow, \leftrightarrow\}$. Assume that $p = 1q$ and that $H = H_1 \wedge H_2$, hence $H[F]_p = H[F]_{1q} = H_1[F]_q \wedge H_2$. By the induction hypothesis, $\mathcal{A}(H_1[F]_q) = \mathcal{A}(H_1[G]_q)$. Hence $\mathcal{A}(H[F]_{1q}) = \mathcal{A}(H_1[F]_q \wedge H_2) = \min(\mathcal{A}(H_1[F]_q), \mathcal{A}(H_2)) = \min(\mathcal{A}(H_1[G]_q), \mathcal{A}(H_2)) = \mathcal{A}(H_1[G]_q \wedge H_2) = \mathcal{A}(H[G]_{1q})$.

The case $p = 2q$ and the other boolean connectives are handled analogously. □

Theorem 2.3.7 *Let F and G be equivalent formulas, let $H = H[F]_p$ be a formula in which F occurs as a subformula at position p.*

Then $H[F]_p$ is equivalent to $H[G]_p$.

Proof. We have to show that $\mathcal{A}(H[F]_p) = \mathcal{A}(H[G]_p)$ for every Π-valuation $\mathcal{A}$.

Choose $\mathcal{A}$ arbitrarily. Since F and G are equivalent, we know that $\mathcal{A}(F) = \mathcal{A}(G)$. Hence, by the previous proposition, $\mathcal{A}(H[F]_p) = \mathcal{A}(H[G]_p)$. □

Some Important Equivalences

Proposition 2.3.8 *The following equivalences hold for all formulas F, G, H:*

$$\begin{array}{rcll} (F \wedge F) & \models\!\dashv & F & \\ (F \vee F) & \models\!\dashv & F & \textit{(Idempotence)} \\ (F \wedge G) & \models\!\dashv & (G \wedge F) & \\ (F \vee G) & \models\!\dashv & (G \vee F) & \textit{(Commutativity)} \\ (F \wedge (G \wedge H)) & \models\!\dashv & ((F \wedge G) \wedge H) & \\ (F \vee (G \vee H)) & \models\!\dashv & ((F \vee G) \vee H) & \textit{(Associativity)} \\ (F \wedge (G \vee H)) & \models\!\dashv & ((F \wedge G) \vee (F \wedge H)) & \\ (F \vee (G \wedge H)) & \models\!\dashv & ((F \vee G) \wedge (F \vee H)) & \textit{(Distributivity)} \end{array}$$

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

An Important Entailment

Proposition 2.3.9 *The following entailment holds for all formulas F, G, H:*

$$(F \vee H) \wedge (G \vee \neg H) \models F \vee G \qquad \textit{(Generalized Resolution)}$$

2.4 Normal Forms

Many theorem proving calculi do not operate on arbitrary formulas, but only on some restricted class of formulas.

We define *conjunctions* of formulas as follows:

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

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

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

And analogously *disjunctions:*

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

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

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

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, also clausal 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 (e.g., P and $\neg P$) 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 coNP-complete.

Conversion to CNF/DNF

Proposition 2.4.1 *For every formula there is an equivalent formula in CNF (and also an equivalent formula in DNF).*

Proof. We describe a (naive) algorithm to convert a formula to CNF.

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

Step 1: Eliminate equivalences:

$$H[F \leftrightarrow G]_p \;\Rightarrow_{\mathrm{CNF}}\; H[(F \to G) \wedge (G \to F)]_p$$

Step 2: Eliminate implications:

$$H[F \to G]_p \;\Rightarrow_{\mathrm{CNF}}\; H[\neg F \vee G]_p$$

Step 3: Push negations inward:

$$H[\neg(F \vee G)]_p \;\Rightarrow_{\text{CNF}}\; H[\neg F \wedge \neg G]_p$$
$$H[\neg(F \wedge G)]_p \;\Rightarrow_{\text{CNF}}\; H[\neg F \vee \neg G]_p$$

Step 4: Eliminate multiple negations:

$$H[\neg\neg F]_p \;\Rightarrow_{\text{CNF}}\; H[F]_p$$

Step 5: Push disjunctions inward:

$$H[(F \wedge F') \vee G]_p \;\Rightarrow_{\text{CNF}}\; H[(F \vee G) \wedge (F' \vee G)]_p$$

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

$$H[F \wedge \top]_p \;\Rightarrow_{\text{CNF}}\; H[F]_p$$
$$H[F \wedge \bot]_p \;\Rightarrow_{\text{CNF}}\; H[\bot]_p$$
$$H[F \vee \top]_p \;\Rightarrow_{\text{CNF}}\; H[\top]_p$$
$$H[F \vee \bot]_p \;\Rightarrow_{\text{CNF}}\; H[F]_p$$
$$H[\neg\bot]_p \;\Rightarrow_{\text{CNF}}\; H[\top]_p$$
$$H[\neg\top]_p \;\Rightarrow_{\text{CNF}}\; H[\bot]_p$$

Proving termination is easy for steps 2, 4, and 6; steps 1, 3, and 5 are slightly more complicated.

For step 1, we can prove termination in the following way: We define a function μ_1 from formulas to positive integers such that $\mu_1(\bot) = \mu_1(\top) = \mu_1(P) = 1$, $\mu_1(\neg F) = \mu_1(F)$, $\mu_1(F \wedge G) = \mu_1(F \vee G) = \mu_1(F \to G) = \mu_1(F) + \mu_1(G)$, and $\mu_1(F \leftrightarrow G) = 2\mu_1(F) + 2\mu_1(G) + 1$. Observe that μ_1 is constructed in such a way that $\mu_1(F) > \mu_1(G)$ implies $\mu_1(H[F]) > \mu_1(H[G])$ for all formulas F, G, and H. Furthermore, μ_1 has the property that swapping the arguments of some $\wedge$ or $\vee$ in a formula F does not change the value of $\mu_1(F)$. (This is important since the transformation rules can be applied modulo commutativity of $\wedge$ and $\vee$.). Using these properties, we can show that whenever a formula H' is the result of applying the rule of step 1 to a formula H, then $\mu_1(H) > \mu_1(H')$. Since μ_1 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_2 from formulas to positive integers such that $\mu_2(\bot) = \mu_2(\top) = \mu_2(P) = 1$, $\mu_2(\neg F) = 2\mu_2(F)$, $\mu_2(F \wedge G) = \mu_2(F \vee G) = \mu_2(F \to G) = \mu_2(F \leftrightarrow G) = \mu_2(F) + \mu_2(G) + 1$. Whenever a formula H' is the result of applying a rule of step 3 to a formula H, then $\mu_2(H) > \mu_2(H')$. Since μ_2 takes only positive integer values, step 3 must terminate.

For step 5, we use a function μ_3 from formulas to positive integers such that $\mu_3(\bot) = \mu_3(\top) = \mu_3(P) = 1$, $\mu_3(\neg F) = \mu_3(F) + 1$, $\mu_3(F \wedge G) = \mu_3(F \to G) = \mu_3(F \leftrightarrow G) = \mu_3(F) + \mu_3(G) + 1$, and $\mu_3(F \vee G) = 2\mu_3(F)\mu_3(G)$. Again, if a formula H' is the result

of applying a rule of step 5 to a formula H, then $\mu_3(H) > \mu_3(H')$. Since μ_3 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 it is conjunctions that must be pushed inward in step 5. □

Example 2.4.2 *The following steps convert the formula* $(P \vee Q) \leftrightarrow R$ *to CNF:*

$$\begin{aligned}
& (P \vee Q) \leftrightarrow R \\
\Rightarrow_{CNF(1)} \; & ((P \vee Q) \rightarrow R) \wedge (R \rightarrow (P \vee Q)) \\
\Rightarrow^{2}_{CNF(2)} \; & (\neg(P \vee Q) \vee R) \wedge (\neg R \vee (P \vee Q)) \\
\Rightarrow_{CNF(3)} \; & ((\neg P \wedge \neg Q) \vee R) \wedge (\neg R \vee (P \vee Q)) \\
\Rightarrow_{CNF(5)} \; & ((\neg P \vee R) \wedge (\neg Q \vee R)) \wedge (\neg R \vee (P \vee Q))
\end{aligned}$$

Exploiting the associativity of $\wedge$ *and* $\vee$*, we obtain*

$$(\neg P \vee R) \wedge (\neg Q \vee R) \wedge (\neg R \vee P \vee Q)$$

Negation Normal Form (NNF)

The formula after application of step 4 is said to be in *Negation Normal Form*, i.e., it contains neither $\rightarrow$ nor $\leftrightarrow$ and negation symbols occur only in front of propositional variables (atoms).

Complexity

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

Example:

$$\begin{aligned}
& (P \leftrightarrow Q) \leftrightarrow (R \leftrightarrow S) \\
\Rightarrow^{+}_{CNF} \; & (\neg P \vee \neg Q \vee \neg R \vee S) \wedge (\neg P \vee \neg Q \vee R \vee \neg S) \wedge \\
& (\neg P \vee Q \vee \neg R \vee \neg S) \wedge (\neg P \vee Q \vee R \vee S) \wedge \\
& (P \vee \neg Q \vee \neg R \vee \neg S) \wedge (P \vee \neg Q \vee R \vee S) \wedge \\
& (P \vee Q \vee \neg R \vee S) \wedge (P \vee Q \vee R \vee \neg S)
\end{aligned}$$

2.5 Improving the CNF Transformation

The goal

"Given a formula F, find an *equivalent* formula G in CNF"

is unpractical.

But if we relax the requirement to

"Given a formula F, find an *equisatisfiable* formula G in CNF"

we can get an efficient transformation.

Literature:

Andreas Nonnengart and Christoph Weidenbach: Computing small clause normal forms, in *Handbook of Automated Reasoning*, pages 335-367. Elsevier, 2001.

Christoph Weidenbach: Automated Reasoning (Chapter 2). Textbook draft, 2021.

Tseitin Transformation

Proposition 2.5.1 *A formula $H[F]_p$ is satisfiable if and only if $H[Q]_p \wedge (Q \leftrightarrow F)$ is satisfiable, where Q is a new propositional variable that works as an abbreviation for F.*

Proof. "$\Rightarrow$": Suppose that the Π-formula $H[F]_p$ is satisfiable. Let $\mathcal{A}$ be a Π-valuation such that $\mathcal{A}(H[F]_p) = 1$. Let Q be a new propositional variable (that is, a variable that is not contained in Π). Let $\Pi' = \Pi \cup \{Q\}$ and let $\mathcal{A}'$ be the Π'-valuation defined by $\mathcal{A}'(P) = \mathcal{A}(P)$ for all $P \in \Pi$ and $\mathcal{A}'(Q) = \mathcal{A}(F)$. Since $H[F]_p$ is a Π-formula, we have $\mathcal{A}'(H[F]_p) = \mathcal{A}(H[F]_p) = 1$ and $\mathcal{A}'(F) = \mathcal{A}(F)$. Therefore $\mathcal{A}'(Q) = \mathcal{A}'(F)$ and by Prop. 2.3.6 $\mathcal{A}'(H[Q]_p) = \mathcal{A}'(H[F]_p) = 1$, thus $\mathcal{A}'(H[Q]_p \wedge (Q \leftrightarrow F)) = 1$.

"$\Leftarrow$": Let $\Pi' = \Pi \cup \{Q\}$. Suppose that the Π'-formula $H[Q]_p \wedge (Q \leftrightarrow F)$ is satisfiable. Let $\mathcal{A}'$ be a Π'-valuation such that $\mathcal{A}'(H[Q]_p \wedge (Q \leftrightarrow F)) = 1$. Then $\mathcal{A}'(H[Q]_p) = 1$ and $\mathcal{A}'(Q) = \mathcal{A}'(F)$, so by Prop. 2.3.6 $\mathcal{A}'(H[F]_p) = \mathcal{A}'(H[Q]_p) = 1$. □

Satisfiability-preserving CNF transformation (Tseitin 1970):

Apply Prop. 2.5.1 recursively bottom up to all subformulas F in the original formula except $\bot$, $\top$, and literals. This introduces a linear number of new propositional variables Q and definitions $Q \leftrightarrow F$.

Convert the resulting conjunction to CNF. This increases the size only by an additional factor, since each formula $Q \leftrightarrow F$ yields at most four clauses in the CNF.

Example 2.5.2 *We convert the formula* $(P \vee Q) \leftrightarrow R$ *to CNF using the Tseitin transformation. First, we name the subformulas:*

$$\underbrace{\underbrace{(P \vee Q)}_{Q_1} \leftrightarrow R}_{Q_2}.$$

Next, we compute the following equisatisfiable formula:

$$Q_2 \wedge (Q_2 \leftrightarrow (Q_1 \leftrightarrow R)) \wedge (Q_1 \leftrightarrow P \vee Q).$$

Finally, we apply the CNF transformation.

2.6 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).

Preliminaries

Recall:

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

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

Assumptions:

Clauses contain neither duplicated literals nor complementary literals.

The order of literals in a clause is irrelevant.

$\Rightarrow$ Clauses behave like *sets* of literals.

Notation:

We use the notation $C \vee L$ to denote a clause with some literal L and a clause rest C. Here L need *not* be the last literal of the clause and C may be empty.

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

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 in a partial valuation $\mathcal{A}$ (or in 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 in C but one are false in $\mathcal{A}$, 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 in $\mathcal{A}$. 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 1 (0) to P.

P is called a *pure literal.*

The Davis-Putnam-Logemann-Loveland Procedure

boolean DPLL(literal set M, clause set N) {
  if (all clauses in N are true in M) return true;
  elsif (some clause in N is false in M) return false;
  elsif (N contains unit literal L) return DPLL($M \cup \{L\}$, N);
  elsif (N contains pure literal L) return DPLL($M \cup \{L\}$, N);
  else {
    let P be some undefined variable in N;
    if (DPLL($M \cup \{\neg P\}$, N)) return true;
    else return DPLL($M \cup \{P\}$, N);
  }
}

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

Example 2.6.1 *We run the DPLL procedure on the clause set*

$$N = \{\neg P \vee R, \neg Q \vee R, \neg R \vee P \vee Q\}.$$

We start with $M = \emptyset$. Since there are no unit or pure literals, we arbitrarily set R to false: $M := \{\neg R\}$. Then $\neg P \vee R$ contains the unit literal $\neg P$, so $M := \{\neg R, \neg P\}$. Moreover, $\neg Q \vee R$ contains the unit literal $\neg Q$, so $M := \{\neg R, \neg P, \neg Q\}$. At this point, all clauses in N are true in M, so the procedure stops with the model M.

2.7 From DPLL to CDCL

The DPLL procedure can be improved significantly:

The pure literal check is done only while preprocessing (otherwise it is too expensive).

If a conflict is detected, information is reused by conflict analysis and learning.

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

The branching variable is not chosen randomly.

Under certain circumstances, the procedure is restarted.

The improved procedure is called CDCL: conflict-driven clause learning.

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 procedure 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/`.

See also Johannsen's SAT Solving practical.

2.8 OBDDs

Goal:

Efficient manipulation of (equivalence classes of) propositional formulas.

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

$\Rightarrow$ Canonical representation of formulas.

$\Rightarrow$ Satisfiability checking as a side effect.

Literature:

Randal E. Bryant: Graph-Based Algorithms for Boolean Function Manipulation, IEEE Transactions on Computers, 35(8):677-691, 1986.

Randal E. Bryant: Symbolic Boolean Manipulation with Ordered Binary Decision Diagrams, ACM Computing Surveys, 24(3), September 1992, pp. 293-318.

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

BDDs

BDD (Binary decision diagram):

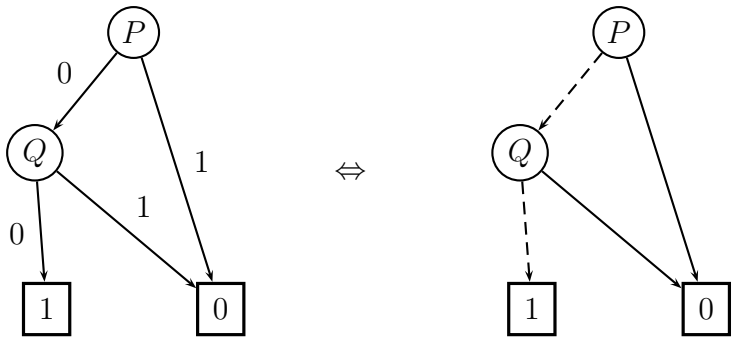
Labeled DAG (directed acyclic graph).

Leaf nodes:

labeled with a truth value (0 or 1).

Nonleaf nodes (inner nodes):

labeled with a propositional variable,
exactly two outgoing edges, labeled with 0 (- - →) and 1 (⟶)



Every BDD node can be interpreted as a mapping from valuations to truth values: Traverse the BDD from the given node to a leaf node; for any node labeled with P, take the 0 edge or 1 edge depending on whether $\mathcal{A}(P)$ is 0 or 1.

$\Rightarrow$ Compact representation of truth tables.

OBDDs

OBDD (Ordered BDD):

Let $<$ be a total ordering of the propositional variables.

An OBDD w.r.t. $<$ is a BDD where every edge from a nonleaf node leads either to a leaf node or to a nonleaf node with a strictly larger label w.r.t. $<$.

OBDDs and formulas:

A leaf node $\boxed{0}$ represents $\bot$ (or any unsatisfiable formula).

A leaf node $\boxed{1}$ represents $\top$ (or any valid formula).

If a nonleaf node v has the label P, and its 0 edge leads to a node representing the formula F_0, and its 1 edge leads to a node representing the formula F_1, then v represents the formula

$$\begin{aligned} F &\models\!\models \text{ if } P \text{ then } F_1 \text{ else } F_0 \\ &\models\!\models (P \wedge F_1) \vee (\neg P \wedge F_0) \\ &\models\!\models (P \rightarrow F_1) \wedge (\neg P \rightarrow F_0) \end{aligned}$$

Conversely:

Define $F\{P \mapsto H\}$ as the formula obtained from F by replacing every occurrence of P in F by H.

For every formula F and propositional variable P:

$$F \models\!\models (P \wedge F\{P \mapsto \top\}) \vee (\neg P \wedge F\{P \mapsto \bot\})$$

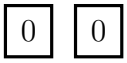
(*Shannon expansion* of F, originally due to Boole).

Consequence: Every formula F can be represented by an OBDD.

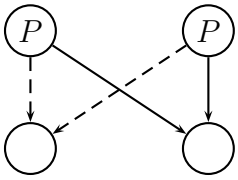
Reduced OBDDs

An OBDD is called *reduced* if it has

- no duplicated leaf nodes



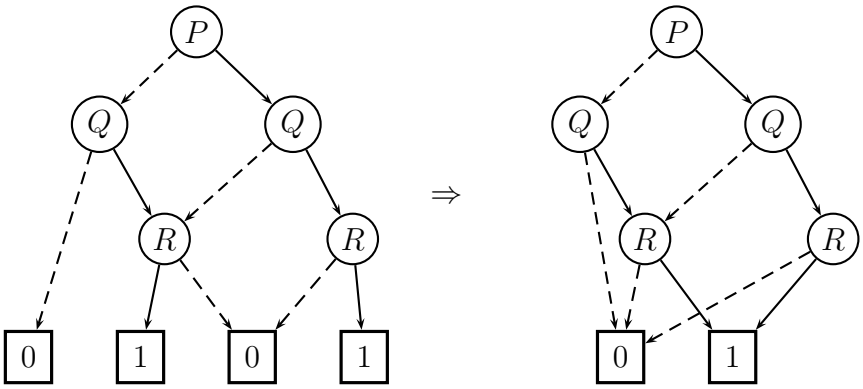
- no duplicated nonleaf nodes

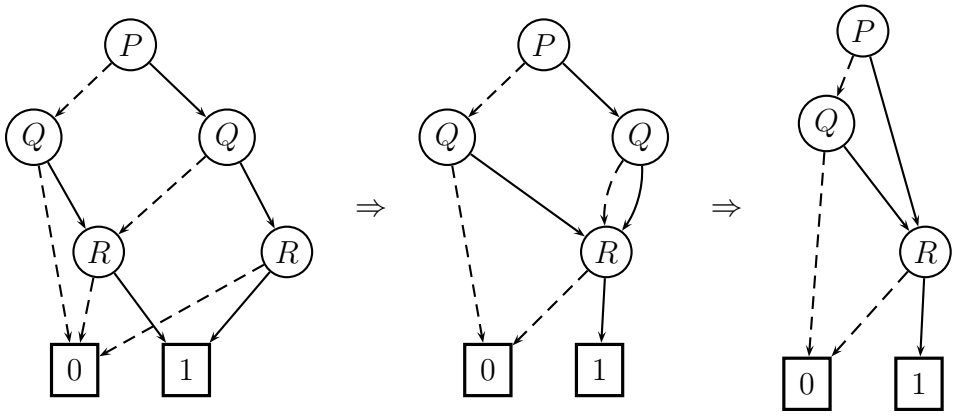


- no redundant tests

Theorem 2.8.1 (Bryant 1986) *Every OBDD can be converted into an equivalent reduced OBDD.*

Example: Reducing an OBDD





Reduced OBDDs

Assumptions from now on:

One fixed ordering $>$.

We consider only reduced OBDDs.

Theorem 2.8.2 (Bryant 1986) *If v and v' are two different nodes in a reduced OBDD, then they represent nonequivalent formulas.*

Proof. We use induction over the maximum of the numbers of nodes reachable from v and v', respectively. Let F and F' be the formulas represented by v and v'.

Case 1: v and v' are nonleaf nodes labeled by different propositional variables P and P'. Without loss of generality, $P < P'$.

Let v_0 and v_1 be the 0-successor and the 1-successor of v, and let F_0 and F_1 be formulas represented by v_0 and v_1. We may assume without loss of generality that all propositional variables occurring in F', F_0, and F_1 are larger than P. By reducedness, $v_0 \neq v_1$, so by induction, $F_0 \not\models\!\models F_1$. Hence there must be a valuation $\mathcal{A}$ such that $\mathcal{A}(F_0) \neq \mathcal{A}(F_1)$. Define valuations $\mathcal{A}_0$ and $\mathcal{A}_1$ by

$$\begin{array}{lll} \mathcal{A}_0(P) = 0 & \mathcal{A}_1(P) = 1 & \\ \mathcal{A}_0(Q) = \mathcal{A}(Q) & \mathcal{A}_1(Q) = \mathcal{A}(Q) & \text{for all } Q \neq P \end{array}$$

We know that the node v represents $F \models\!\!\models (P \wedge F_1) \vee (\neg P \wedge F_0)$, so $\mathcal{A}_0(F) = \mathcal{A}_0(F_0) = \mathcal{A}(F_0)$ and $\mathcal{A}_1(F) = \mathcal{A}_1(F_1) = \mathcal{A}(F_1)$, and therefore $\mathcal{A}_0(F) \neq \mathcal{A}_1(F)$. On the other hand, P does not occur in F', therefore $\mathcal{A}_0(F') = \mathcal{A}_1(F')$. So we must have $\mathcal{A}_0(F) \neq \mathcal{A}_0(F')$ or $\mathcal{A}_1(F) \neq \mathcal{A}_1(F')$, which implies $F \not\models\!\models F'$.

Case 2: v and v' are nonleaf nodes labeled by the same propositional variable.
Case 3: v is a nonleaf node, v' is a nonleaf node, or vice versa.
Case 4: v and v' are different leaf nodes.

Analogously. □

Corollary 2.8.3 *F is valid if and only if it is represented by $\boxed{1}$. F is unsatisfiable if and only if it is represented by $\boxed{0}$.*

Operations on OBDDs

Example:

Let $\circ$ be a binary connective.

Let P be the smallest propositional variable that occurs in F or G or both.

$$\begin{aligned} F \circ G &\models\!\models (P \wedge (F \circ G)\{P \mapsto \top\}) \vee (\neg P \wedge (F \circ G)\{P \mapsto \bot\}) \\ &\models\!\models (P \wedge (F\{P \mapsto \top\} \circ G\{P \mapsto \top\}) \\ &\qquad \vee (\neg P \wedge (F\{P \mapsto \bot\} \circ G\{P \mapsto \bot\}))) \end{aligned}$$

$\Rightarrow$ Straightforward recursive function on OBDD nodes
(needs memoizing for efficient implementation).

The size of the OBDD for $F \circ G$ is bounded by mn, where F has size m and G has size n. (Size = number of nodes.)

With memoization, the time for computing $F \circ G$ is also at most $O(mn)$.

The size of an OBDD for a given formula depends crucially on the chosen ordering of the propositional variables:

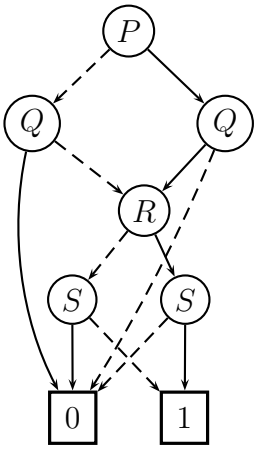
Let $F = (P_1 \wedge P_2) \vee (P_3 \wedge P_4) \vee \cdots \vee (P_{2n-1} \wedge P_{2n})$.

$P_1 < P_2 < P_3 < P_4 < \cdots < P_{2n-1} < P_{2n}$: $2n + 2$ nodes.

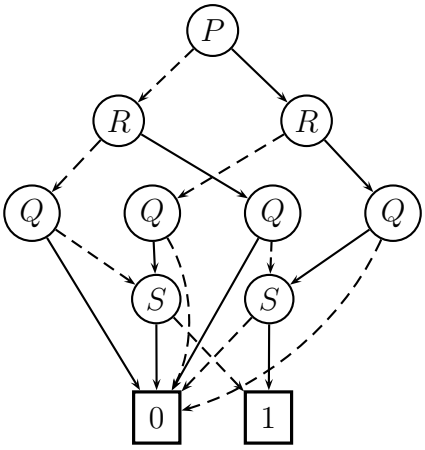
$P_1 < P_3 < \cdots < P_{2n-1} < P_2 < P_4 < \cdots < P_{2n}$: 2^{n+1} nodes.

Example: Variable Ordering in OBDDs

The reduced OBDD for $(P \leftrightarrow Q) \wedge (R \leftrightarrow S)$ with variable ordering $P < Q < R < S$.



The reduced OBDD for $(P \leftrightarrow Q) \wedge (R \leftrightarrow S)$ with variable ordering $P < R < Q < S$.



Operations on OBDDs

Even worse: There are (practically relevant) formulas for which the OBDD has exponential size *for every ordering* of the propositional variables.

2.9 Other Calculi

FRAIGs (functionally reduced and-inverter graphs)
Ordered resolution
Tableau calculus
Hilbert calculus
Sequent calculus
Natural deduction

3 First-Order Logic

First-order logic

- is expressive:
  can be used to formalize mathematical concepts,
  can be used to encode Turing machines,
  but cannot axiomatize natural numbers or uncountable sets,
- has important decidable fragments,
- has interesting logical properties (model and proof theory).

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

3.1 Syntax

Syntax:

- nonlogical symbols (domain-specific)
  $\Rightarrow$ terms, atomic formulas
- logical connectives (domain-independent)
  $\Rightarrow$ boolean combinations, quantifiers

Signatures

A signature $\Sigma = (\Omega, \Pi)$ fixes an alphabet of nonlogical 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 nonconstant function symbols,

P, Q, R, S for predicate symbols.

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

Refined concept for practical applications:
many-sorted signatures (corresponds to simple type systems in programming languages); no big change 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*.

Terms

Terms over Σ and X (Σ-terms) are formed according to these syntactic rules:

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

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

Atoms

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

$$\begin{array}{lcll} A,B & ::= & P(s_1,\ldots,s_m) & , P/m \in \Pi \quad \text{(nonequational atom)} \\ & \left[\,|\right. & (s \approx t) & \qquad\qquad\qquad\qquad \text{(equation)} \left.\,\right] \end{array}$$

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 (see next part). But deductive systems where equality is treated specifically are much more efficient.

Literals

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

Clauses

$$\begin{array}{rcll} C, D & ::= & \bot & \text{(empty clause)} \\ & | & L_1 \vee \cdots \vee L_k, \;\; k \geq 1 & \text{(nonempty clause)} \end{array}$$

General First-Order Formulas

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

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

Notational Conventions

We omit parentheses according to the conventions for propositional logic.

$\forall x_1, \ldots, x_n\, F$ and $\exists x_1, \ldots, x_n\, F$ abbreviate $\forall x_1 \ldots \forall x_n\, F$ and $\exists x_1 \ldots \exists x_n\, F$.

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

Examples:

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

Example: Peano Arithmetic

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

Examples of formulas over this signature are

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

Positions in Terms and Formulas

The set of positions is extended from propositional logic to first-order logic:

The *positions* of a term s (formula F):

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

$\mathrm{pos}(P(t_1, \ldots, t_n)) = \{\varepsilon\} \cup \bigcup_{i=1}^{n} \{ip \mid p \in \mathrm{pos}(t_i)\}$,

$\mathrm{pos}(\forall x\, F) = \{\varepsilon\} \cup \{1p \mid p \in \mathrm{pos}(F)\}$,
$\mathrm{pos}(\exists x\, F) = \{\varepsilon\} \cup \{1p \mid p \in \mathrm{pos}(F)\}$.

The prefix order $\leq$, the subformula (subterm) operator, the formula (term) replacement operator, and the size operator are extended accordingly.

Variables

The *set of variables* occurring in a term t is denoted by $\mathrm{var}(t)$ (and analogously for atoms, literals, clauses, and formulas).

Bound and Free Variables

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

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

Formulas without variables are called *ground*.

Example:

$$\overbrace{\forall y \quad ((\forall x \overbrace{P(x)}^{\text{scope of } \forall x}) \quad \rightarrow \quad R(x, y))}^{\text{scope of } \forall y}$$

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.

Substitutions are mappings

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

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

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

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

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

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

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 recursion over the syntactic structure of t or F by the equations below.

In the presence of quantification it is surprisingly complex: We must not only ensure that bound variables are not replaced by σ. We must also make sure that the (free) variables in the codomain of σ are not *captured* upon placing them into the scope of a quantifier $\mathsf{Q}y$. Hence the bound variable must be renamed into a "fresh," that is, previously unused, variable z.

Application of a Substitution

"Homomorphic" extension of σ to terms and formulas:

$$\begin{aligned}
f(s_1, \ldots, s_n)\sigma &= f(s_1\sigma, \ldots, s_n\sigma) \\
\bot\sigma &= \bot \\
\top\sigma &= \top \\
P(s_1, \ldots, s_n)\sigma &= P(s_1\sigma, \ldots, s_n\sigma) \\
(u \approx v)\sigma &= (u\sigma \approx v\sigma) \\
\neg F\sigma &= \neg(F\sigma) \\
(F \circ G)\sigma &= (F\sigma \circ G\sigma) \quad \text{for each binary connective } \circ \\
(\mathsf{Q}x\, F)\sigma &= \mathsf{Q}z\, (F\, \sigma[x \mapsto z]) \quad \text{with } z \text{ a fresh variable}
\end{aligned}$$

If $s = t\sigma$ for some substitution σ, we call the term s an *instance* of the term t, and we call t a *generalization* of s (analogously for formulas).

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.

Algebras

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.

Σ-algebras generalize the valuations from propositional logic.

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* (over a given Σ-algebra $\mathcal{A}$) is a function $\beta : X \to U_{\mathcal{A}}$.

Variable assignments are the semantic counterparts of substitutions.

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

By recursion we define

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

as follows:

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

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 $a \in U_{\mathcal{A}}$, 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:

$$\begin{aligned} \mathcal{A}(\beta)(\bot) &= 0 \\ \mathcal{A}(\beta)(\top) &= 1 \\ \mathcal{A}(\beta)(P(s_1, \ldots, s_n)) &= \text{if } (\mathcal{A}(\beta)(s_1), \ldots, \mathcal{A}(\beta)(s_n)) \in P_{\mathcal{A}} \text{ then 1 else 0} \\ \mathcal{A}(\beta)(s \approx t) &= \text{if } \mathcal{A}(\beta)(s) = \mathcal{A}(\beta)(t) \text{ then 1 else 0} \\ \mathcal{A}(\beta)(\neg F) &= 1 - \mathcal{A}(\beta)(F) \\ \mathcal{A}(\beta)(F \wedge G) &= \min(\mathcal{A}(\beta)(F), \mathcal{A}(\beta)(G)) \\ \mathcal{A}(\beta)(F \vee G) &= \max(\mathcal{A}(\beta)(F), \mathcal{A}(\beta)(G)) \\ \mathcal{A}(\beta)(F \to G) &= \max(1 - \mathcal{A}(\beta)(F), \mathcal{A}(\beta)(G)) \\ \mathcal{A}(\beta)(F \leftrightarrow G) &= \text{if } \mathcal{A}(\beta)(F) = \mathcal{A}(\beta)(G) \text{ then 1 else 0} \\ \mathcal{A}(\beta)(\forall x\, F) &= \min_{a \in U_{\mathcal{A}}} \{\mathcal{A}(\beta[x \mapsto a])(F)\} \\ \mathcal{A}(\beta)(\exists x\, F) &= \max_{a \in U_{\mathcal{A}}} \{\mathcal{A}(\beta[x \mapsto a])(F)\} \end{aligned}$$

Example

The "standard" interpretation for Peano arithmetic:

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

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

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

Ground Terms and Closed Formulas

If t is a ground term, then $\mathcal{A}(\beta)(t)$ does not depend on β, that is, $\mathcal{A}(\beta)(t) = \mathcal{A}(\beta')(t)$ for every β and β'.

Analogously, if F is a closed formula, then $\mathcal{A}(\beta)(F)$ does not depend on β, that is, $\mathcal{A}(\beta)(F) = \mathcal{A}(\beta')(F)$ for every β and β'.

An element $a \in U_{\mathcal{A}}$ is called *term-generated* if $a = \mathcal{A}(\beta)(t)$ for some ground term t.

In general, not every element of an algebra is term-generated.

3.3 Models, Validity, and Satisfiability

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

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

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

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

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

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 $\beta \in X \to U_{\mathcal{A}}$, we have

$$\mathcal{A}, \beta \models F \quad \Rightarrow \quad \mathcal{A}, \beta \models G$$

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

$$\mathcal{A}, \beta \models F \quad \Leftrightarrow \quad \mathcal{A}, \beta \models G$$

Proposition 3.3.1 *$F \models G$ if and only if $F \to G$ is valid*

Proof. ($\Rightarrow$) Suppose that $(F \to G)$ is not valid. Then there exist an algebra $\mathcal{A}$ and an assignment β such that $\mathcal{A}(\beta)(F \to G) = 0$, which means that $\mathcal{A}(\beta)(F) = 1$ and $\mathcal{A}(\beta)(G) = 0$, or in other words $\mathcal{A}, \beta \models F$ but not $\mathcal{A}, \beta \models G$. Consequently, $F \models G$ does not hold.

($\Leftarrow$) Suppose that $F \models G$ does not hold. Then there exist an algebra $\mathcal{A}$ and an assignment β such that $\mathcal{A}, \beta \models F$ but not $\mathcal{A}, \beta \models G$. Therefore $\mathcal{A}(\beta)(F) = 1$ and $\mathcal{A}(\beta)(G) = 0$, which implies $\mathcal{A}(\beta)(F \to G) = 0$, so $(F \to G)$ is not valid. □

Proposition 3.3.2 *$F \models\!\!\models G$ if and only if $F \leftrightarrow G$ is valid.*

Extension to sets of formulas N as in propositional logic, e.g.:

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

Validity vs. Unsatisfiability

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

Proposition 3.3.3 *Let F and G be formulas, let N be a set of formulas. Then*

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

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

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

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

Substitution Lemma

Lemma 3.3.4 *Let $\mathcal{A}$ be a Σ-algebra, let β be an assignment, let σ be a substitution. Then for any Σ-term t*

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

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

Proof. We use induction over the structure of Σ-terms.

If $t = x$, then $\mathcal{A}(\beta \circ \sigma)(x) = \beta \circ \sigma(x) = \mathcal{A}(\beta)(x\sigma)$ by definition of $\beta \circ \sigma$.

If $t = f(t_1, \ldots, t_n)$, then $\mathcal{A}(\beta \circ \sigma)(f(t_1, \ldots, t_n)) = f_{\mathcal{A}}(\mathcal{A}(\beta \circ \sigma)(t_1), \ldots, \mathcal{A}(\beta \circ \sigma)(t_n)) = f_{\mathcal{A}}(\mathcal{A}(\beta)(t_1\sigma), \ldots, \mathcal{A}(\beta)(t_n\sigma)) = \mathcal{A}(\beta)(f(t_1\sigma, \ldots, t_n\sigma)) = \mathcal{A}(\beta)(f(t_1, \ldots, t_n)\sigma)$ by induction. □

Proposition 3.3.5 *Let $\mathcal{A}$ be a Σ-algebra, let β be an assignment, let σ be a substitution. Then for every Σ-formula F*

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

Corollary 3.3.6 $\mathcal{A}, \beta \models F\sigma \quad \Leftrightarrow \quad \mathcal{A}, \beta \circ \sigma \models F$

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

Two Lemmas

Lemma 3.3.7 *Let $\mathcal{A}$ be a Σ-algebra. Let F be a Σ-formula with free variables $x_1, \ldots, x_n$. Then*

$$\mathcal{A} \models \forall x_1, \ldots, x_n\, F \quad \textit{if and only if} \quad \mathcal{A} \models F\,.$$

Proof. ($\Rightarrow$) Suppose that $\mathcal{A} \models \forall x_1, \ldots, x_n\, F$, that is, $\mathcal{A}(\beta)(\forall x_1, \ldots, x_n\, F) = 1$ for all assignments β. By definition, that means

$$\min_{a_1,\ldots,a_n \in U_\mathcal{A}} \{\mathcal{A}(\beta[x_1 \mapsto a_1, \ldots, x_n \mapsto a_n])(F)\} = 1,$$

and therefore $\mathcal{A}(\beta[x_1 \mapsto a_1, \ldots, x_n \mapsto a_n])(F) = 1$ for all $a_1, \ldots, a_n \in U_\mathcal{A}$.

Let γ be an arbitrary assignment. We have to show that $\mathcal{A}(\gamma)(F) = 1$. For every $i \in \{1, \ldots, n\}$ define $a_i = \gamma(x_i)$, then $\gamma = \gamma[x_1 \mapsto a_1, \ldots, x_n \mapsto a_n]$, and therefore $\mathcal{A}(\gamma)(F) = \mathcal{A}(\gamma[x_1 \mapsto a_1, \ldots, x_n \mapsto a_n])(F) = 1$.

($\Leftarrow$) Suppose that $\mathcal{A} \models F$, that is, $\mathcal{A}(\gamma)(F) = 1$ for all assignments γ.

Then in particular $\mathcal{A}(\beta[x_1 \mapsto a_1, \ldots, x_n \mapsto a_n])(F) = 1$ for all $a_1, \ldots, a_n \in U_\mathcal{A}$ (take $\gamma = \beta[x_1 \mapsto a_1, \ldots, x_n \mapsto a_n]$). Therefore

$$\mathcal{A}(\beta)(\forall x_1, \ldots, x_n\, F) = \min_{a_1,\ldots,a_n \in U_\mathcal{A}} \{\mathcal{A}(\beta[x_1 \mapsto a_1, \ldots, x_n \mapsto a_n])(F)\} = 1.$$

□

Lemma 3.3.8 *Let $\mathcal{A}$ be a Σ-algebra. Let F be a Σ-formula with free variables $x_1, \ldots, x_n$. Let σ be a substitution and let $y_1, \ldots, y_m$ be the free variables of $F\sigma$. Then*

$$\mathcal{A} \models \forall x_1, \ldots, x_n\, F \quad \textit{implies} \quad \mathcal{A} \models \forall y_1, \ldots, y_m\, F\sigma\,.$$

Proof. By the previous lemma, we have $\mathcal{A} \models \forall x_1, \ldots, x_n\, F$ if and only if $\mathcal{A} \models F$ and similarly $\mathcal{A} \models \forall y_1, \ldots, y_m\, F\sigma$ if and only if $\mathcal{A} \models F\sigma$. So it suffices to show that $\mathcal{A} \models F$ implies $\mathcal{A} \models F\sigma$. Suppose that $\mathcal{A} \models F$, that is, $\mathcal{A}(\beta)(F) = 1$ for all assignments β. Then for every assignment γ, we have by Prop. 3.3.5 $\mathcal{A}(\gamma)(F\sigma) = \mathcal{A}(\gamma \circ \sigma)(F) = 1$ (take $\beta = \gamma \circ \sigma$), and therefore $\mathcal{A} \models F\sigma$. □

3.4 Algorithmic Problems

Validity(F): $\models F$?

Satisfiability(F): F satisfiable?

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

Model($\mathcal{A}$,F): $\mathcal{A} \models F$?

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

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

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

Theory of an Algebra

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

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

Problem of axiomatizability:

Given an algebra $\mathcal{A}$ (or a class of algebras) can one *axiomatize* $\mathrm{Th}(\mathcal{A})$, that is, can one write down a formula F (or a semidecidable set F of formulas) such that

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

Two Interesting Theories

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

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

(Non)computability Results

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

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

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 (Traditional)

Prenex formulas have the form

$$\mathsf{Q}_1 x_1 \ldots \mathsf{Q}_n x_n \; F,$$

where F is quantifier-free and $\mathsf{Q}_i \in \{\forall, \exists\}$; we call $\mathsf{Q}_1 x_1 \ldots \mathsf{Q}_n x_n$ the *quantifier prefix* and F the *matrix* of the formula.

Computing prenex normal form by the reduction system $\Rightarrow_P$:

$$\begin{array}{rcl}
H[(F \leftrightarrow G)]_p & \Rightarrow_P & H[(F \to G) \wedge (G \to F)]_p \\
H[\neg \mathsf{Q} x\, F]_p & \Rightarrow_P & H[\overline{\mathsf{Q}} x\, \neg F]_p \\
H[((\mathsf{Q} x\, F) \circ G)]_p & \Rightarrow_P & H[\mathsf{Q} y\, (F\{x \mapsto y\} \circ G)]_p, \\
 & & \quad \circ \in \{\wedge, \vee\} \\
H[((\mathsf{Q} x\, F) \to G)]_p & \Rightarrow_P & H[\overline{\mathsf{Q}} y\, (F\{x \mapsto y\} \to G)]_p, \\
H[(F \circ (\mathsf{Q} x\, G))]_p & \Rightarrow_P & H[\mathsf{Q} y\, (F \circ G\{x \mapsto y\})]_p, \\
 & & \quad \circ \in \{\wedge, \vee, \to\}
\end{array}$$

Here y is always assumed to be some fresh variable and $\overline{\mathsf{Q}}$ denotes the quantifier *dual* to Q, i.e., $\overline{\forall} = \exists$ and $\overline{\exists} = \forall$.

Skolemization

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

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

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

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 3.5.1 *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 $\Sigma = (\Omega, \Pi)$.

The Complete Picture

$$\begin{array}{lll} F & \Rightarrow_P^* & \mathsf{Q}_1 y_1 \ldots \mathsf{Q}_n y_n\, G \qquad (G \text{ quantifier-free}) \\ & \Rightarrow_S^* & \forall x_1, \ldots, x_m\, H \qquad (m \leq n,\ H \text{ quantifier-free}) \\ & \Rightarrow_{CNF}^* & \underbrace{\underbrace{\forall x_1, \ldots, x_m}_{\text{leave out}} \bigwedge_{i=1}^{k} \underbrace{\bigvee_{j=1}^{n_i} L_{ij}}_{\text{clauses } C_i}}_{F'} \end{array}$$

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

Theorem 3.5.2 *Let F be closed. Then $F' \models F$. (The converse is not true in general.)*

Theorem 3.5.3 *Let F be closed. Then F is satisfiable if and only if F' is satisfiable if and only if N is satisfiable.*

Example 3.5.4 *We clausify* $\neg\exists x\,(\forall y\,(P(x,y) \vee Q(y,x)))$*:*

$$\begin{aligned}
& \neg\exists x\,(\forall y\,(P(x,y) \vee Q(y,x))) \\
\Rightarrow_P\ & \forall x\,(\neg\forall y\,(P(x,y) \vee Q(y,x))) \\
\Rightarrow_P\ & \forall x\,\exists y\,(\neg(P(x,y) \vee Q(y,x))) \\
\Rightarrow_S\ & \forall x\,(\neg(P(x,f_1(x)) \vee Q(f_1(x),x))) \\
\Rightarrow_{CNF}\ & \forall x\,(\neg P(x,f_1(x)) \wedge \neg Q(f_1(x),x))
\end{aligned}$$

Thus $N = \{\neg P(x,f_1(x)),\ \neg Q(f_1(x),x)\}$.

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.

- the size of the clausal normal form is exponential when done naively; the transformations we already introduced for propositional logic avoid this exponential growth;
- we want to preserve the original formula structure;
- we want small arity of Skolem functions.

See Nonnengart and Weidenbach 2001 for details.

3.6 Herbrand Interpretations

From now on we will consider first-order logic without equality. We assume that Ω contains at least one constant symbol.

An *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)
- $f_{\mathcal{A}} : (s_1,\ldots,s_n) \mapsto f(s_1,\ldots,s_n),\ \ f/n \in \Omega$

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 3.6.1 *Every set of ground atoms I uniquely determines an Herbrand interpretation $\mathcal{A}$ via*

$$(s_1,\ldots,s_n) \in P_{\mathcal{A}} \ \text{ if and only if } \ P(s_1,\ldots,s_n) \in I$$

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

Existence of Herbrand Models

An Herbrand interpretation I is called an *Herbrand model* of F if $I \models F$.

The importance of Herbrand models lies in the following theorem, which we will prove later in this lecture:

Let N be a set of (universally quantified) Σ-clauses. Then the following properties are equivalent:

(1) N has a model.
(2) N has an Herbrand model (over Σ).
(3) $G_\Sigma(N)$ has an Herbrand model (over Σ).

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

3.7 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

$$\frac{\overbrace{F_1 \quad \cdots \quad F_n}^{\text{premises}}}{\underbrace{F_{n+1}}_{\text{conclusion}}} \quad \textit{side condition}$$

Clausal inference system: Premises and conclusions are clauses. One also considers inference systems over other data structures.

Inference Systems

Inference systems Γ are shorthands for reduction systems over sets of formulas. If N is a set of formulas, then

$$\frac{\overbrace{F_1 \quad \cdots \quad F_n}^{\text{premises}}}{\underbrace{F_{n+1}}_{\text{conclusion}}} \quad \textit{side condition}$$

is a shorthand for

$$N \cup \{F_1, \ldots, F_n\} \quad \Rightarrow_\Gamma \quad N \cup \{F_1, \ldots, F_n\} \cup \{F_{n+1}\}$$
if *side condition*

Proofs

A *proof* in Γ of a formula F from 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 there exists an inference

$$\frac{F_{m_1} \quad \cdots \quad F_{m_n}}{F_i}$$

in Γ, such that $0 \leq m_j < i$, for $1 \leq j \leq n$.

Soundness and Completeness

Provability $\vdash_\Gamma$ of F from N in Γ:

$N \vdash_\Gamma F$ if there exists a proof in Γ of F from N.

Γ is called *sound* if

$$\frac{F_1 \quad \cdots \quad F_n}{F} \in \Gamma \text{ implies } F_1, \ldots, F_n \models F$$

Γ is called *complete* if

$N \models F$ implies $N \vdash_\Gamma F$

Γ is called *refutationally complete* if

$N \models \bot$ implies $N \vdash_\Gamma \bot$

Proposition 3.7.1

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

(ii) If $N \vdash_\Gamma F$ then there exist finitely many $F_1, \ldots, F_n \in N$ such that $F_1, \ldots, F_n \vdash_\Gamma F$.

Reduced Proofs

The definition of a proof of F given above admits sequences $F_1, \ldots, F_k$ of formulas where some F_i are not ancestors of $F_k = F$ (i.e., some F_i are not actually used to derive F).

A proof is called *reduced* if every F_i with $i < k$ is an ancestor of F_k.

We obtain a reduced proof from a proof by marking first F_k and then recursively all the premises used to derive a marked conclusion, and by deleting all nonmarked formulas in the end.

Reduced Proofs as Trees

markings	=	formulas				
leaves	=	assumptions and axioms				
other nodes	=	inferences:	conclusion	=	parent node	
			premises	=	child nodes	

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

Mandatory vs. Admissible Inferences

It is useful to distinguish between two kinds of inferences:

- Mandatory (required) inferences:

  Must be performed to ensure refutational completeness.

  The fewer, the better.

- Optional (admissible) inferences:

  May be performed if useful.

We will first consider only mandatory inferences.

3.8 Ground (or Propositional) Resolution

We observe that propositional clauses and ground clauses are essentially the same, as long as we do not consider equational atoms.

In this section we deal only with ground clauses.

Unlike in Part 2 we admit duplicated literals in clauses, i.e., we treat clauses as multisets of literals, not as sets.

The Resolution Calculus *Res*

Resolution inference rule:

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

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

(Positive) factorization 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, $C = \bot$.

An Example 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)

Soundness of Resolution

Theorem 3.8.1 *Ground first-order resolution is sound.*

Proof. As in propositional logic. □

Note: In ground first-order logic we have (like in propositional logic):

1. $\mathcal{B} \models L_1 \vee \cdots \vee L_n$ if and only if 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.

3.9 Refutational Completeness of Resolution

How to show refutational completeness of ground 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.

Closure of Clause Sets under *Res*

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

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

Proposition 3.9.1

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

(ii) *Res is refutationally complete if and only if for each set* N *of ground clauses:*

$$N \models \bot \;\; \textit{implies} \;\; \bot \in Res^*(N)$$

Proof. (i): We have to show that $Res(Res^*(N)) \subseteq Res^*(N)$, or in other words, that the conclusion of every inference in Res with premises in $Res^*(N)$ is again contained in $Res^*(N)$. An inference in Res is either a resolution inference or a factorization inference. Let us first consider a resolution inference with premises $C_1 \in Res^*(N)$ and $C_2 \in Res^*(N)$ and conclusion C. Since $Res^*(N) = \bigcup_{n\geq 0} Res^n(N)$, we know that there exist $j, k \geq 0$ such that $C_1 \in Res^j(N)$ and $C_2 \in Res^k(N)$. Without loss of generality assume that $j \geq k$. It is easy to see that in this case $Res^k(N) \subseteq Res^j(N)$, hence $C_1 \in Res^j(N)$ and $C_2 \in Res^j(N)$. Consequently, $C \in Res(Res^j(N)) \subseteq Res^{j+1}(N) \subseteq Res^*(N)$.

Otherwise we have a factorization inference with premise $C_1 \in Res^*(N)$ and conclusion C. Again we conclude that $C_1 \in Res^j(N)$ for some $j \geq 0$, hence $C \in Res(Res^j(N)) \subseteq Res^{j+1}(N) \subseteq Res^*(N)$.

(ii) This part follows immediately from the fact that for every clause C we have $N \vdash_{Res} C$ if and only if $C \in Res^*(N)$. □

Orderings

Let $\succ$ be a strict partial ordering on M; let M' be a multiset over M.

$a \in M'$ is called *strictly maximal in* M' if there is no $b \in M' - \{a\}$ with $a \preceq b$.

The notions of maximal and strictly maximal elements coincide except that a maximal element can have duplicates, whereas a strictly maximal element cannot.

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 length-based ordering on atoms when these are viewed as words over a suitable alphabet.)

2. Extend $\succ$ to an *ordering* $\succ_{\text{L}}$ *on ground literals:*

$$\begin{array}{llll} A & \succ_{\text{L}} & B & \text{if } A \succ B \\ A & \succ_{\text{L}} & \neg B & \text{if } A \succ B \\ \neg A & \succ_{\text{L}} & B & \text{if } A \succ B \\ \neg A & \succ_{\text{L}} & \neg B & \text{if } A \succ B \\ \neg A & \succ_{\text{L}} & A & \end{array}$$

3. Extend $\succ_{\text{L}}$ to an *ordering* $\succ_{\text{C}}$ *on ground clauses:*
   $\succ_{\text{C}} = (\succ_{\text{L}})_{\text{mul}}$, the multiset extension of $\succ_{\text{L}}$.

   Notation: $\succ$ also for $\succ_{\text{L}}$ and $\succ_{\text{C}}$.

Example

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

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

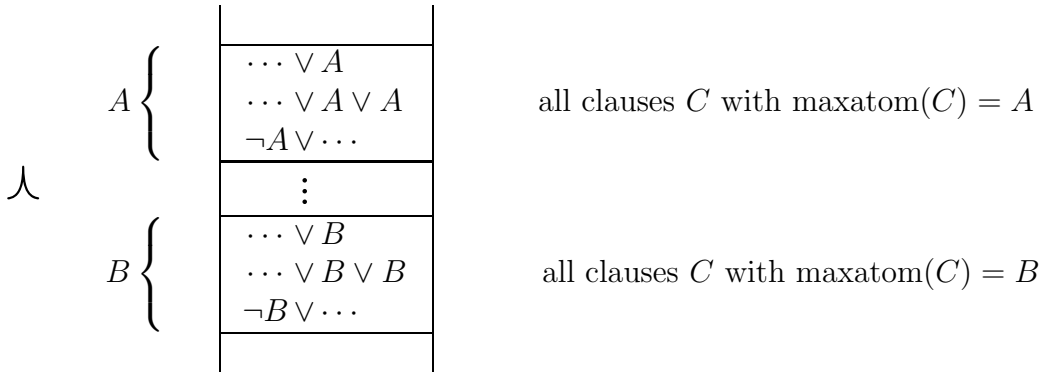
Properties of the Clause Ordering

Proposition 3.9.2

1. *The orderings on literals and clauses are total and well-founded.*
2. *Let C and D be clauses with $A = \text{maxatom}(C)$, $B = \text{maxatom}(D)$, where $\text{maxatom}(C)$ denotes the maximal atom in C.*
   - *(i) If $A \succ B$ then $C \succ D$.*
   - *(ii) If $A = B$ and 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:



Construction of Interpretations

Given: set N of ground clauses, atom ordering $\succ$.

Wanted: Herbrand interpretation I such that

$$I \models N \quad \text{if } N \text{ is saturated and } \bot \notin N$$

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

Main Ideas of the Construction

- Clauses are considered in the order given by $\succ$.
- When considering C, one already has an interpretation so far available (I_C). Initially $I_C = \emptyset$.
- If C is true in this interpretation, nothing needs to to be changed.
- Otherwise, one would like to change the interpretation such that C becomes true.
- Changes should, however, be *monotone*. One never deletes atoms from the interpretation, and the truth value of clauses smaller than C should not change from true to false.
- Hence, one adds $\Delta_C = \{A\}$ 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*). Otherwise, $\Delta_C = \emptyset$.
- We say that the construction fails for a clause C if C is false in I_C and $\Delta_C = \emptyset$.
- We will show: If there are clauses for which the construction fails, then some inference with the smallest such clause (the so-called "minimal counterexample") has not been computed. Otherwise, the limit interpretation is a model of all 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 $\Delta_C = \{A\}$.

Note that the definitions satisfy the conditions of Thm. 1.3.7; so they are well-defined even if $\{D \mid C \succ D\}$ is infinite.

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

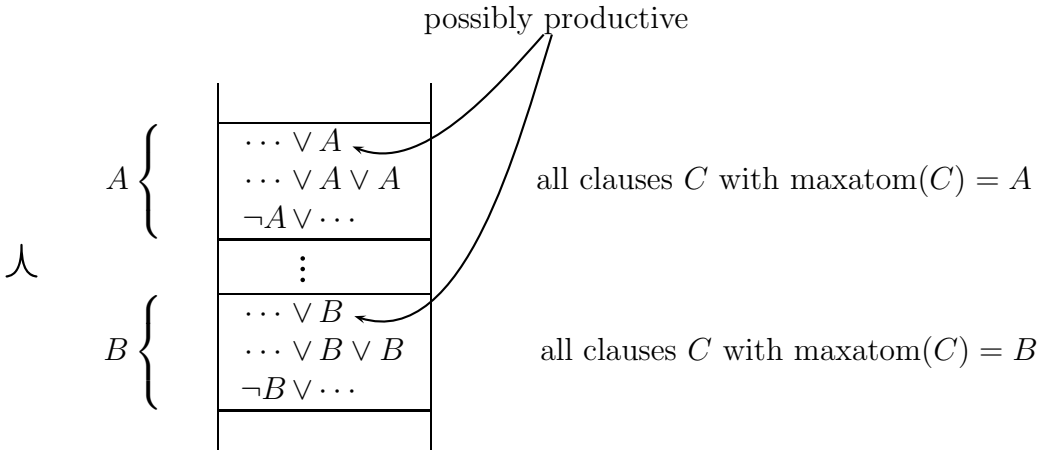
Let $A_5 \succ A_4 \succ A_3 \succ A_2 \succ A_1 \succ A_0$ (max. literals in red).

Iter.	Clause C	I_C	Δ_C	Remarks
0	$\neg A_0$	$\emptyset$	$\emptyset$	true in I_C
1	$A_0 \vee A_1$	$\emptyset$	$\{A_1\}$	A_1 maximal
2	$A_1 \vee A_2$	$\{A_1\}$	$\emptyset$	true in I_C
3	$\neg A_1 \vee A_2$	$\{A_1\}$	$\{A_2\}$	A_2 maximal
4	$A_0 \vee \neg A_1 \vee A_3 \vee A_4$	$\{A_1, A_2\}$	$\{A_4\}$	A_4 maximal
5	$\neg A_1 \vee A_3 \vee \neg A_4$	$\{A_1, A_2, A_4\}$	$\emptyset$	max. lit. $\neg A_4$ neg.; *min. counter-ex.*
6	$\neg A_1 \vee A_5$	$\{A_1, A_2, A_4\}$	$\{A_5\}$	

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

Structure of $N, \succ$

Let $B \succ A$. Note that producing a new atom does not change the truth value of smaller clauses.



Some Properties of the Construction

Proposition 3.9.3

(i) If $D = D' \vee \neg A$, then no $C \succeq D$ produces A.
(ii) If $I_D \models D$, then $I_C \models D$ for every $C \succeq D$ and $I_N^\succ \models D$.
(iii) If $D = D' \vee A$ produces A, then $I_C \models D$ for every $C \succ D$ and $I_N^\succ \models D$.
(iv) If $D = D' \vee A$ produces A, then $I_C \not\models D'$ for every $C \succeq D$ and $I_N^\succ \not\models D'$.
(v) If for every clause $C \in N$, C is productive or $I_C \models C$, then $I_N^\succ \models N$.

Proof. *(i) If C produces A, then $A \succeq L$ for every literal L of C. On the other hand, D contains $\neg A$, and $\neg A \succ A$. Since $\neg A \succ L$ for every literal L of C, we obtain $D \succ C$.*

(ii) Suppose that $I_D \models D$ and $C \succeq D$. If $I_D \models A$ for some positive literal A of D, then $A \in I_D \subseteq I_C \subseteq I_N^\succ$, so $I_C \models D$ and $I_N^\succ \models D$. Otherwise $I_D \models \neg A$ for some negative literal $\neg A$ of D, hence $A \notin I_D$. By (i), no clause that is larger than or equal to D produces A, so $A \notin I_C$ and $A \notin I_N^\succ$. Again, $I_C \models D$ and $I_N^\succ \models D$.

(iii) Obvious, since $C \succ D$ implies $A \in \Delta_D \subseteq I_C \subseteq I_N^\succ$.

(iv) If $D = D' \vee A$ produces A, then $A \succ L$ for every literal L of D' and $I_D \not\models A$. Since $I_D \not\models D$, we have $I_D \not\models L$ for every literal L of D'. Let $C \succeq D$. If L is a positive literal A', then $A' \notin I_D$. Since all atoms in $I_C \setminus I_D$ and $I_N^\succ \setminus I_D$ are larger than or equal to A, we get $A' \notin I_C$ and $A' \notin I_N^\succ$. Otherwise L is a negative literal $\neg A'$, then obviously $A' \in I_D \subseteq I_C \subseteq I_N^\succ$. In both cases L is false in I_C and $I_N^\succ$.

(v) By (ii) and (iii). □

Model Existence Theorem

Proposition 3.9.4 *Let $\succ$ be a clause ordering. If N is saturated w.r.t. Res and $\bot \notin N$, then for every clause $C \in N$, C is productive or $I_C \models C$.*

Proof. Let N be saturated w.r.t. *Res* and $\bot \notin N$. Assume that the proposition does not hold. By well-foundedness, there must exist a minimal clause $C \in N$ (w.r.t. $\succ$) such that C is neither productive nor $I_C \models C$. As $C \neq \bot$ there exists a maximal literal in C. There are two possible reasons why C is not productive:

Case 1: The maximal literal $\neg A$ is negative, i.e., $C = C' \vee \neg A$. Then $I_C \models A$ and $I_C \not\models C'$. So some $D = D' \vee A \in N$ with $C \succ D$ produces A, and $I_C \not\models D'$. The inference

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

yields a clause $D' \vee C' \in N$ that is smaller than C. As $I_C \not\models D' \vee C'$, we know that $D' \vee C'$ is neither productive nor $I_{D' \vee C'} \models D' \vee C'$. This contradicts the minimality of C.

Case 2: The maximal literal A is positive, but not strictly maximal, i.e., $C = C' \vee A \vee A$. Then there is an inference

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

that yields a smaller clause $C' \vee A \in N$. As $I_C \not\models C' \vee A$, this clause is neither productive nor $I_{C' \vee A} \models C' \vee A$. Since $C \succ C' \vee A$, this contradicts the minimality of C. □

Theorem 3.9.5 (Bachmair and Ganzinger 1990) *Let $\succ$ be a clause ordering. If N is saturated w.r.t. Res and $\bot \notin N$, then $I_N^{\succ} \models N$.*

Proof. By Prop. 3.9.4 and part (v) of Prop. 3.9.3. □

Corollary 3.9.6 *Let N be saturated w.r.t. Res. Then $N \models \bot$ if and only if $\bot \in N$.*

Compactness of Propositional Logic

Lemma 3.9.7 *Let N be a set of propositional (or first-order ground) clauses. Then N is unsatisfiable if and only if some finite subset $N' \subseteq N$ is unsatisfiable.*

Proof. The "if" part is trivial. For the "only if" part, assume that N be unsatisfiable. Consequently, $Res^*(N)$ is unsatisfiable as well. By refutational completeness of resolution, $\bot \in Res^*(N)$. So there exists an $n \geq 0$ such that $\bot \in Res^n(N)$, which means that $\bot$ has a finite resolution proof. Now choose N' as the set of assumptions in this proof. □

Theorem 3.9.8 (Compactness for Propositional Formulas) *Let S be a set of propositional (or first-order ground) formulas. Then S is unsatisfiable if and only if some finite subset $S' \subseteq S$ is unsatisfiable.*

Proof. The "if" part is again trivial. For the "only if" part, assume that S be unsatisfiable. Transform S into an equivalent set N of clauses. By the previous lemma, N has a finite unsatisfiable subset N'. Now choose for every clause C in N' one formula F of S such that C is contained in the CNF of F. Let S' be the set of these formulas. □

3.10 General Resolution

Propositional (ground) 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 CDCL procedure.

But in contrast to the CDCL procedure, resolution can be easily extended to nonground clauses.

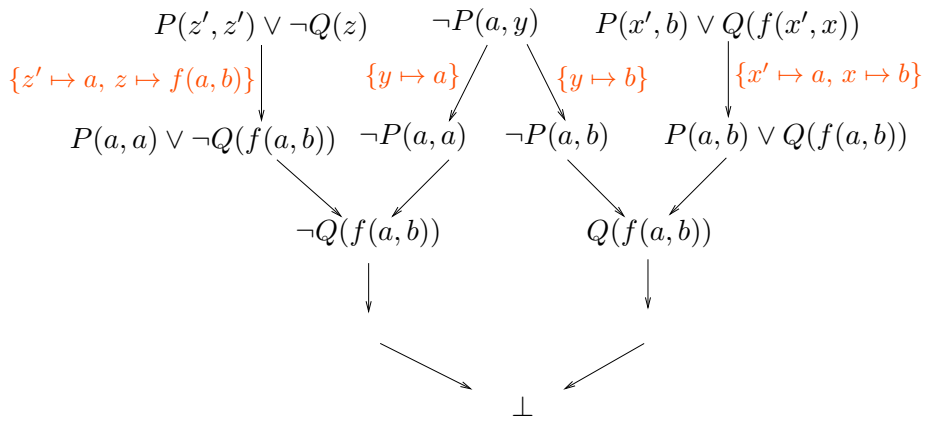
Observation

If $\mathcal{A}$ is a model of an (implicitly universally quantified) clause C, then by Lemma 3.3.8 it is also a model of all (implicitly universally quantified) instances $C\sigma$ of C.

Consequently, if we show that some instances of clauses in a set N are unsatisfiable, then we have also shown that N itself is unsatisfiable.

General Resolution through Instantiation

Idea: instantiate clauses appropriately:



Early approaches (Gilmore 1960, Davis and Putnam 1960):

- Generate ground instances of clauses.
- Try to refute the set of ground instances by resolution.

If no contradiction is found, generate more ground instances.

Problems:

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

Even worse: There are infinitely many possible instances.

Observation:

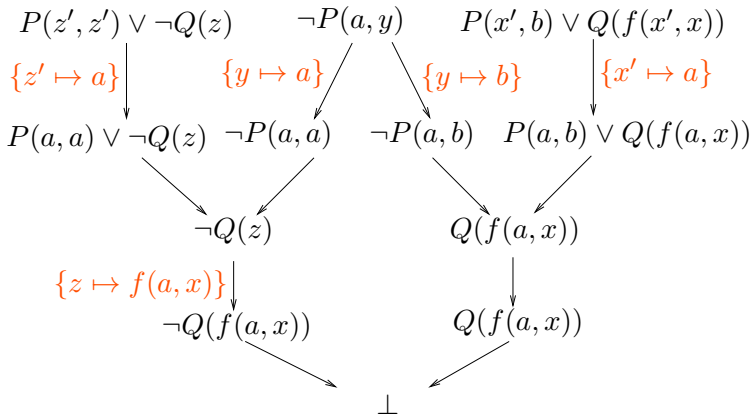
Instantiation must produce complementary literals (so that inferences become possible).

Idea (Robinson 1965):

Do not instantiate more than necessary to get complementary literals
$\Rightarrow$ most general unifiers (mgu).

Calculus works with nonground clauses; inferences with nonground clauses represent infinite sets of ground inferences which are computed simultaneously
$\Rightarrow$ lifting principle.

Computation of instances becomes a by-product of boolean reasoning.



Unification

Let $E = \{s_1 \doteq t_1, \ldots, s_n \doteq t_n\}$ (s_i, t_i terms or atoms) be 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 3.10.1

(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 3.10.2 *σ is idempotent if and only if* $\mathrm{dom}(\sigma) \cap \mathrm{codom}(\sigma) = \emptyset$.

Rule-Based Naive Standard Unification

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

Properties of SU

If $E = \{x_1 \doteq u_1, \ldots, x_k \doteq u_k\}$, with x_i pairwise distinct, $x_i \notin \mathrm{var}(u_j)$, 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 3.10.3 *If E is a solved form then σ_E is an mgu of E.*

Theorem 3.10.4

1. *If $E \Rightarrow_{SU} E'$ then σ is a unifier of E if and only if 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 us treat the case for the fourth 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: $u\sigma = v\sigma$ if and only if $u\{x \mapsto t\}\sigma = v\{x \mapsto t\}\sigma$. (2) and (3) follow by induction from (1) using Proposition 3.10.3. □

Main Unification Theorem

Theorem 3.10.5 *E is unifiable if and only if there is a most general unifier σ of E such that σ is idempotent and* $\mathrm{dom}(\sigma) \cup \mathrm{codom}(\sigma) \subseteq \mathrm{var}(E)$.

Proof. The right-to-left implication is trivial. For the left-to-right implication we observe the following:

- $\Rightarrow_{SU}$ is terminating. 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 nonvariable left-hand sides of equations in E.
- 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 nonunifiability) of E from E' (Theorem 3.10.4, Proposition 3.10.3).
- σ is idempotent because of the substitution in rule 4. $\mathrm{dom}(\sigma) \cup \mathrm{codom}(\sigma) \subseteq \mathrm{var}(E)$, as no new variables are generated.

□

Example of SU

Example 3.10.6 *We unify $g(x, f(x))$ and $g(b, y)$ using standard unification:*

$$\begin{aligned}
& g(x, f(x)) \doteq g(b, y) \\
\Rightarrow_{SU}\ & x \doteq b,\ f(x) \doteq y \\
\Rightarrow_{SU}\ & x \doteq b,\ f(b) \doteq y \\
\Rightarrow_{SU}\ & x \doteq b,\ y \doteq f(b)
\end{aligned}$$

Resulting substitution: $\{x \mapsto b, y \mapsto f(b)\}$.

Exponential Growth of SU

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

Example 3.10.7 *We unify $g(x, y, z)$ and $g(f(y, y), f(z, z), f(a, a))$ using SU:*

$$\begin{aligned}
& g(x, y, z) \doteq g(f(y, y), f(z, z), f(a, a)) \\
\Rightarrow_{SU}\ & x \doteq f(y, y),\ y \doteq f(z, z),\ z \doteq f(a, a) \\
\Rightarrow_{SU}\ & x \doteq f(f(z, z), f(z, z)),\ y \doteq f(z, z),\ z \doteq f(a, a) \\
\Rightarrow_{SU}\ & x \doteq f(f(f(a, a), f(a, a)), f(f(a, a), f(a, a))),\ y \doteq f(f(a, a), f(a, a)), \\
& z \doteq f(a, a)
\end{aligned}$$

Resulting substitution: $\{x \mapsto f(f(f(a, a), f(a, a)), f(f(a, a), f(a, a))), y \mapsto f(f(a, a), f(a, a)), z \mapsto f(a, a)\}$.

Rule-Based Polynomial Unification

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

$$\begin{aligned}
t \doteq t, E \ &\Rightarrow_{PU}\ E \\
f(s_1, \ldots, s_n) \doteq f(t_1, \ldots, t_n), E \ &\Rightarrow_{PU}\ s_1 \doteq t_1, \ldots, s_n \doteq t_n, E \\
f(\ldots) \doteq g(\ldots), E \ &\Rightarrow_{PU}\ \bot \\
& \quad \text{if } f \neq g \\
x \doteq y, E \ &\Rightarrow_{PU}\ x \doteq y, E\{x \mapsto y\} \\
& \quad \text{if } x \in \mathrm{var}(E), x \neq y \\
x_1 \doteq t_1, \ldots, x_n \doteq t_n, E \ &\Rightarrow_{PU}\ \bot \\
& \quad \text{if there are positions } p_i \text{ with} \\
& \quad t_i|_{p_i} = x_{i+1}, t_n|_{p_n} = x_1 \\
& \quad \text{and some } p_i \neq \varepsilon
\end{aligned}$$

$$
\begin{array}{rcl}
x \doteq t, E & \Rightarrow_{PU} & \bot \\
 & & \text{if } x \neq t, x \in \mathrm{var}(t) \\
t \doteq x, E & \Rightarrow_{PU} & x \doteq t, E \\
 & & \text{if } t \notin X \\
x \doteq t, x \doteq s, E & \Rightarrow_{PU} & x \doteq t, t \doteq s, E \\
 & & \text{if } t, s \notin X \text{ and } |t| \leq |s|
\end{array}
$$

Properties of PU

Theorem 3.10.8

1. *If $E \Rightarrow_{PU} E'$ then σ is a unifier of E if and only if 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.*

The solved form of $\Rightarrow_{PU}$ is different from the solved form obtained from $\Rightarrow_{SU}$. 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 $\{x_1 \mapsto t_1\} \circ \cdots \circ \{x_k \mapsto t_k\}$.

Example of PU

Example 3.10.9 *We unify $g(x, f(x))$ and $g(b, y)$ using polynomial unification:*

$$
\begin{array}{rl}
 & g(x, f(x)) \doteq g(b, y) \\
\Rightarrow_{PU} & x \doteq b,\ f(x) \doteq y \\
\Rightarrow_{PU} & x \doteq b,\ y \doteq f(x)
\end{array}
$$

Resulting substitution: $\{x \mapsto b\} \circ \{y \mapsto f(x)\} = \{x \mapsto b,\ y \mapsto f(b)\}$.

Polynomial Growth of PU

Example 3.10.10 *We unify $g(x, y, z)$ and $g(f(y, y), f(z, z), f(a, a))$ using PU:*

$$
\begin{array}{rl}
 & g(x, y, z) \doteq g(f(y, y), f(z, z), f(a, a)) \\
\Rightarrow_{PU} & x \doteq f(y, y),\ y \doteq f(z, z),\ z \doteq f(a, a) \\
= & z \doteq f(a, a),\ y \doteq f(z, z),\ x \doteq f(y, y)
\end{array}
$$

Resulting substitution: $\{z \mapsto f(a, a)\} \circ \{y \mapsto f(z, z)\} \circ \{x \mapsto f(y, y)\}$.

Resolution for General Clauses

We obtain the resolution inference rules for nonground clauses from the inference rules for ground clauses by replacing equality by unifiability:

General resolution *Res*:

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

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

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.

Example 3.10.11 *Consider the clauses*

$$\begin{array}{ll} P(z', z') \vee \neg Q(z) & (1) \\ \neg P(a, y) & (2) \\ P(x', b) \vee Q(f(x', x)) & (3) \end{array}$$

From (1) and (2), using "Resolution" we obtain $\neg Q(z)$ *(4).*

From (3) and (2), using "Resolution" we obtain $Q(f(a, x))$ *(5).*

From (5) and (4), using "Resolution" we obtain the empty clause.

Lifting Lemma

Lemma 3.10.12 *Let* C *and* D *be variable-disjoint clauses. If*

$$\frac{\begin{array}{cc} D & C \\ \downarrow \theta_1 & \downarrow \theta_2 \\ D\theta_1 & C\theta_2 \end{array}}{C'} \qquad \textit{[ground resolution]}$$

then there exists a substitution ρ *such that*

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

An analogous lifting lemma holds for factorization.

Saturation of Sets of General Clauses

Corollary 3.10.13 *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. Without loss of generality, 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 $C' \in Res(G_\Sigma(N))$. Then either (i) there exist resolvable ground instances $D\theta_1$ and $C\theta_2$ of N with resolvent C', or else (ii) C' is a factor of a ground instance $C\theta$ of C.

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

Case (ii): Similar. □

Soundness for General Clauses

Proposition 3.10.14 *The general resolution calculus is sound.*

Proof. We have to show that, if $\sigma = \mathrm{mgu}(A, B)$ then $\{\forall \vec{x}\ (D \vee B),\ \forall \vec{y}\ (C \vee \neg A)\} \models \forall \vec{z}\ (D \vee C)\sigma$ and $\{\forall \vec{x}\ (C \vee A \vee B)\} \models \forall \vec{z}\ (C \vee A)\sigma$.

Let $\mathcal{A}$ be a model of $\forall \vec{x}\ (D \vee B)$ and $\forall \vec{y}\ (C \vee \neg A)$. By Lemma 3.3.8, $\mathcal{A}$ is also a model of $\forall \vec{z}\ (D \vee B)\sigma$ and $\forall \vec{z}\ (C \vee \neg A)\sigma$ and by Lemma 3.3.7, $\mathcal{A}$ is also a model of $(D \vee B)\sigma$ and $(C \vee \neg A)\sigma$. Let β be an assignment. If $\mathcal{A}(\beta)(B\sigma) = 0$, then $\mathcal{A}(\beta)(D\sigma) = 1$. Otherwise $\mathcal{A}(\beta)(B\sigma) = \mathcal{A}(\beta)(A\sigma) = 1$, hence $\mathcal{A}(\beta)(\neg A\sigma) = 0$ and therefore $\mathcal{A}(\beta)(C\sigma) = 1$. In both cases $\mathcal{A}(\beta)((D \vee C)\sigma) = 1$, so $\mathcal{A} \models (D \vee C)\sigma$ and by Lemma 3.3.7, $\mathcal{A} \models \forall \vec{z}\ (D \vee C)\sigma$.

The proof for factorization inferences is similar. □

Herbrand's Theorem

Lemma 3.10.15 *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 3.10.16 *Let N be a set of Σ-clauses, let $\mathcal{A}$ be an* Herbrand *interpretation. Then $\mathcal{A} \models G_\Sigma(N)$ implies $\mathcal{A} \models N$.*

Proof. Let $\mathcal{A}$ be an Herbrand model of $G_\Sigma(N)$. We have to show that $\mathcal{A} \models \forall \vec{x}\ C$ for all clauses $\forall \vec{x}\ C$ in N. This is equivalent to $\mathcal{A} \models C$, which in turn is equivalent to $\mathcal{A}(\beta)(C) = 1$ for all assignments β.

Choose $\beta : X \to U_\mathcal{A}$ arbitrarily. Since $\mathcal{A}$ is an Herbrand interpretation, $\beta(x)$ is a ground term for every variable x, so there is a substitution σ such that $x\sigma = \beta(x)$ for all variables x occurring in C. Now let γ be an arbitrary assignment, then for every variable occurring in C we have $(\gamma \circ \sigma)(x) = \mathcal{A}(\gamma)(x\sigma) = x\sigma = \beta(x)$ and consequently $\mathcal{A}(\beta)(C) = \mathcal{A}(\gamma \circ \sigma)(C) = \mathcal{A}(\gamma)(C\sigma)$. Since $C\sigma \in G_\Sigma(N)$ and $\mathcal{A}$ is an Herbrand model of $G_\Sigma(N)$, we get $\mathcal{A}(\gamma)(C\sigma) = 1$, so $\mathcal{A}$ is a model of C. □

Theorem 3.10.17 (Herbrand) *A set N of Σ-clauses is satisfiable if and only if it has an Herbrand model over Σ.*

Proof. The "⇐" part is trivial. For the "⇒" part let $N \not\models \bot$. Since resolution is sound, this implies that $\bot \notin \mathit{Res}^*(N)$. Obviously, a ground instance of a clause has the same number of literals as the clause itself, so we can conclude that $\bot \notin G_\Sigma(\mathit{Res}^*(N))$. Since $\mathit{Res}^*(N)$ is saturated, $G_\Sigma(\mathit{Res}^*(N))$ is saturated as well by Cor. 3.10.13. Now $I_{G_\Sigma(\mathit{Res}^*(N))}$ is an Herbrand interpretation over Σ and by Thm. 3.9.5 it is a model of $G_\Sigma(\mathit{Res}^*(N))$. By Lemma 3.10.16, every Herbrand model of $G_\Sigma(\mathit{Res}^*(N))$ is a model of $\mathit{Res}^*(N)$. Now $N \subseteq \mathit{Res}^*(N)$, so $I_{G_\Sigma(\mathit{Res}^*(N))} \models N$. □

Corollary 3.10.18 *A set N of Σ-clauses is satisfiable if and only if its set of ground instances $G_\Sigma(N)$ is satisfiable.*

Proof. The "⇒" part follows directly from Lemma 3.10.15. For the "⇐" part assume that $G_\Sigma(N)$ is satisfiable. By Thm. 3.10.17 $G_\Sigma(N)$ has an Herbrand model. By Lemma 3.10.16, every Herbrand model of $G_\Sigma(N)$ is a model of N. □

Refutational Completeness of General Resolution

Theorem 3.10.19 *Let N be a set of general clauses that is saturated w.r.t. Res. Then $N \models \bot$ if and only if $\bot \in N$.*

Proof. The "$\Leftarrow$" part is trivial. For the "$\Rightarrow$" part assume that N is saturated, that is, $Res(N) \subseteq N$. By Corollary 3.10.13, $G_\Sigma(N)$ is saturated as well, i.e., $Res(G_\Sigma(N)) \subseteq G_\Sigma(N)$. By Cor. 3.10.18, $N \models \bot$ implies $G_\Sigma(N) \models \bot$. By the refutational completeness of ground resolution, $G_\Sigma(N) \models \bot$ implies $\bot \in G_\Sigma(N)$, so $\bot \in N$. □

3.11 Theoretical Consequences

We get some classical results on properties of first-order logic as easy corollaries.

The Theorem of Löwenheim-Skolem

Theorem 3.11.1 (Löwenheim–Skolem) *Let Σ be a countable signature and let S be a set of closed Σ-formulas. Then S is satisfiable if and only if 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 3.10.17. □

There exist more refined versions of this theorem. For instance, one can show that if S has some infinite model, then S has a model with a universe of cardinality κ for every κ that is larger than or equal to the cardinalty of the signature Σ.

Compactness of Predicate Logic

Theorem 3.11.2 (Compactness Theorem for First-Order Logic) *Let S be a set of closed first-order formulas. S is unsatisfiable $\Leftrightarrow$ some finite subset $S' \subseteq S$ is unsatisfiable.*

Proof. The "$\Leftarrow$" part is trivial. For the "$\Rightarrow$" part let S be unsatisfiable and let N be the set of clauses obtained by Skolemization and CNF transformation of the formulas in S. Clearly $Res^*(N)$ is unsatisfiable. By Theorem 3.10.19, $\bot \in Res^*(N)$, 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 S' as the subset of formulas in S such that the corresponding clauses contain the assumptions (leaves) of B. □

3.12 Ordered Resolution with Selection

Motivation: Search space for *Res very* large.

Ideas for improvement:

1. In the completeness proof (Model Existence Theorem 3.9.5) 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*

Ordering Restrictions

In the completeness proof one only needs to resolve and factor maximal atoms. Therefore the proof remains correct if we impose ordering restrictions on ground inferences.

(Ground) Ordered Resolution:

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

if $A \succ L$ for all L in D and $\neg A \succeq L$ for all L in C.

(Ground) Ordered Factorization:

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

if $A \succeq L$ for all L in C.

Problem: How to extend this to nonground inferences?

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

In the nonground calculus, we have to consider those literals that correspond to (strictly) maximal literals of ground instances.

An ordering $\succ$ on atoms (or terms) is called *stable under substitutions* if $A \succ B$ implies $A\sigma \succ B\sigma$.

Note:

- We can not require that $A \succ B$ if and only if $A\sigma \succ B\sigma$ for all σ, because this is not computable.
- We can not require that $\succ$ is total on nonground atoms, because this would be incompatible with stability under substitution.

Consequence: In the ordering restrictions for nonground inferences, we have to replace $\succ$ by $\not\preceq$ and $\succeq$ by $\not\prec$.

Ordered Resolution:

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

if $\sigma = \mathrm{mgu}(A, B)$ and $B\sigma \not\preceq L\sigma$ for all L in D and $\neg A\sigma \not\prec L\sigma$ for all L in C.

Ordered Factorization:

$$\frac{C \vee A \vee B}{(C \vee A)\sigma}$$

if $\sigma = \mathrm{mgu}(A, B)$ and $A\sigma \not\prec L\sigma$ for all L in C.

Selection Functions

Selection functions can be used to override ordering restrictions for individual clauses.

A *selection function* is a mapping

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

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 $Res^{\succ}_{\text{sel}}$

The resolution calculus $Res^{\succ}_{\text{sel}}$ is parameterized by

- a selection function sel
- and a well-founded ordering $\succ$ on atoms that is total on ground atoms and stable under substitutions.

(Ground) Ordered Resolution with Selection:

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

if the following conditions are satisfied:

(i) $A \succ L$ for all L in D;

(ii) nothing is selected in $D \vee A$ by sel;

(iii) $\neg A$ is selected in $C \vee \neg A$, or nothing is selected in $C \vee \neg A$ and $\neg A \succeq L$ for all L in C.

(Ground) Ordered Factorization with Selection:

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

if the following conditions are satisfied:

(i) $A \succeq L$ for all L in C;

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

The extension from ground inferences to nonground inferences is analogous to ordered resolution (replace $\succ$ by $\not\preceq$ and $\succeq$ by $\not\prec$). Again we assume that $\succ$ is stable under substitutions.

Ordered Resolution with Selection:

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

if the following conditions are satisfied:

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

(ii) $B\sigma \not\preceq L\sigma$ for all L in D;

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

(iv) $\neg A$ is selected in $C \vee \neg A$, or nothing is selected in $C \vee \neg A$ and $\neg A\sigma \not\prec L\sigma$ for all L in C.

Ordered Factorization with Selection:

$$\frac{C \vee A \vee B}{(C \vee A)\sigma}$$

if the following conditions are satisfied:

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

(ii) $A\sigma \not\prec L\sigma$ for all L in C;

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

Lifting Lemma for $Res^{\succ}_{\mathrm{sel}}$

Lemma 3.12.1 *Let C and D be variable-disjoint clauses. If*

$$\frac{\begin{array}{cc} D & C \\ \downarrow \theta_1 & \downarrow \theta_2 \\ D\theta_1 & C\theta_2 \end{array}}{C'} \quad \textit{[ground inference in } Res^{\succ}_{\mathrm{sel}}\textit{]}$$

and if $\mathrm{sel}(D\theta_1) \simeq \mathrm{sel}(D)$, $\mathrm{sel}(C\theta_2) \simeq \mathrm{sel}(C)$ *(that is, "corresponding" literals are selected), then there exists a substitution ρ such that*

$$\frac{\begin{array}{cc} D & C \end{array}}{\begin{array}{c} C'' \\ \downarrow \rho \\ C' = C''\rho \end{array}} \quad \textit{[inference in } Res^{\succ}_{\mathrm{sel}}\textit{]}$$

An analogous lifting lemma holds for factorization.

Saturation of Sets of General Clauses

Corollary 3.12.2 *Let N be a set of general clauses saturated under $Res^{\succ}_{\mathrm{sel}}$, i.e., $Res^{\succ}_{\mathrm{sel}}(N) \subseteq N$. Then there exists a selection function sel' such that $\mathrm{sel}|_N = \mathrm{sel}'|_N$ and $G_\Sigma(N)$ is also saturated, i.e.,*

$$Res^{\succ}_{\mathrm{sel}'}(G_\Sigma(N)) \subseteq G_\Sigma(N).$$

Proof. We first define the selection function sel' such that $\mathrm{sel}'(C) = \mathrm{sel}(C)$ 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 $\mathrm{sel}'(C)$ to be those occurrences of literals that are ground instances of the occurrences selected by sel in D. Then proceed as in the proof of Cor. 3.10.13 using the lifting lemma above. □

Soundness and Refutational Completeness

Theorem 3.12.3 *Let $\succ$ be an atom ordering and* sel *a selection function such that $Res^{\succ}_{\mathrm{sel}}(N) \subseteq N$. 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 never productive (even if they are false in I_C and if their maximal atom occurs only once and is positive). The result for general clauses follows using Corollary 3.12.2. □

What Do We Gain?

Search spaces become smaller:

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

We assume $P \succ Q$ and sel as indicated by $\boxed{X}$. The maximal literal in a clause is depicted in red.

Rotation redundancy can be avoided:

From

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

we can obtain by *rotation*

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

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

3.13 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 (e.g., if they are tautologies)?

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 $C_1, \ldots, C_n \in N$, $n \geq 0$, such that $C_i \prec C$ and $C_1, \ldots, C_n \models C$.

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: If a ground clause C is redundant and all clauses smaller than C hold in I_C, then C holds in I_C (so C is neither a minimal counterexample nor productive).

Notation: The set of all clauses that are redundant w.r.t. N is denoted by $Red(N)$.

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

Examples of Redundancy

In general, redundancy is undecidable. Decidable approximations are sufficient for us, however.

Proposition 3.13.1 *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.* $N \cup \{C\}$.

(Under certain conditions one may also use nonstrict 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. $Res^{\succ}_{\mathrm{sel}}$) if

$$Res^{\succ}_{\mathrm{sel}}(N \setminus Red(N)) \subseteq N \cup Red(N)$$

Theorem 3.13.2 *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 $I^{\succ}_N$ for $Res^{\succ}_{\mathrm{sel}}$.

If a clause $C \in N$ is redundant, then there exist $C_1, \ldots, C_n \in N$, $n \geq 0$, such that $C_i \prec C$ and $C_1, \ldots, C_n \models C$.

If $I_C \models C_i$ by minimality, then $I_C \models C$.

In particular, C is not productive.

$\Rightarrow$ Redundant clauses are not used as premises for "essential" inferences.

By saturation, the conclusion $D' \vee C'$ of a resolution inference is contained in N (as before) or in $Red(N)$. In the first case, minimality of C ensures that $D' \vee C'$ is productive or $I_{D' \vee C'} \models D' \vee C'$; in the second case, it ensures that $I_{D' \vee C'} \models D' \vee C'$. So in both cases we get a contradiction (analogously for factorization). The rest of the proof works as before.

(ii) Lifting: no additional problems over the proof of Theorem 3.12.3. □

Monotonicity Properties of Redundancy

When we want to delete redundant clauses during a derivation, we have to ensure that redundant clauses *remain redundant* in the rest of the derivation.

Theorem 3.13.3

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

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

Proof. (i) Obvious.

(ii) For ground clause sets N, the well-foundedness of the multiset extension of the clause ordering implies that every clause in $Red(N)$ is entailed by smaller clauses in N that are themselves not in $Red(N)$.

For general clause sets N, the result follows from the fact that every clause in $G_\Sigma(N) \setminus Red(G_\Sigma(N))$ is an instance of a clause in $N \setminus Red(N)$. □

Recall that $Red(N)$ may include clauses that are not in N.

Computing Saturated Sets

Redundancy is preserved when, during a theorem proving derivation one adds new clauses or deletes redundant clauses. This motivates the following definitions:

A *run* of the resolution calculus is a sequence $N_0 \vdash N_1 \vdash N_2 \vdash \cdots$, such that
(i) $N_i \models N_{i+1}$, and
(ii) all clauses in $N_i \setminus N_{i+1}$ are redundant w.r.t. N_{i+1}.

In other words, during a run we may add a new clause if it follows from the old ones, and we may delete a clause if it is redundant w.r.t. the remaining ones.

For a run, we define $N_\infty = \bigcup_{i\geq 0} \bigcap_{j\geq i} N_j$.The set N_∞ of all *persistent* clauses is called the *limit* of the run.

Lemma 3.13.4 *Let $N_0 \vdash N_1 \vdash N_2 \vdash \cdots$ be a run. Then $Red(N_i) \subseteq Red(\bigcup_{i\geq 0} N_i)$ and $Red(N_i) \subseteq Red(N_\infty)$ for every i.*

Proof. Omitted. □

Corollary 3.13.5 *$N_i \subseteq N_\infty \cup Red(N_\infty)$ for every i.*

Proof. If $C \in N_i \setminus N_\infty$, then there is a $k \geq i$ such that $C \in N_k \setminus N_{k+1}$, so C must be redundant w.r.t. N_{k+1}. Consequently, C is redundant w.r.t. N_∞. □

Even if a set N is inconsistent, it could happen that $\bot$ is never derived, because some required inference is never computed.

The following definition rules out such runs:

A run is called *fair* if the conclusion of every inference from clauses in $N_\infty \setminus Red(N_\infty)$ is contained in some $N_i \cup Red(N_i)$.

Lemma 3.13.6 *If a run is fair, then its limit is saturated up to redundancy.*

Proof. If the run is fair, then the conclusion of every inference from nonredundant clauses in N_∞ is contained in some $N_i \cup \mathit{Red}(N_i)$, and therefore contained in $N_\infty \cup \mathit{Red}(N_\infty)$. Hence N_∞ is saturated up to redundancy. □

Theorem 3.13.7 (Refutational Completeness: Dynamic View) *Let $N_0 \vdash N_1 \vdash N_2 \vdash \cdots$ be a fair run, let N_∞ be its limit. Then N_0 has a model if and only if $\bot \notin N_\infty$.*

Proof. ($\Leftarrow$): By fairness, N_∞ is saturated up to redundancy. If $\bot \notin N_\infty$, then it has an Herbrand model. Since every clause in N_0 is contained in N_∞ or redundant w.r.t. N_∞, this model is also a model of $G_\Sigma(N_0)$ and therefore a model of N_0.

($\Rightarrow$): Obvious, since $N_0 \models N_\infty$. □

Simplifications

In theory, the definition of a run permits to add arbitrary clauses that are entailed by the current ones.

In practice, we restrict to two cases:

- We add conclusions of $\mathit{Res}^{\succ}_{\mathrm{sel}}$-inferences from nonredundant premises.
  $\rightsquigarrow$ necessary to guarantee fairness
- We add clauses that are entailed by the current ones if this *makes* other clauses redundant:

  $$N \cup \{C\} \vdash N \cup \{C, D\} \vdash N \cup \{D\}$$
  $$\text{if } N \cup \{C\} \models D \text{ and } C \in \mathit{Red}(N \cup \{D\}).$$

  Net effect: C is *simplified* to D.
  $\rightsquigarrow$ useful to get easier/smaller clause sets

Notation for simplification rules:

$$\frac{C_1 \ \ldots \ C_n}{D_1 \ \ldots \ D_m}$$

means

$$N \cup \{C_1, \ldots, C_n\} \vdash N \cup \{D_1, \ldots, D_m\}$$

Examples of simplification techniques:

- Deletion of duplicated literals:

$$\frac{C \vee L \vee L}{C \vee L}$$

- Subsumption resolution:

$$\frac{D \vee L \qquad C \vee D\sigma \vee \overline{L}\sigma}{D \vee L \qquad C \vee D\sigma}$$

3.14 Hyperresolution

There are *many* variants of resolution.

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 $Res^{\succ}_{\mathrm{sel}}$, the calculus is parameterized by an atom ordering $\succ$ and a selection function sel.

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

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 sel, or 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.

3.15 Implementing Resolution: 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.

The set of clauses is split into two subsets:

- WO = "Worked-off" (or "active") clauses: Have already been selected as "given clause."
- 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 WO;
$New :=$ Conclusions of inferences from $\{D_i \mid i \in I\} \cup C$ where one premise is C;
$U := U \cup New$;
$WO := WO \cup \{C\}$

$\Rightarrow$ At any time, all inferences between clauses in WO have been computed.

$\Rightarrow$ The procedure is fair if no clause remains in U forever.

Additionally:

Try to simplify C using WO. (Skip the remainder of the iteration if C can be eliminated.)

Try to simplify (or even eliminate) clauses from WO using C.

Design decision: should one also simplify U using C?

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 WO 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.)

3.16 Summary: Resolution Theorem Proving

- Resolution is a machine-oriented calculus.
- Using unification, the enumeration of instances becomes a by-product of inference computation.
- Parameters: atom ordering $\succ$ and selection function sel. On the nonground level, ordering constraints can (only) be solved approximately.
- Completeness proof by constructing candidate interpretations from productive clauses $C \vee A$, $A \succ C$.
- *Local* restrictions of inferences via $\succ$ and sel
  $\Rightarrow$ fewer proof variants.
- *Global* restrictions of the search space via redundancy
  $\Rightarrow$ computing with "smaller"/"easier" clause sets.
  (In practice: simplification and detection of redundant clauses uses 90% of the prover runtime.)
- 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.

3.17 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 1960s, independently by Zbigniew Lis and Raymond Smullyan on the basis of work by Gentzen in the 1930s and of Beth in the 1950s.

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 correspond closely to the logical meaning of the symbols.

goal-oriented: inferences operate directly on the goal to be proved.

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
$F \wedge G$	F	G	$\neg(F \wedge G)$	$\neg F$	$\neg G$
$\neg(F \vee G)$	$\neg F$	$\neg G$	$F \vee G$	F	G
$\neg(F \to G)$	F	$\neg G$	$F \to G$	$\neg F$	G

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

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

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 $\{F_1, \ldots, F_n\}$.

Note: We may also consider the *limit tableau* of a tableau expansion; this can be an *infinite* tree.

A *path* (from the root to a leaf) in a tableau is called *closed* if it either contains $\bot$ 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 π in a tableau is called *maximal* if for each formula F on π that is neither a literal nor $\bot$ nor $\top$ there exists a node in π at which the expansion rule for F has been applied.

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

(i) α_1 and α_2 if F is a α-formula,

(ii) β_1 or β_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.

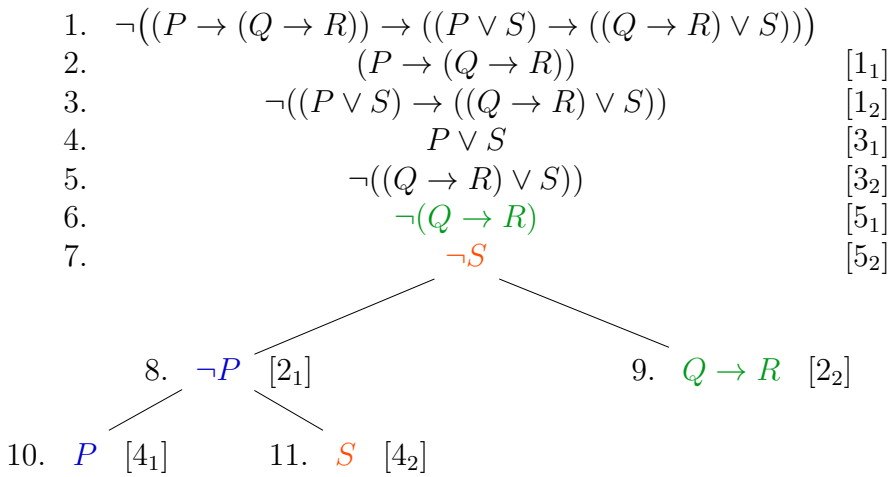
A tableau is called *maximal* if each path is closed or maximal.

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.

An Example Proof

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



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

Proof. ($\Leftarrow$) Trivial, since every path contains in particular $F_1, \ldots, F_n$.
($\Rightarrow$) By induction over the structure of T. □

Corollary 3.17.2 *T closed $\Rightarrow$ $\{F_1, \ldots, F_n\}$ unsatisfiable*

Theorem 3.17.3 *Every strict propositional tableau expansion is finite.*

Proof. New formulas resulting from expansion are $\bot$, $\top$, or subformulas of the expanded formula (modulo de Morgan's law), so the number of formulas that can occur is finite. 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 König's lemma. □

Conclusion: Strict and maximal tableaux can be effectively constructed.

Refutational Completeness

A set $\mathcal{H}$ of propositional formulas is called a Hintikka set if

(1) there is no $P \in \Pi$ with $P \in \mathcal{H}$ and $\neg P \in \mathcal{H}$;

(2) $\bot \notin \mathcal{H}$, $\neg\top \notin \mathcal{H}$;

(3) if $\neg\neg F \in \mathcal{H}$, then $F \in \mathcal{H}$;

(4) if $\alpha \in \mathcal{H}$, then $\alpha_1 \in \mathcal{H}$ and $\alpha_2 \in \mathcal{H}$;

(5) if $\beta \in \mathcal{H}$, then $\beta_1 \in \mathcal{H}$ or $\beta_2 \in \mathcal{H}$.

Lemma 3.17.4 (Hintikka's Lemma) *Every Hintikka set is satisfiable.*

Proof. Let $\mathcal{H}$ be a Hintikka set. Define a valuation $\mathcal{A}$ by $\mathcal{A}(P) = 1$ if $P \in \mathcal{H}$ and $\mathcal{A}(P) = 0$ otherwise. Then show that $\mathcal{A}(F) = 1$ for all $F \in \mathcal{H}$ by induction over the size of formulas. □

Theorem 3.17.5 *Let π be a maximal open path in a tableau. Then the set of formulas on π is satisfiable.*

Proof. We show that set of formulas on π is a Hintikka set: Conditions (3), (4), (5) follow from the fact that π is maximal; conditions (1) and (2) follow from the fact that π is open and from maximality for the second negation elimination rule. □

Theorem 3.17.6 $\{F_1, \ldots, F_n\}$ *satisfiable* $\Leftrightarrow$ *there exists no closed strict tableau for* $\{F_1, \ldots, F_n\}$.

Proof. ($\Rightarrow$) Clear by Cor. 3.17.2.
($\Leftarrow$) Let T be a strict maximal tableau for $\{F_1, \ldots, F_n\}$ and let π be an open path in T. By the previous theorem, the set of formulas on π is satisfiable, and hence by Theorem 3.17.1 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 3.17.5).
- 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 nondeterministically ("*proof confluence*").
- The expansion strategy, however, can have a dramatic impact on the tableau size.

A Variant of the β-Rule

Since $F \vee G \models\!\!\!\models F \vee (G \wedge \neg F)$, the β expansion rule

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

can be replaced by the following variant:

$$\frac{\beta}{\begin{array}{l|l}\beta_1 & \beta_2 \\ & \neg\beta_1\end{array}}$$

The variant β-rule can lead to much shorter proofs, but it is not always beneficial.

In general, it is most helpful if $\neg\beta_1$ can be at most (iteratively) α-expanded.

3.18 Semantic Tableaux for First-Order Logic

There are two ways to extend the tableau calculus to quantified formulas:

- using ground instantiation,
- using free variables.

Tableaux with Ground Instantiation

Classification of quantified formulas:

universal		existential	
γ	$\gamma(t)$	δ	$\delta(t)$
$\forall x F$	$F\{x \mapsto t\}$	$\exists x F$	$F\{x \mapsto t\}$
$\neg\exists x F$	$\neg F\{x \mapsto t\}$	$\neg\forall x F$	$\neg F\{x \mapsto t\}$

Idea:

Replace universally quantified formulas by appropriate ground instances.

γ-expansion

$$\frac{\gamma}{\gamma(t)} \quad \text{where } t \text{ is some ground term}$$

δ-expansion

$$\frac{\delta}{\delta(c)} \quad \text{where } c \text{ is a new Skolem constant}$$

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:

Skolem *constants* are sufficient:
In a δ-formula $\exists x\, F$, $\exists$ is the outermost quantifier and x is the only free variable in F.

Problems:

Having to guess ground terms is impractical.

Even worse, we may have to guess *several* ground instances, since strictness for γ is incomplete. For instance, constructing a closed tableau for

$$\{\forall x\,(P(x) \to P(f(x))),\ P(b),\ \neg P(f(f(b)))\}$$

is impossible without applying γ-expansion twice on one path.

Free-Variable Tableaux

An alternative approach:

Delay the instantiation of universally quantified variables.

Replace universally quantified variables by new free variables.

Intuitively, the free variables are universally quantified *outside* of the entire tableau.

γ-expansion

$$\frac{\gamma}{\gamma(x)} \quad \text{where } x \text{ is a new free variable}$$

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

Application of expansion rules has to be supplemented by a *substitution rule:*

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

One can show that it is sufficient to consider substitutions σ for which there is a path in T containing two *literals* $\neg A$ and B such that $\sigma = \text{mgu}(A, B)$. Such tableaux are called *AMGU-Tableaux*.

Example of a Free-Variable Tableau

1.	$\neg\big(\exists w \forall x\ P(x, w, f(x, w)) \to \exists w \forall x \exists y\ P(x, w, y)\big)$	
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.	$\neg P(b(v_1), v_1, v_3)$	$6(v_3)$ $[\gamma]$

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 $\sigma = \{v_1 \mapsto c,\ v_2 \mapsto b(c),\ v_3 \mapsto f(b(c), c)\}$, and hence, $T\sigma$ is a closed (linear) tableau for the formula in 1.

Free-Variable Tableaux

Problem:

Strictness for γ is still incomplete. For instance, constructing a closed tableau for

$$\{\forall x\,(P(x) \to P(f(x))),\ P(b),\ \neg P(f(f(b)))\}$$

is impossible without applying γ-expansion twice on one path.

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.

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, as follows.

4.1 Handling Equality Naively

Proposition 4.1.1 *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*

$$\forall x\,(x \sim x)$$
$$\forall x, y\,(x \sim y \to y \sim x)$$
$$\forall x, y, z\,(x \sim y \wedge y \sim z \to x \sim z)$$
$$\forall \vec{x}, \vec{y}\,(x_1 \sim y_1 \wedge \cdots \wedge x_n \sim y_n \to f(x_1, \ldots, x_n) \sim f(y_1, \ldots, y_n))$$
$$\forall \vec{x}, \vec{y}\,(x_1 \sim y_1 \wedge \cdots \wedge x_m \sim y_m \wedge P(x_1, \ldots, x_m) \to P(y_1, \ldots, y_m))$$

for every $f/n \in \Omega$ and $P/m \in \Pi$. 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 $\Sigma_1 = (\Omega, \Pi \cup \{\sim/2\})$.

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 $Eq(\Sigma)$.

For the "if" part assume that the Σ_1-algebra $\mathcal{B} = (U_\mathcal{B},\ (f_\mathcal{B} : U_\mathcal{B}^n \to U_\mathcal{B})_{f\in\Omega},\ (P_\mathcal{B} \subseteq U_\mathcal{B}^m)_{P\in\Pi\cup\{\sim\}})$ is a model of $Eq(\Sigma) \cup \{\tilde{F}\}$. Then the interpretation $\sim_\mathcal{B}$ of $\sim$ in $\mathcal{B}$ is a congruence relation on $U_\mathcal{B}$ with respect to the functions $f_\mathcal{B}$ and the predicates $P_\mathcal{B}$.

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 $f_\mathcal{A}([a_1], \ldots, [a_n]) = [f_\mathcal{B}(a_1, \ldots, a_n)]$, and for a predicate symbol $P \in \Pi$, we define $([a_1], \ldots, [a_n]) \in P_\mathcal{A}$ if and only if $(a_1, \ldots, a_n) \in P_\mathcal{B}$. Observe that this is well-defined: If we take different representatives of the same congruence class, we get the same result by congruence of $\sim_\mathcal{B}$. For any $\mathcal{A}$-assignment γ choose some $\mathcal{B}$-assignment β such that $\mathcal{B}(\beta)(x) \in \mathcal{A}(\gamma)(x)$ for every x, then for every Σ-term t we have $\mathcal{A}(\gamma)(t) = [\mathcal{B}(\beta)(t)]$, and analogously for every Σ-formula G, $\mathcal{A}(\gamma)(G) = \mathcal{B}(\beta)(\tilde{G})$. Both properties can easily be shown by structural induction. Therefore, $\mathcal{A}$ is a model of F. □

An analogous proposition holds for *sets* of closed first-order formulas with equality.

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 part: Equations (unit clauses with equality).

  Term rewrite systems.
  Knuth–Bendix completion.

- Next part: Equational clauses.

  Combining resolution and Knuth–Bendix completion. → Superposition.

4.2 Rewrite Systems

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

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

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

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

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

Notation: $l \to r$.

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

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

(Analogously for other properties of abstract reduction systems.)

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

E-Algebras

Let E be a set of 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 us to derive consequences of E:

$$E \vdash t \approx t \quad \text{for every } t \in \mathrm{T}_\Sigma(X) \qquad \textit{(Reflexivity)}$$

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

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

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

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

Lemma 4.2.1 *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 equation 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:

$U_\mathcal{T} = \{[t] \mid t \in \mathrm{T}_\Sigma(X)\}$.

$f_\mathcal{T}([t_1], \ldots, [t_n]) = [f(t_1, \ldots, t_n)]$ for $f/n \in \Omega$.

Lemma 4.2.2 *$f_\mathcal{T}$ is well-defined: If $[t_i] = [t_i']$, then $[f(t_1, \ldots, t_n)] = [f(t_1', \ldots, t_n')]$.*

Proof. Follows directly from the *Congruence* rule for $\vdash$. □

Lemma 4.2.3 *$\mathcal{T} = \mathrm{T}_\Sigma(X)/E$ is an E-algebra.*

Proof. Let $\forall x_1 \ldots x_n (s \approx t)$ 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 $\gamma = \beta[\, x_i \mapsto [v_i] \mid 1 \leq i \leq n\,]$ with $[v_i] \in U_\mathcal{T}$.

Let $\sigma = \{x_1 \mapsto v_1, \ldots, x_n \mapsto v_n\}$, then we get by structural induction that $u\sigma \in \mathcal{T}(\gamma)(u)$ for every $u \in \mathrm{T}_\Sigma(\{x_1, ..., x_n\})$. In particular, $s\sigma \in \mathcal{T}(\gamma)(s)$ 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 4.2.4 *Let X be a countably infinite set of variables; let $s, t \in \mathrm{T}_\Sigma(Y)$. If $\mathrm{T}_\Sigma(X)/E \models \forall \vec{x}\,(s \approx t)$, then $E \vdash s \approx t$ is derivable.*

Proof. Without loss of generality, we assume that all variables in $\vec{x}$ are contained in X. (Otherwise, we rename the variables in the equation. Since X is countably infinite, this is always possible.) 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 $\gamma = \beta[\, x_i \mapsto [v_i] \mid 1 \leq i \leq n\,]$ with $[v_i] \in U_\mathcal{T}$.

Choose $v_i := x_i$, then by structural induction $[u] = \mathcal{T}(\gamma)(u)$ for every $u \in \mathrm{T}_\Sigma(\{x_1, ..., x_n\})$, so $[s] = \mathcal{T}(\gamma)(s) = \mathcal{T}(\gamma)(t) = [t]$. Therefore $E \vdash s \approx t$ is derivable by definition of $\mathcal{T}$. □

Theorem 4.2.5 ("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 4.2.1.

(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 4.2.4. □

4.3 Confluence

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

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

The relation $\to$ is called

Church–Rosser if $b \leftrightarrow^* c$ implies $b \downarrow c$;

confluent if $b \leftarrow^* a \to^* c$ implies $b \downarrow c$;

locally confluent if $b \leftarrow a \to c$ implies $b \downarrow c$;

convergent if it is confluent and terminating.

Theorem 4.3.1 *The following properties are equivalent:*

(i) $\to$ *has the Church–Rosser property.*

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

Proof. (i)⇒(ii): trivial.

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

Lemma 4.3.2 *If* $\to$ *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 $b \leftarrow^* a \to^* c$. If $\to$ is confluent, then $b \to^* d \leftarrow^* c$ for some $d \in A$. Since b and c are normal forms, both derivations must be empty, hence $b \to^0 d \leftarrow^0 c$, so b, c, and d must be identical. □

Corollary 4.3.3 *If* $\to$ *is normalizing and confluent, then every element* b *has a unique normal form.*

Proposition 4.3.4 *If* $\to$ *is normalizing and confluent, then* $b \leftrightarrow^* c$ *if and only if* $b{\downarrow} = c{\downarrow}$.

Proof. Either using Thm. 4.3.1 or directly by induction on the length of the derivation of $b \leftrightarrow^* c$. □

Confluence and Local Confluence

Theorem 4.3.5 ("Newman's Lemma") *If a terminating relation $\to$ is locally confluent, then it is confluent.*

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

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

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

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

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

Rewrite Relations

Corollary 4.3.6 *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 4.3.7 *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 nonvariable 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 without loss of generality 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 $l_i \rightarrow r_i$ ($i \in \{1, 2\}$) be two rewrite rules in a TRS R whose variables have been renamed such that $\mathrm{var}(l_1) \cap \mathrm{var}(l_2) = \emptyset$. (Recall that $\mathrm{var}(l_i) \supseteq \mathrm{var}(r_i)$.)

Let $p \in \mathrm{pos}(l_1)$ be a position such that $l_1|_p$ is not a variable and σ is an mgu of $l_1|_p$ and l_2.

Then $r_1\sigma \leftarrow l_1\sigma \rightarrow (l_1\sigma)[r_2\sigma]_p$.

$\langle r_1\sigma, (l_1\sigma)[r_2\sigma]_p \rangle$ is called a *critical pair* of R.

The critical pair is *joinable* (or: converges) if $r_1\sigma \downarrow_R (l_1\sigma)[r_2\sigma]_p$.

Theorem 4.4.1 ("Critical Pair Theorem") *A TRS R is locally confluent if and only if all its critical pairs are joinable.*

Proof. "only if": Obvious, since joinability of a critical pair is a special case of local confluence.

"if": Suppose s rewrites to t_1 and t_2 using rewrite rules $l_i \rightarrow r_i \in R$ at positions $p_i \in \mathrm{pos}(s)$, where $i \in \{1, 2\}$. 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 ($p_1 \parallel p_2$) or one is a prefix of the other (without loss of generality, $p_1 \leq p_2$).

Case 1: $p_1 \parallel p_2$.

Then $s = s[l_1\theta]_{p_1}[l_2\theta]_{p_2}$, and therefore $t_1 = s[r_1\theta]_{p_1}[l_2\theta]_{p_2}$ and $t_2 = s[l_1\theta]_{p_1}[r_2\theta]_{p_2}$.

Let $t_0 = s[r_1\theta]_{p_1}[r_2\theta]_{p_2}$. Then clearly $t_1 \to_R t_0$ using $l_2 \to r_2$ and $t_2 \to_R t_0$ using $l_1 \to r_1$.

Case 2: $p_1 \leq p_2$.

Case 2.1: $p_2 = p_1q_1q_2$, where $l_1|_{q_1}$ is some variable x.

In other words, the second rewrite step takes place at or below a variable in the first rule. Suppose that x occurs m times in l_1 and n times in r_1 (where $m \geq 1$ and $n \geq 0$).

Then $t_1 \to_R^* t_0$ by applying $l_2 \to r_2$ at all positions $p_1q'q_2$, where q' is a position of x in r_1.

Conversely, $t_2 \to_R^* t_0$ by applying $l_2 \to r_2$ at all positions p_1qq_2, where q is a position of x in l_1 different from q_1, and by applying $l_1 \to r_1$ at p_1 with the substitution θ', where $\theta' = \theta[x \mapsto (x\theta)[r_2\theta]_{q_2}]$.

Case 2.2: $p_2 = p_1p$, where p is a nonvariable 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 4.4.2 *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 4.4.3 *For a finite terminating TRS, confluence is decidable.*

Proof. For every pair of rules and every nonvariable 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 nonconfluent situation $u_1' \leftarrow_R^* u_1 \leftarrow_R s \to_R u_2 \to_R^* u_2'$. □

Critical Pairs: Example

We compute the critical pairs for the following rewrite system and determine whether they are joinable:

$$f(g(f(x))) \to x \quad (1) \qquad f(g(x)) \to g(f(x)) \quad (2)$$

- Between (1) at position 11 and a renamed copy of (1):
  $\sigma = \{x \mapsto g(f(x'))\}$,
  $g(f(x')) \leftarrow f(g(f(g(f(x'))))) \to f(g(x'))$,
  critical pair: $\langle g(f(x')), f(g(x'))\rangle$, joinable at $f(g(x'))$.
- Between (1) at position ε and a renamed copy of (2):
  $\sigma = \{x' \mapsto f(x)\}$,
  $x \leftarrow f(g(f(x))) \to g(f(f(x)))$,
  critical pair: $\langle x, g(f(f(x)))\rangle$, not joinable.
- Between (1) at position 11 and a renamed copy of (2):
  $\sigma = \{x \mapsto g(x')\}$,
  $f(g(g(f(x')))) \leftarrow f(g(f(g(x')))) \to g(x')$,
  critical pair: $\langle f(g(g(f(x')))), g(x')\rangle$, joinable at $g(x')$.

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 4.5.1 *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.

Two 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 focus on case (ii).

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 $\mathrm{T}_\Sigma(X)$ 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 $s, s', t_i \in \mathrm{T}_\Sigma(X)$.

Lemma 4.5.2 *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 $s, s', t \in \mathrm{T}_\Sigma(X)$ and $p \in \mathrm{pos}(t)$.*

Note: *compatible with Σ-operations = compatible with contexts.*

A binary relation $\sqsupset$ over $\mathrm{T}_\Sigma(X)$ is called *stable under substitutions* if $s \sqsupset s'$ implies $s\sigma \sqsupset s'\sigma$ for all $s, s' \in \mathrm{T}_\Sigma(X)$ 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 $\mathrm{T}_\Sigma(X)$ that is a rewrite relation is called *rewrite ordering*.

A well-founded rewrite ordering is called *reduction ordering*.

Theorem 4.5.3 *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. □

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 $s \succ_{\mathcal{A}} t$ if and only if $\mathcal{A}(\beta)(s) \succ \mathcal{A}(\beta)(t)$ for all assignments $\beta : X \to U_{\mathcal{A}}$.

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

Lemma 4.5.4 *$\succ_{\mathcal{A}}$ 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 $\phi : U_{\mathcal{A}}^n \to U_{\mathcal{A}}$ is called *monotone* (with respect to $\succ$) if $a \succ a'$ implies $\phi(b_1, \ldots, a, \ldots, b_n) \succ \phi(b_1, \ldots, a', \ldots, b_n)$ for all $a, a', b_i \in U_{\mathcal{A}}$.

Lemma 4.5.5 *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 $s \succ_{\mathcal{A}} s'$, 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

$$\begin{aligned}\mathcal{A}(\beta)(f(t_1, \ldots, s, \ldots, t_n)) &= f_{\mathcal{A}}(\mathcal{A}(\beta)(t_1), \ldots, \mathcal{A}(\beta)(s), \ldots, \mathcal{A}(\beta)(t_n)) \\ &\succ f_{\mathcal{A}}(\mathcal{A}(\beta)(t_1), \ldots, \mathcal{A}(\beta)(s'), \ldots, \mathcal{A}(\beta)(t_n)) \\ &= \mathcal{A}(\beta)(f(t_1, \ldots, s', \ldots, t_n))\end{aligned}$$

Therefore $f(t_1, \ldots, s, \ldots, t_n) \succ_{\mathcal{A}} f(t_1, \ldots, s', \ldots, t_n)$. □

Theorem 4.5.6 *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 $s_1 \succ_{\mathcal{A}} s_2 \succ_{\mathcal{A}} \ldots$, then it would correspond to an infinite chain $\mathcal{A}(\beta)(s_1) \succ \mathcal{A}(\beta)(s_2) \succ \cdots$ (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 $\mathbb{N}$.

With every function symbol f/n we associate a polynomial $P_f(X_1, \ldots, X_n) \in \mathbb{N}[X_1, \ldots, X_n]$ with coefficients in $\mathbb{N}$ and indeterminates $X_1, \ldots, X_n$. Then we define $f_{\mathcal{A}}(a_1, \ldots, a_n) = P_f(a_1, \ldots, a_n)$ for $a_i \in U_{\mathcal{A}}$.

Requirement 1:

If $a_1, \ldots, a_n \in U_{\mathcal{A}}$, then $f_{\mathcal{A}}(a_1, \ldots, a_n) \in U_{\mathcal{A}}$. (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 $\text{arity}(f) = 0$, then P_f is a constant ≥ 1.

If $\text{arity}(f) = n \geq 1$, then P_f is a polynomial $P(X_1, \ldots, X_n)$ such that every X_i occurs in some monomial $m \cdot X_1^{j_1} \cdots X_k^{j_k}$ with exponent at least 1 and nonzero coefficient $m \in \mathbb{N}$.

$\Rightarrow$ Requirements 1 and 2 are satisfied.

The mapping from function symbols 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 $P_t(X, Y) = 9 + X^2 Y$.

Given polynomials P, Q 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, $s \succ_{\mathcal{A}} t$ if and only if $P_s > P_t$ if and only if $P_s - P_t > 0$.

Question: Can we check $P_s - P_t > 0$ 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 4.5.7 *Hilbert's 10th Problem is undecidable.*

Proposition 4.5.8 *Given a polynomial interpretation and two terms s, t, it is undecidable whether $P_s > P_t$.*

Proof. By reduction of Hilbert's 10th Problem. □

One easy case:

If we restrict to linear polynomials, deciding whether $P_s - P_t > 0$ 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 $P_s(a_1, \ldots, a_n) > P_t(a_1, \ldots, a_n)$ 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 $P_s > P_t$.

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

Define $\rhd_{\mathrm{emb}} = \to^+_{R_{\mathrm{emb}}}$ and $\trianglerighteq_{\mathrm{emb}} = \to^*_{R_{\mathrm{emb}}}$ *("homeomorphic embedding relation").*

$\rhd_{\mathrm{emb}}$ is a simplification ordering.

Lemma 4.5.9 *If $\succ$ is a simplification ordering, then $s \rhd_{\mathrm{emb}} t$ implies $s \succ t$ and $s \trianglerighteq_{\mathrm{emb}} t$ implies $s \succeq t$.*

Proof. Since $\succ$ is transitive and $\succeq$ is transitive and reflexive, it suffices to show that $s \to_{R_{\text{emb}}} t$ implies $s \succ t$. By definition, $s \to_{R_{\text{emb}}} t$ if and only if $s = s[l\sigma]$ and $t = s[r\sigma]$ for some rule $l \to r \in R_{\text{emb}}$. Obviously, $l \rhd r$ for all rules in R_{emb}, hence $l \succ r$. Since $\succ$ is a rewrite relation, $s = s[l\sigma] \succ s[r\sigma] = t$. □

Goal:

Show that every simplification ordering is well-founded (and therefore a reduction ordering).

Note: This works only for *finite* signatures.

To fix this for infinite signatures, the definition of simplification orderings and the definition of embedding have to be modified.

Theorem 4.5.10 ("Kruskal's Theorem") *Let Σ be a finite signature, and 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, pages 113–115. □

Theorem 4.5.11 (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 $t_1 \succ t_2 \succ t_3 \succ \cdots$ 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 irreflexivity.

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 $t_i \succ t_j$. □

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: $s \succ_{\text{lpo}} t$ if

(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 $(s_1, \ldots, s_m)\ (\succ_{\text{lpo}})_{\text{lex}}\ (t_1, \ldots, t_n)$.

where $(\succ_{\text{lpo}})_{\text{lex}}$ is the m-fold lexicographic combination of $\succ_{\text{lpo}}$
(note that $f = g$ implies $m = n$).

Lemma 4.5.12 *$s \succ_{\text{lpo}} t$ implies* $\mathrm{var}(s) \supseteq \mathrm{var}(t)$.

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

Theorem 4.5.13 *$\succ_{\text{lpo}}$ 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, pages 119–120. □

Theorem 4.5.14 *If the precedence $\succ$ is total, then the lexicographic path ordering $\succ_{\text{lpo}}$ is total on ground terms, i.e., for all $s, t \in \mathrm{T}_\Sigma(\emptyset)$: $s \succ_{\text{lpo}} t \vee t \succ_{\text{lpo}} s \vee s = t$.*

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

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: $s \succ_{\text{lpo}} t$ if

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

(2) $s = f(s_1, \ldots, s_m)$, $t = g(t_1, \ldots, t_n)$, and

  (a) $s_i \succeq_{\text{lpo}} t$ for some i, or

(b) $f \succ g$ and $s \succ_{\text{lpo}} t_j$ for all j, or

(c) $f = g$, $s \succ_{\text{lpo}} t_j$ for all j, and $(s_1, \ldots, s_m)\ (\succ_{\text{lpo}})_{\text{lex}}\ (t_1, \ldots, t_n)$.

There are several possibilities to compare subterms in (2)(c):

- compare list of subterms lexicographically left-to-right (*"lexicographic path ordering (lpo),"* Kamin and Lévy)
- compare list of subterms lexicographically right-to-left (or according to some permutation π)
- compare multiset of subterms using the multiset extension (*"multiset path ordering (mpo),"* Dershowitz)
- with each function symbol $f/n \in \Omega$ with $n \geq 1$ associate a status $\in \{\text{mul}\} \cup \{\text{lex}_\pi \mid \pi : \{1, \ldots, n\} \to \{1, \ldots, n\}\}$ and compare according to that status (*"recursive path ordering (rpo) with status"*)

Example 4.5.15 *Consider the following set of equations:*

$$\begin{aligned} f(h(h(x))) &\approx h(f(f(x))) \\ g(g(x)) &\approx f(h(f(h(h(f(x)))))) \\ f(h(x)) &\approx f(f(x)) \end{aligned}$$

Using the lpo with the precedence $g \succ h \succ f$, the left-hand side of each equation is greater than the corresponding right-hand side.

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 $f/1 \in \Omega$, then $f \succ g$ for all $g/n \in \Omega$ with $f \neq g$.

The weight function w can be extended to terms recursively:

$$w(f(t_1, \ldots, t_n)) = w(f) + \sum_{1 \leq i \leq n} w(t_i)$$

or alternatively

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

where $\#(a, t)$ is the number of occurrences of a in t.

The *Knuth–Bendix ordering* $\succ_{\text{kbo}}$ on $\mathrm{T}_\Sigma(X)$ induced by $\succ$ and w is defined by: $s \succ_{\text{kbo}} t$ if

(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 4.5.16 *The Knuth–Bendix ordering induced by $\succ$ and w is a simplification ordering on $\mathrm{T}_\Sigma(X)$.*

Proof. See Baader and Nipkow, pages 125–129. □

Example 4.5.17 *Consider the following set of equations:*

$$\begin{aligned} f(h(h(x))) &\approx h(f(f(x))) \\ g(g(x)) &\approx f(h(f(h(h(f(x)))))) \\ f(h(x)) &\approx f(f(x)) \end{aligned}$$

Using the kbo with weight 100 for g, weight 10 for h, weight 1 for f and variables, and an arbitrary precedence, the left-hand side of each equation is greater than the corresponding right-hand side.

Remark

If $\Pi \neq \emptyset$, then all the term orderings described in this section can also be used to compare nonequational 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.)

Knuth–Bendix Completion: Idea

How to ensure termination?

Fix a reduction ordering $\succ$ and construct R in such a way that $\to_R \subseteq \succ$ (i.e., $l \succ r$ for every $l \to r \in R$).

How to ensure confluence?

Check that all critical pairs are joinable.

Note: Every critical pair $\langle s, t \rangle$ can be *made* joinable by adding $s \to t$ or $t \to s$ to R.

(Actually, we first add $s \approx t$ to E and later try to turn it into a rule that is contained in $\succ$; this gives us more freedom.)

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: $E_0, R_0 \vdash E_1, R_1 \vdash E_2, R_2 \vdash \cdots$.

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:

$$\frac{E \cup \{s \mathrel{\dot{\approx}} t\}, \quad R}{E, \quad R \cup \{s \to t\}} \qquad \text{if } s \succ t$$

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 do not 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 strictly necessary, but are very useful (e.g., to eliminate joinable critical pairs and to cope 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:

$$\frac{E, \quad R \cup \{s \rightarrow t\}}{E \cup \{u \approx t\}, \quad R} \qquad \begin{array}{l}\text{if } s \rightarrow_R u \text{ using a rule } l \rightarrow r \in R \\ \text{such that } s \sqsupset l \text{ (see below).}\end{array}$$

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* $\supseteq$ is defined by

$$s \supseteq l \quad \text{if} \quad s|_p = l\sigma \text{ for some } p \text{ and } \sigma$$

and $\sqsupset = \supseteq \setminus \sqsubseteq$ is the strict part of $\supseteq$.

Lemma 4.6.1 *$\sqsupset$ is a well-founded strict partial ordering.*

Lemma 4.6.2 *If $E, R \vdash E', R'$, then $\approx_{E \cup R} = \approx_{E' \cup R'}$.*

Lemma 4.6.3 *If $E, R \vdash E', R'$ and $\rightarrow_R \subseteq \succ$, then $\rightarrow_{R'} \subseteq \succ$.*

Note: Like in ordered resolution, simplification should be preferred to deduction:

- Simplify/delete whenever possible.
- Otherwise, orient an equation if possible.
- Last resort: compute critical pairs.

Knuth–Bendix Completion: Example

We apply the Knuth–Bendix procedure to the set of equations

$$add(zero, zero) \approx zero \quad (1) \qquad add(x, succ(y)) \approx succ(add(x, y)) \quad (2)$$
$$add(succ(x), y) \approx succ(add(x, y)) \quad (3)$$

using the lpo with the precedence $add \succ succ \succ zero$.

We first apply "Orient" to (1)–(3), resulting in the rewrite rules

$$add(zero, zero) \to zero \quad (4) \qquad add(x, succ(y)) \to succ(add(x, y)) \quad (5)$$
$$add(succ(x), y) \to succ(add(x, y)) \quad (6)$$

$$add(zero, zero) \to zero \quad (4) \qquad add(x, succ(y)) \to succ(add(x, y)) \quad (5)$$
$$add(succ(x), y) \to succ(add(x, y)) \quad (6)$$

Then we apply "Deduce" between (5) and a renamed copy of (6):

$$succ(add(succ(x), y)) \approx succ(add(x, succ(y))) \quad (7)$$

We can now apply "Simplify-Eq" to both sides of (7) using (6) and (5):

$$succ(succ(add(x, y))) \approx succ(succ(add(x, y))) \quad (8)$$

This last equation is trivial and can be deleted using "Delete."

All critical pairs have been checked. The resulting term rewrite system is $\{(4), (5), (6)\}$.

Knuth–Bendix Completion: Correctness Proof

What can happen if we run the completion procedure on a set E of equations?

(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.

To treat these cases simultaneously, we need some definitions.

A (finite or infinite sequence) $E_0, R_0 \vdash E_1, R_1 \vdash E_2, R_2 \vdash \cdots$ with $R_0 = \emptyset$ is called a *run* of the completion procedure with input E_0 and $\succ$.

For a run, $E_\cup = \bigcup_{i \geq 0} E_i$ and $R_\cup = \bigcup_{i \geq 0} R_i$.

The sets of *persistent equations or rules* of the run are $E_\infty = \bigcup_{i \geq 0} \bigcap_{j \geq i} E_j$ and $R_\infty = \bigcup_{i \geq 0} \bigcap_{j \geq i} R_j$.

Note: If the run is finite and ends with E_n, R_n, then $E_\infty = E_n$ and $R_\infty = R_n$.

A run is called *fair* if $CP(R_\infty) \subseteq E_\cup$ (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 $\approx_{E_0} = \approx_{E_\cup \cup R_\cup} = \leftrightarrow^*_{E_\cup \cup R_\cup} = \downarrow_{R_\infty}$.

General assumptions from now on:

$E_0, R_0 \vdash E_1, R_1 \vdash E_2, R_2 \vdash \cdots$ is a fair run.

R_0 and E_∞ are empty.

A *proof* of $s \approx t$ in $E_\cup \cup R_\cup$ is a finite sequence $(s_0, \ldots, s_n)$ such that $s = s_0$, $t = s_n$, and for all $i \in \{1, \ldots, n\}$:

(1) $s_{i-1} \leftrightarrow_{E_\cup} s_i$, or

(2) $s_{i-1} \rightarrow_{R_\cup} s_i$, or

(3) $s_{i-1} \leftarrow_{R_\cup} s_i$.

The pairs (s_{i-1}, s_i) are called *proof steps.*

A proof is called a *rewrite proof in* R_∞ if there is a $k \in \{0, \ldots, n\}$ such that $s_{i-1} \rightarrow_{R_\infty} s_i$ for $1 \leq i \leq k$ and $s_{i-1} \leftarrow_{R_\infty} s_i$ for $k+1 \leq i \leq n$

Idea (Bachmair, Dershowitz, Hsiang):

Define a well-founded ordering on proofs such that for every proof that is not a rewrite proof in R_∞ there is an equivalent smaller proof.

Consequence: For every proof there is an equivalent rewrite proof in R_∞.

We associate a *cost* $c(s_{i-1}, s_i)$ with every proof step as follows:

(1) If $s_{i-1} \leftrightarrow_{E_\cup} s_i$, 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 $s_{i-1} \to_{R_\cup} s_i$ using $l \to r$, then $c(s_{i-1}, s_i) = (\{s_{i-1}\}, l, s_i)$.

(3) If $s_{i-1} \leftarrow_{R_\cup} s_i$ using $l \to r$, 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* $\succ_{\mathrm{C}}$ compares the costs of proofs using the multiset extension of the proof step ordering.

Lemma 4.6.4 *$\succ_{\mathrm{C}}$ is a well-founded ordering.*

Lemma 4.6.5 *Let P be a proof in $E_\cup \cup R_\cup$. If P is not a rewrite proof in R_∞, then there exists an equivalent proof P' in $E_\cup \cup R_\cup$ such that $P \succ_{\mathrm{C}} P'$.*

Proof. If P is not a rewrite proof in R_∞, then it contains

(a) a proof step that is in $E_\cup$, or
(b) a proof step that is in $R_\cup \setminus R_\infty$, or
(c) a subproof $s_{i-1} \leftarrow_{R_\infty} s_i \to_{R_\infty} s_{i+1}$ (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 $s \mathrel{\dot{\approx}} t$ is in $E_\cup$. This equation must be deleted during the run.

If $s \mathrel{\dot{\approx}} t$ is deleted using *Orient*:

$$\ldots s_{i-1} \leftrightarrow_{E_\cup} s_i \ldots \quad \Longrightarrow \quad \ldots s_{i-1} \to_{R_\cup} s_i \ldots$$

If $s \mathrel{\dot{\approx}} t$ is deleted using *Delete*:

$$\ldots s_{i-1} \leftrightarrow_{E_\cup} s_{i-1} \ldots \quad \Longrightarrow \quad \ldots s_{i-1} \ldots$$

If $s \mathrel{\dot{\approx}} t$ is deleted using *Simplify-Eq*:

$$\ldots s_{i-1} \leftrightarrow_{E_\cup} s_i \ldots \quad \Longrightarrow \quad \ldots s_{i-1} \to_{R_\cup} s' \leftrightarrow_{E_\cup} s_i \ldots$$

Case (b): A proof step using a rule $s \to t$ is in $R_\cup \setminus R_\infty$. This rule must be deleted during the run.

If $s \to t$ is deleted using *R-Simplify-Rule*:

$$\ldots s_{i-1} \to_{R_\cup} s_i \ldots \quad \Longrightarrow \quad \ldots s_{i-1} \to_{R_\cup} s' \leftarrow_{R_\cup} s_i \ldots$$

If $s \to t$ is deleted using *L-Simplify-Rule*:

$$\ldots s_{i-1} \to_{R_\cup} s_i \ldots \quad \Longrightarrow \quad \ldots s_{i-1} \to_{R_\cup} s' \leftrightarrow_{E_\cup} s_i \ldots$$

Case (c): A subproof has the form $s_{i-1} \leftarrow_{R_\infty} s_i \to_{R_\infty} s_{i+1}$.

If there is no overlap or a noncritical overlap:

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

If there is a critical pair that has been added using "Deduce":

$$\dots s_{i-1} \leftarrow_{R_\infty} s_i \rightarrow_{R_\infty} s_{i+1} \dots \Longrightarrow \quad \dots s_{i-1} \leftrightarrow_{E_\cup} s_{i+1} \dots$$

In all cases, checking that the replacement subproof is smaller than the replaced subproof is routine. □

Theorem 4.6.6 *Let $E_0, R_0 \vdash E_1, R_1 \vdash E_2, R_2 \vdash \cdots$ be a fair run and let R_0 and E_∞ be empty. Then*

(1) every proof in $E_\cup \cup R_\cup$ 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 $\approx_{E_\cup \cup R_\cup} = \approx_{E_0}$. Since $R_\infty \subseteq R_\cup$, we get $\approx_{R_\infty} \subseteq \approx_{E_\cup \cup R_\cup}$. On the other hand, by (1), $\approx_{E_\cup \cup R_\cup} \subseteq \approx_{R_\infty}$.

(3) Since $\rightarrow_{R_\infty} \subseteq \succ$, R_∞ is terminating. By (1), R_∞ is confluent. □

4.7 Unfailing Completion

Classical completion:

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

Fail if an equation can be neither 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.

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 Superposition

First-order calculi considered so far:

Resolution: for first-order clauses without equality.

(Unfailing) Knuth–Bendix completion: for unit equations.

Goal:

Combine the ideas of ordered resolution (overlap maximal literals in a clause) and Knuth–Bendix completion (overlap maximal sides of equations) to get a calculus for equational clauses.

5.1 Recapitulation

First-order logic:

Atom: either $P(s_1, \ldots, s_m)$ with $P \in \Pi$ or $s \approx t$.

Literal: atom or negated atom.

Clause: (possibly empty) disjunction of literals (all variables implicitly universally quantified).

Refutational theorem proving:

For refutational theorem proving, it suffices to consider sets of clauses: every first-order formula F can be translated into a set of clauses N such that F is unsatisfiable if and only if N is unsatisfiable.

In the nonequational case, unsatisfiability can for instance be checked using the (ordered) resolution calculus.

(Ordered) resolution: inference rules:

	Ground case:	Nonground case:
Resolution:	$\dfrac{D' \vee A \qquad C' \vee \neg A}{D' \vee C'}$	$\dfrac{D' \vee A \qquad C' \vee \neg A'}{(D' \vee C')\sigma}$ where $\sigma = \mathrm{mgu}(A, A')$.
Factoring:	$\dfrac{C' \vee A \vee A}{C' \vee A}$	$\dfrac{C' \vee A \vee A'}{(C' \vee A)\sigma}$ where $\sigma = \mathrm{mgu}(A, A')$.

Ordering restrictions:

Let $\succ$ be a well-founded and total ordering on ground atoms.

Literal ordering $\succ_{\mathrm{L}}$: compares literals by comparing lexicographically first the respective atoms using $\succ$ and then their polarities (negative $>$ positive).

Clause ordering $\succ_{\mathrm{C}}$: compares clauses by comparing their multisets of literals using the multiset extension of $\succ_{\mathrm{L}}$.

Ordering restrictions (ground case):

Inference are necessary only if the following conditions are satisfied:

- The left premise of a "Resolution" inference is not larger than or equal to the right premise.
- The literals that are involved in the inferences ($[\neg]\, A$) are maximal in the respective clauses (strictly maximal for the left premise of "Resolution").

Ordering restrictions (nonground case):

Define the atom ordering $\succ$ also for nonground atoms.

Need stability under substitutions: $A \succ B$ implies $A\sigma \succ B\sigma$.

Note: $\succ$ cannot be total on nonground atoms.

For literals involved in inferences, we have the same maximality requirements as in the ground case.

Resolution is (even with ordering restrictions) refutationally complete:

Dynamic view of refutational completeness:

If N is unsatisfiable ($N \models \bot$), then *fair* derivations from N produce $\bot$.

Static view of refutational completeness:

If N is *saturated*, then N is unsatisfiable if and only if $\bot \in N$.

Proving refutational completeness for the ground case:

We have to show:

If N is saturated (i.e., if sufficiently many inferences have been computed), and $\bot \notin N$, then N has a model.

Constructing a candidate interpretation:

Suppose that N be saturated and $\bot \notin N$. We inspect all clauses in N in ascending order and construct a sequence of Herbrand interpretations (starting with the empty interpretation—all atoms are false).

If a clause C is false in the current interpretation, and has a positive and strictly maximal literal A, then extend the current interpretation such that C becomes true: add A to the current interpretation. (Then C is called *productive*.)

Otherwise, leave the current interpretation unchanged.

The sequence of interpretations has the following properties:

(1) If an atom is true in some interpretation, then it remains true in all future interpretations.

(2) If a clause is true at the time where it is inspected, then it remains true in all future interpretations.

(3) If a clause $C = C' \vee A$ is productive, then C remains true and C' remains false in all future interpretations.

Show by induction: If N is saturated and $\bot \notin N$, then every clause in N is either true at the time when it is inspected or productive.

Note:
For the induction proof, it is not necessary that the conclusion of an inference is contained in N. It is sufficient that it is redundant w.r.t. N.

N is called *saturated up to redundancy* if the conclusion of every inference from clauses in $N \setminus \mathit{Red}(N)$ is contained in $N \cup \mathit{Red}(N)$.

Proving refutational completeness for the nonground case:

If $C_i\theta$ is a ground instance of the clause C_i for $i \in \{0, \ldots, n\}$ and

$$\frac{C_n \quad \cdots \quad C_1}{C_0}$$

and

$$\frac{C_n\theta, \ldots, C_1\theta}{C_0\theta}$$

are inferences, then the latter inference is called a *ground instance* of the former.

For a set N of clauses, let $G_\Sigma(N)$ be the set of all ground instances of clauses in N.

Construct the interpretation from the set $G_\Sigma(N)$ of all ground instances of clauses in N:

N is saturated and does not contain $\bot$
$\Rightarrow$ $G_\Sigma(N)$ is saturated and does not contain $\bot$
$\Rightarrow$ $G_\Sigma(N)$ has a Herbrand model I
$\Rightarrow$ I is a model of N.

It is possible to encode an arbitrary predicate P using a function f_P and a new constant *true*:

$$\begin{array}{rcl} P(t_1, \ldots, t_n) & \rightsquigarrow & f_P(t_1, \ldots, t_n) \approx \mathit{true} \\ \neg\, P(t_1, \ldots, t_n) & \rightsquigarrow & \neg\, f_P(t_1, \ldots, t_n) \approx \mathit{true} \end{array}$$

In equational logic it is therefore sufficient to consider the case that $\Pi = \emptyset$, i.e., equality is the only predicate symbol.

Abbreviation: $s \not\approx t$ instead of $\neg\, s \approx t$.

5.2 The Superposition Calculus—Informally

Conventions:

From now on: $\Pi = \emptyset$ (equality is the only predicate).

Inference rules are to be read modulo symmetry of the equality symbol.

We will first explain the ideas and motivations behind the superposition calculus and its completeness proof. Precise definitions will be given later.

Ground inference rules:

Pos. Superposition:
$$\frac{D' \vee t \approx t' \qquad C' \vee s[t] \approx s'}{D' \vee C' \vee s[t'] \approx s'}$$

Neg. Superposition:
$$\frac{D' \vee t \approx t' \qquad C' \vee s[t] \not\approx s'}{D' \vee C' \vee s[t'] \not\approx s'}$$

Equality Resolution:
$$\frac{C' \vee s \not\approx s}{C'}$$

(Note: We will need one further inference rule.)

Ordering wishlist:

Like in resolution, we want to perform only inferences between (strictly) maximal literals.

Like in completion, we want to perform only inferences between (strictly) maximal sides of literals.

Like in resolution, in inferences with two premises, the left premise should not be larger than the right one.

Like in resolution and completion, the conclusion should then be smaller than the larger premise.

The ordering should be total on ground literals.

Consequences:

The literal ordering must depend primarily on the larger term of an equation.

As in the resolution case, negative literals must be slightly larger than the corresponding positive literals.

Additionally, we need the following property: If $s \succ t \succ u$, then $s \not\approx u$ must be larger than $s \approx t$. In other words, we must compare first the larger term, then the polarity, and finally the smaller term.

The following construction has the required properties:

Let $\succ$ be a *reduction ordering that is total on ground terms.*

To a positive literal $s \approx t$, we assign the multiset $\{s, t\}$, to a negative literal $s \not\approx t$ the multiset $\{s, s, t, t\}$. The *literal ordering* $\succ_{\text{L}}$ compares these multisets using the multiset extension of $\succ$.

The *clause ordering* $\succ_{\text{C}}$ compares clauses by comparing their multisets of literals using the multiset extension of $\succ_{\text{L}}$.

Constructing a candidate interpretation:

We want to use roughly the same ideas as in the completeness proof for resolution.

However, a Herbrand interpretation does not work for equality: The equality symbol $\approx$ must be interpreted by equality in the interpretation.

Solution: Productive clauses contribute ground rewrite rules to a TRS R.

The interpretation has the universe $\mathrm{T}_\Sigma(\emptyset)/R = \mathrm{T}_\Sigma(\emptyset)/{\approx_R}$; a ground atom $s \approx t$ holds in the interpretation if and only if $s \approx_R t$ if and only if $s \leftrightarrow_R^* t$.

We will construct R in such a way that it is terminating and confluent. In this case, $s \approx_R t$ if and only if $s \downarrow_R t$.

One problem:

The completeness proof for the resolution calculus depends on the following property:

> If $C = C' \vee A$ with a strictly maximal and positive literal A is false in the current interpretation, then adding A to the current interpretation cannot make any literal of C' true.

This property does not hold for superposition:

> Let $b \succ c \succ d$. Assume that the current rewrite system (representing the current interpretation) contains the rule $c \to d$. Now consider the clause $b \approx d \vee b \approx c$.

We need a further inference rule to deal with clauses of this kind, either the "Merging Paramodulation" rule of Bachmair and Ganzinger or the following "Equality Factoring" rule due to Nieuwenhuis:

Equality Factoring:
$$\frac{C' \vee s \approx t' \vee s \approx t}{C' \vee t \not\approx t' \vee s \approx t'}$$

What do the nonground versions of the inference rules for superposition look like?

> Main idea as in the resolution calculus:
>
> Replace identity by unifiability. Apply the mgu to the resulting clause. In the ordering restrictions, use $\not\succeq$ instead of $\succ$.

However:

> As in Knuth–Bendix completion, we do not want to consider overlaps at or below a variable position.
>
> Consequence: There are inferences between ground instances $D\theta$ and $C\theta$ of clauses D and C which are *not* ground instances of inferences between D and C.
>
> Such inferences have to be treated in a special way in the completeness proof.

5.3 The Superposition Calculus—Formally

Until now, we have seen most of the ideas behind the superposition calculus and its completeness proof.

We will now start again from the beginning giving precise definitions and some proofs.

Inference rules:

Pos. Superposition:
$$\frac{D' \vee t \approx t' \qquad C' \vee s[u] \approx s'}{(D' \vee C' \vee s[t'] \approx s')\sigma}$$
where $\sigma = \mathrm{mgu}(t, u)$ and u is not a variable.

Neg. Superposition:
$$\frac{D' \vee t \approx t' \qquad C' \vee s[u] \not\approx s'}{(D' \vee C' \vee s[t'] \not\approx s')\sigma}$$
where $\sigma = \mathrm{mgu}(t, u)$ and u is not a variable.

Equality Resolution:
$$\frac{C' \vee s \not\approx s'}{C'\sigma}$$
where $\sigma = \mathrm{mgu}(s, s')$.

Equality Factoring:
$$\frac{C' \vee s' \approx t' \vee s \approx t}{(C' \vee t \not\approx t' \vee s \approx t')\sigma}$$
where $\sigma = \mathrm{mgu}(s, s')$.

Theorem 5.3.1 *All inference rules of the superposition calculus are* sound, *i.e., for every rule*

$$\frac{C_n \quad \cdots \quad C_1}{C_0}$$

we have $\{C_1, \ldots, C_n\} \models C_0$.

Proof. Omitted. □

Orderings:

Let $\succ$ be a *reduction ordering that is total on ground terms.*

To a positive literal $s \approx t$, we assign the multiset $\{s, t\}$, to a negative literal $s \not\approx t$ the multiset $\{s, s, t, t\}$. The *literal ordering* $\succ_{\mathrm{L}}$ compares these multisets using the multiset extension of $\succ$.

The *clause ordering* $\succ_{\mathrm{C}}$ compares clauses by comparing their multisets of literals using the multiset extension of $\succ_{\mathrm{L}}$.

Inferences have to be computed only if the following ordering restrictions are satisfied (after applying the unifier to the premises):

- In superposition inferences, the left premise is not greater than or equal to the right one.
- The last literal in each premise is maximal in the respective premise, i.e., there exists no greater literal (strictly maximal for positive literals in superposition inferences, i.e., there exists no greater or equal literal).
- In these literals, the lhs is neither smaller than nor equal to the rhs (except in equality resolution inferences).

A ground clause C is called *redundant w.r.t. a set of ground clauses* N if it follows from clauses in N that are smaller than C.

A clause is *redundant w.r.t. a set of clauses* N if all its ground instances are redundant w.r.t. $G_\Sigma(N)$.

The set of all clauses that are redundant w.r.t. N is denoted by $Red(N)$.

N is called *saturated up to redundancy* if the conclusion of every inference from clauses in $N \setminus Red(N)$ is contained in $N \cup Red(N)$.

The Superposition Calculus: Example

We consider the clause set (Bentkamp et al., *CACM* 2023)

$$\begin{aligned} x \approx zero \vee div(one, x) &\approx inv(x) \quad (1)\\ pi &\not\approx zero \quad (2)\\ abs(div(one, pi)) &\not\approx abs(inv(pi)) \quad (3) \end{aligned}$$

using an lpo with the precedence $abs > div > inv > pi > one > zero$.

From (1) and (3), we obtain via "Negative Superposition" $pi \approx zero \vee abs(inv(pi)) \not\approx abs(inv(pi))$ (4). From (4), we obtain via "Equality Resolution" $pi \approx zero$ (5). From (5) and (2), we obtain via "Negative Superposition" $zero \not\approx zero$ (6). From (6), we obtain via "Equality Resolution" the empty clause.

5.4 Superposition: Refutational Completeness

For a set E of ground equations, $\mathrm{T}_\Sigma(\emptyset)/E$ is an E-interpretation (or E-algebra) with universe $\{[t] \mid t \in \mathrm{T}_\Sigma(\emptyset)\}$.

One can show (similar to the proof of Birkhoff's Theorem) that for every *ground* equation $s \approx t$ we have $\mathrm{T}_\Sigma(\emptyset)/E \models s \approx t$ if and only if $s \leftrightarrow_E^* t$.

In particular, if E is a convergent set of rewrite rules R and $s \approx t$ is a ground equation, then $\mathrm{T}_\Sigma(\emptyset)/R \models s \approx t$ if and only if $s \downarrow_R t$. By abuse of terminology, we say that an equation or clause is valid (or true) in R if and only if it is true in $\mathrm{T}_\Sigma(\emptyset)/R$.

Construction of candidate interpretations (Bachmair and Ganzinger 1990):

Let N be a set of clauses not containing $\bot$. Using induction on the clause ordering we define sets of rewrite rules E_C and R_C for all $C \in G_\Sigma(N)$ as follows:

Assume that E_D has already been defined for all $D \in G_\Sigma(N)$ with $D \prec_\mathrm{C} C$. Then $R_C = \bigcup_{D \prec_\mathrm{C} C} E_D$.

The set E_C contains the rewrite rule $s \to t$ if

(a) $C = C' \vee s \approx t$.

(b) $s \approx t$ is strictly maximal in C.

(c) $s \succ t$.

(d) C is false in R_C.

(e) C' is false in $R_C \cup \{s \to t\}$.

(f) s is irreducible w.r.t. R_C.

In this case, C is called *productive*. Otherwise $E_C = \emptyset$.

Finally, $R_\infty = \bigcup_{D \in G_\Sigma(N)} E_D$.

Example: We use the lpo with the precedence $f \succ e \succ d \succ c \succ b \succ a$ (max. side of max. literals in red).

Let $N = \{d \approx c,\ b \approx a \vee e \not\approx c,\ b \not\approx b \vee f(b) \approx a,\ f(c) \approx b,\ f(b) \approx a \vee f(c) \not\approx b,\ f(b) \approx a \vee f(d) \not\approx b\}$ be a clause set saturated w.r.t. the ground superposition calculus.

The next table shows each iteration of the candidate interpretation construction for N.

Iter.	Clause C	R_C	E_C
0	$d \approx c$	$\emptyset$	$\{d \to c\}$
1	$b \approx a \vee e \not\approx c$	$\{d \to c\}$	$\emptyset$
2	$b \not\approx b \vee f(b) \approx a$	$\{d \to c\}$	$\{f(b) \to a\}$
3	$f(c) \approx b$	$\{d \to c, f(b) \to a\}$	$\{f(c) \to b\}$
4	$f(b) \approx a \vee f(c) \not\approx b$	$\{d \to c, f(b) \to a, f(c) \to b\}$	$\emptyset$
5	$f(b) \approx a \vee f(d) \not\approx b$	$\{d \to c, f(b) \to a, f(c) \to b\}$	$\emptyset$

At each iteration $i+1$, the term rewriting system consists of the union of the rewrite rules R_C and the "epsilon" E_C of iteration i. The interpretation $R_\infty = \{d \to c, f(b) \to a, f(c) \to b\}$ after iteration 5 is a model of N.

Lemma 5.4.1 *If $E_C = \{s \to t\}$ and $E_D = \{u \to v\}$, then $s \succ u$ if and only if $C \succ_{\mathrm{C}} D$.*

Proof. ($\Rightarrow$): By condition (b), $s \approx t$ is strictly maximal in C and $u \approx v$ is strictly maximal in D, and since the literal ordering is total on ground literals, this implies that all other literals in C or in D are actually smaller than $s \approx t$ or $u \approx v$, respectively.

Moreover, $s \succ t$ and $u \succ v$ by condition (c). Therefore $s \succ u$ implies $\{s, t\} \succ_{\mathrm{mul}} \{u, v\}$. Hence $s \approx t \succ_{\mathrm{L}} u \approx v \succeq_{\mathrm{L}} L$ for every literal L of D, and thus $C \succ_{\mathrm{C}} D$.

($\Leftarrow$): Let $C \succ_{\mathrm{C}} D$, then $E_D \subseteq R_C$. By condition (f), s must be irreducible w.r.t. R_C, so $s \neq u$.

Assume that $s \not\succ u$. By totality, this implies $s \preceq u$, and since $s \neq u$, we obtain $s \prec u$. But then $C \prec_{\mathrm{C}} D$ can be shown in the same way as in the ($\Rightarrow$)-part, contradicting the assumption. □

Corollary 5.4.2 *The rewrite systems R_C and R_∞ are convergent (i.e., terminating and confluent).*

Proof. By condition (c), $s \succ t$ for all rules $s \to t$ in R_C and R_∞, so R_C and R_∞ are terminating.

Furthermore, it is easy to check that there are no critical pairs between any two rules: Assume that there are rules $u \to v$ in E_D and $s \to t$ in E_C such that u is a subterm of s. As $\succ$ is a reduction ordering that is total on ground terms, we get $u \prec s$ and therefore $D \prec_{\mathrm{C}} C$ and $E_D \subseteq R_C$. But then s would be reducible by R_C, contradicting condition (f).

Now the absence of critical pairs implies local confluence, and termination and local confluence imply confluence. □

Lemma 5.4.3 *If $D \preceq_{\mathrm{C}} C$ and $E_C = \{s \to t\}$, then $s \succ u$ for every term u occurring in a negative literal in D and $s \succeq u$ for every term u occurring in a positive literal in D.*

Proof. If $s \preceq u$ for some term u occurring in a negative literal $u \not\approx v$ in D, then $\{u, u, v, v\} \succ_{\text{mul}} \{s, t\}$. So $u \not\approx v \succ_{\text{L}} s \approx t \succeq_{\text{L}} L$ for every literal L of C, and therefore $D \succ_{\text{C}} C$.

Similarly, if $s \prec u$ for some term u occurring in a positive literal $u \approx v$ in D, then $\{u, v\} \succ_{\text{mul}} \{s, t\}$. So $u \approx v \succ_{\text{L}} s \approx t \succeq_{\text{L}} L$ for every literal L of C, and therefore $D \succ_{\text{C}} C$. □

Corollary 5.4.4 *If $D \in G_\Sigma(N)$ is true in R_D, then D is true in R_∞ and R_C for all $C \succ_{\text{C}} D$.*

Proof. If a positive literal of D is true in R_D, then this is obvious.

Otherwise, some negative literal $s \not\approx t$ of D must be true in R_D, hence $s \not\downarrow_{R_D} t$. As the rules in $R_\infty \setminus R_D$ have left-hand sides that are larger than s and t, they cannot be used in a rewrite proof of $s \downarrow t$, hence $s \not\downarrow_{R_C} t$ and $s \not\downarrow_{R_\infty} t$. □

Corollary 5.4.5 *If $D = D' \vee u \approx v$ is productive, then D' is false and D is true in R_∞ and R_C for all $C \succ_{\text{C}} D$.*

Proof. Obviously, D is true in R_∞ and R_C for all $C \succ_{\text{C}} D$.

Since all negative literals of D' are false in R_D, it is clear that they are false in R_∞ and R_C. For the positive literals $u' \approx v'$ of D', condition (e) ensures that they are false in $R_D \cup \{u \to v\}$. Since $u' \preceq u$ and $v' \preceq u$ and all rules in $R_\infty \setminus R_D$ have left-hand sides that are larger than u, these rules cannot be used in a rewrite proof of $u' \downarrow v'$, hence $u' \not\downarrow_{R_C} v'$ and $u' \not\downarrow_{R_\infty} v'$. □

Lemma 5.4.6 ("Lifting Lemma") *Let C be a clause and let θ be a substitution such that $C\theta$ is ground. Then every equality resolution or equality factoring inference from $C\theta$ is a ground instance of an inference from C.*

Proof. Omitted. □

Lemma 5.4.7 ("Lifting Lemma") *Let $D = D' \vee u \approx v$ and $C = C' \vee [\neg]\, s \approx t$ be two clauses (without common variables) and let θ be a substitution such that $D\theta$ and $C\theta$ are ground.*

If there is a superposition inference between $D\theta$ and $C\theta$ where $u\theta$ and some subterm of $s\theta$ are overlapped, and $u\theta$ does not occur in $s\theta$ at or below a variable position of s, then the inference is a ground instance of a superposition inference from D and C.

Proof. Omitted. □

Theorem 5.4.8 ("Model Construction") *Let N be a set of clauses that is saturated up to redundancy and does not contain the empty clause. Then we have for every ground clause $C\theta \in G_\Sigma(N)$:*

(i) $E_{C\theta} = \emptyset$ if and only if $C\theta$ is true in $R_{C\theta}$.

(ii) If $C\theta$ is redundant w.r.t. $G_\Sigma(N)$, then it is true in $R_{C\theta}$.

(iii) $C\theta$ is true in R_∞ and in R_D for every $D \in G_\Sigma(N)$ with $D \succ_{\mathrm{C}} C\theta$.

Proof. We use induction on the clause ordering $\succ_{\mathrm{C}}$ and assume that (i)–(iii) are already satisfied for all clauses in $G_\Sigma(N)$ that are smaller than $C\theta$. Note that the "if" part of (i) is obvious from the construction and that condition (iii) follows immediately from (i) and Corollaries 5.4.4 and 5.4.5. So it remains to show (ii) and the "only if" part of (i).

Case 1: $C\theta$ is redundant w.r.t. $G_\Sigma(N)$.

If $C\theta$ is redundant w.r.t. $G_\Sigma(N)$, then it follows from clauses in $G_\Sigma(N)$ that are smaller than $C\theta$. By part (iii) of the induction hypothesis, these clauses are true in $R_{C\theta}$. Hence $C\theta$ is true in $R_{C\theta}$.

Case 2: $x\theta$ is reducible by $R_{C\theta}$.

Suppose there is a variable x occurring in C such that $x\theta$ is reducible by $R_{C\theta}$, say $x\theta \to_{R_{C\theta}} w$. Let the substitution θ' be defined by $x\theta' = w$ and $y\theta' = y\theta$ for every variable $y \neq x$. The clause $C\theta'$ is smaller than $C\theta$. By part (iii) of the induction hypothesis, it is true in $R_{C\theta}$. By congruence, every literal of $C\theta$ is true in $R_{C\theta}$ if and only if the corresponding literal of $C\theta'$ is true in $R_{C\theta}$; hence $C\theta$ is true in $R_{C\theta}$.

Case 3: $C\theta$ contains a maximal negative literal.

Suppose that $C\theta$ does not fall into Case 1 or 2 and that $C\theta = C'\theta \vee s\theta \not\approx s'\theta$, where $s\theta \not\approx s'\theta$ is maximal in $C\theta$. If $s\theta \approx s'\theta$ is false in $R_{C\theta}$, then $C\theta$ is clearly true in $R_{C\theta}$ and we are done. So assume that $s\theta \approx s'\theta$ is true in $R_{C\theta}$, that is, $s\theta \downarrow_{R_{C\theta}} s'\theta$. Without loss of generality, $s\theta \succeq s'\theta$.

Case 3.1: $s\theta = s'\theta$.

If $s\theta = s'\theta$, then there is an "Equality Resolution" inference

$$\frac{C'\theta \vee s\theta \not\approx s'\theta}{C'\theta}\ .$$

As shown in the Lifting Lemma, this is an instance of an "Equality Resolution" inference

$$\frac{C' \vee s \not\approx s'}{C'\sigma}$$

where $C = C' \vee s \not\approx s'$ is contained in N and $\theta = \rho \circ \sigma$. (Without loss of generality, σ is idempotent, therefore $C'\theta = C'\sigma\rho = C'\sigma\sigma\rho = C'\sigma\theta$, so $C'\theta$ is a ground instance of $C'\sigma$.) Since $C\theta$ is not redundant w.r.t. $G_\Sigma(N)$, C is not redundant w.r.t. N. As N is saturated up to redundancy, the conclusion $C'\sigma$ of the inference from C is contained in $N \cup Red(N)$. Therefore, $C'\theta$ is either contained in $G_\Sigma(N)$ and smaller than $C\theta$, or it follows from clauses in $G_\Sigma(N)$ that are smaller than itself (and therefore smaller than $C\theta$). By the induction hypothesis, clauses in $G_\Sigma(N)$ that are smaller than $C\theta$ are true in $R_{C\theta}$, thus $C'\theta$ and $C\theta$ are true in $R_{C\theta}$.

Case 3.2: $s\theta \succ s'\theta$.

If $s\theta \downarrow_{R_{C\theta}} s'\theta$ and $s\theta \succ s'\theta$, then $s\theta$ must be reducible by some rule in some $E_{D\theta} \subseteq R_{C\theta}$. (Without loss of generality we assume that C and D are variable disjoint; so we can use the same substitution θ.) Let $D\theta = D'\theta \vee t\theta \approx t'\theta$ with $E_{D\theta} = \{t\theta \to t'\theta\}$. Since $D\theta$ is productive, $D'\theta$ is false in $R_{C\theta}$. Besides, by part (ii) of the induction hypothesis, $D\theta$ is not redundant w.r.t. $G_\Sigma(N)$, so D is not redundant w.r.t. N. Note that $t\theta$ cannot occur in $s\theta$ at or below a variable position of s, say $x\theta = w[t\theta]$, since otherwise $C\theta$ would be subject to Case 2 above. Consequently, the "Negative Superposition" inference

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

is a ground instance of a "Negative Superposition" inference from D and C. By saturation up to redundancy, its conclusion is either contained in $G_\Sigma(N)$ and smaller than $C\theta$, or it follows from clauses in $G_\Sigma(N)$ that are smaller than itself (and therefore smaller than $C\theta$). By the induction hypothesis, these clauses are true in $R_{C\theta}$, thus $D'\theta \vee C'\theta \vee s\theta[t'\theta] \not\approx s'\theta$ is true in $R_{C\theta}$. Since $D'\theta$ and $s\theta[t'\theta] \not\approx s'\theta$ are false in $R_{C\theta}$, both $C'\theta$ and $C\theta$ must be true.

Case 4: $C\theta$ does not contain a maximal negative literal.

Suppose that $C\theta$ does not fall into Cases 1 to 3. Then $C\theta$ can be written as $C'\theta \vee s\theta \approx s'\theta$, where $s\theta \approx s'\theta$ is a maximal literal of $C\theta$. If $E_{C\theta} = \{s\theta \to s'\theta\}$ or $C'\theta$ is true in $R_{C\theta}$ or $s\theta = s'\theta$, then there is nothing to show, so assume that $E_{C\theta} = \emptyset$ and that $C'\theta$ is false in $R_{C\theta}$. Without loss of generality, $s\theta \succ s'\theta$.

Case 4.1: $s\theta \approx s'\theta$ is maximal in $C\theta$, but not strictly maximal.

If $s\theta \approx s'\theta$ is maximal in $C\theta$, but not strictly maximal, then $C\theta$ can be written as $C''\theta \vee t\theta \approx t'\theta \vee s\theta \approx s'\theta$, where $t\theta = s\theta$ and $t'\theta = s'\theta$. In this case, there is a

"Equality Factoring" inference

$$\frac{C'' \theta \vee t\theta \approx t'\theta \vee s\theta \approx s'\theta}{C'' \theta \vee t'\theta \not\approx s'\theta \vee t\theta \approx t'\theta}$$

This inference is a ground instance of an inference from C. By saturation, its conclusion is true in $R_{C\theta}$. Trivially, $t'\theta = s'\theta$ implies $t'\theta \downarrow_{R_{C\theta}} s'\theta$, so $t'\theta \not\approx s'\theta$ must be false and $C\theta$ must be true in $R_{C\theta}$.

Case 4.2: $s\theta \approx s'\theta$ is strictly maximal in $C\theta$ and $s\theta$ is reducible.

Suppose that $s\theta \approx s'\theta$ is strictly maximal in $C\theta$ and $s\theta$ is reducible by some rule in $E_{D\theta} \subseteq R_{C\theta}$. Let $D\theta = D'\theta \vee t\theta \approx t'\theta$ and $E_{D\theta} = \{t\theta \to t'\theta\}$. Since $D\theta$ is productive, $D\theta$ is not redundant and $D'\theta$ is false in $R_{C\theta}$. We can now proceed in essentially the same way as in Case 3.2: If $t\theta$ occurred in $s\theta$ at or below a variable position of s, say $x\theta = w[t\theta]$, then $C\theta$ would be subject to Case 2 above. Otherwise, the "Positive Superposition" inference

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

is a ground instance of a "Positive Superposition" inference from D and C. By saturation up to redundancy, its conclusion is true in $R_{C\theta}$. Since $D'\theta$ and $C'\theta$ are false in $R_{C\theta}$, $s\theta[t'\theta] \approx s'\theta$ must be true in $R_{C\theta}$. On the other hand, $t\theta \approx t'\theta$ is true in $R_{C\theta}$, so by congruence, $s\theta[t\theta] \approx s'\theta$ and $C\theta$ are true in $R_{C\theta}$.

Case 4.3: $s\theta \approx s'\theta$ is strictly maximal in $C\theta$ and $s\theta$ is irreducible.

Suppose that $s\theta \approx s'\theta$ is strictly maximal in $C\theta$ and $s\theta$ is irreducible by $R_{C\theta}$. Then there are three possibilities: $C\theta$ can be true in $R_{C\theta}$, or $C'\theta$ can be true in $R_{C\theta} \cup \{s\theta \to s'\theta\}$, or $E_{C\theta} = \{s\theta \to s'\theta\}$. In the first and the third case, there is nothing to show. Let us therefore assume that $C\theta$ is false in $R_{C\theta}$ and $C'\theta$ is true in $R_{C\theta} \cup \{s\theta \to s'\theta\}$. Then $C'\theta = C''\theta \vee t\theta \approx t'\theta$, where the literal $t\theta \approx t'\theta$ is true in $R_{C\theta} \cup \{s\theta \to s'\theta\}$ and false in $R_{C\theta}$. In other words, $t\theta \downarrow_{R_{C\theta} \cup \{s\theta \to s'\theta\}} t'\theta$, but not $t\theta \downarrow_{R_{C\theta}} t'\theta$. Consequently, there is a rewrite proof of $t\theta \to^* u \leftarrow^* t'\theta$ by $R_{C\theta} \cup \{s\theta \to s'\theta\}$ in which the rule $s\theta \to s'\theta$ is used at least once. Without loss of generality we assume that $t\theta \succeq t'\theta$. Since $s\theta \approx s'\theta \succ_{\mathrm{L}} t\theta \approx t'\theta$ and $s\theta \succ s'\theta$ we can conclude that $s\theta \succeq t\theta \succ t'\theta$. But then there is only one possibility how the rule $s\theta \to s'\theta$ can be used in the rewrite proof: We must have $s\theta = t\theta$ and the rewrite proof must have the form $t\theta \to s'\theta \to^* u \leftarrow^* t'\theta$, where the first step uses $s\theta \to s'\theta$ and all other steps use rules from $R_{C\theta}$. Consequently, $s'\theta \approx t'\theta$ is true in $R_{C\theta}$. Now observe that there is an "Equality Factoring" inference

$$\frac{C'' \theta \vee t\theta \approx t'\theta \vee s\theta \approx s'\theta}{C'' \theta \vee t'\theta \not\approx s'\theta \vee t\theta \approx t'\theta}$$

whose conclusion is true in $R_{C\theta}$ by saturation. Since the literal $t'\theta \not\approx s'\theta$ must be false in $R_{C\theta}$, the rest of the clause must be true in $R_{C\theta}$, and therefore $C\theta$ must be true in $R_{C\theta}$, contradicting our assumption. This concludes the proof of the theorem. □

A Σ-interpretation $\mathcal{A}$ is called *term-generated* if for every $b \in U_\mathcal{A}$ there is a ground term $t \in \mathrm{T}_\Sigma(\emptyset)$ such that $b = \mathcal{A}(\beta)(t)$.

Lemma 5.4.9 *Let N be a set of (universally quantified) Σ-clauses and let $\mathcal{A}$ be a term-generated Σ-interpretation. Then $\mathcal{A}$ is a model of $G_\Sigma(N)$ if and only if it is a model of N.*

Proof. ($\Rightarrow$): Let $\mathcal{A} \models G_\Sigma(N)$; let $(\forall \vec{x}\, C) \in N$. Then $\mathcal{A} \models \forall \vec{x}\, C$ iff $\mathcal{A}(\gamma[x_i \mapsto a_i])(C) = 1$ for all γ and a_i. Choose ground terms t_i such that $\mathcal{A}(\gamma)(t_i) = a_i$; define θ such that $x_i\theta = t_i$, then $\mathcal{A}(\gamma[x_i \mapsto a_i])(C) = \mathcal{A}(\gamma \circ \theta)(C) = \mathcal{A}(\gamma)(C\theta) = 1$ since $C\theta \in G_\Sigma(N)$.

($\Leftarrow$): Let $\mathcal{A}$ be a model of N; let $\forall \vec{x}\, C \in N$ and $C\theta \in G_\Sigma(N)$. Then $\mathcal{A} \models \forall \vec{x}\, C$ and therefore $\mathcal{A} \models C$. Consequently $\mathcal{A}(\gamma)(C\theta) = \mathcal{A}(\gamma \circ \theta)(C) = 1$. □

Theorem 5.4.10 (Refutational Completeness: Static View) *Let N be a set of clauses that is saturated up to redundancy. Then N has a model if and only if N does not contain the empty clause.*

Proof. If $\bot \in N$, then obviously N does not have a model. If $\bot \notin N$, then the interpretation R_∞ (that is, $\mathrm{T}_\Sigma(\emptyset)/R_\infty$) is a model of all ground instances in $G_\Sigma(N)$ according to part (iii) of the model construction theorem. As $\mathrm{T}_\Sigma(\emptyset)/R_\infty$ is term-generated, it is a model of N. □

So far, we have considered only inference rules that add new clauses to the current set of clauses (corresponding to the "Deduce" rule of Knuth–Bendix completion).

In other words, we have derivations of the form $N_0 \vdash N_1 \vdash N_2 \vdash \cdots$, where each N_{i+1} is obtained from N_i by adding the consequence of some inference from clauses in N_i.

Under which circumstances are we allowed to delete (or simplify) a clause during the derivation?

A *run* of the superposition calculus is a sequence $N_0 \vdash N_1 \vdash N_2 \vdash \cdots$ such that
(i) $N_i \models N_{i+1}$, and
(ii) all clauses in $N_i \setminus N_{i+1}$ are redundant w.r.t. N_{i+1}.

In other words, during a run we may add a new clause if it follows from the old ones, and we may delete a clause if it is redundant w.r.t. the remaining ones.

For a run, $N_\infty = \bigcup_{i \geq 0} \bigcap_{j \geq i} N_j$. The set N_∞ of all *persistent* clauses is called the *limit* of the run.

Lemma 5.4.11 *If $N \subseteq N'$, then $Red(N) \subseteq Red(N')$.*

Proof. Obvious. □

Lemma 5.4.12 *If $N' \subseteq Red(N)$, then $Red(N) \subseteq Red(N \setminus N')$.*

Proof. Omitted. □

Lemma 5.4.13 *Let $N_0 \vdash N_1 \vdash N_2 \vdash \cdots$ be a run. Then $Red(N_i) \subseteq Red(\bigcup_{j \geq 0} N_j)$ and $Red(N_i) \subseteq Red(N_\infty)$ for every i.*

Proof. Omitted. □

Corollary 5.4.14 *$N_i \subseteq N_\infty \cup Red(N_\infty)$ for every i.*

Proof. If $C \in N_i \setminus N_\infty$, then there is a $k \geq i$ such that $C \in N_k \setminus N_{k+1}$. Therefore C must be redundant w.r.t. N_{k+1}. Consequently, C is redundant w.r.t. N_∞. □

A run is called *fair* if the conclusion of every inference from clauses in $N_\infty \setminus Red(N_\infty)$ is contained in some $N_i \cup Red(N_i)$.

Lemma 5.4.15 *If a run is fair, then its limit is saturated up to redundancy.*

Proof. If the run is fair, then the conclusion of every inference from nonredundant clauses in N_∞ is contained in some $N_i \cup Red(N_i)$, and therefore contained in $N_\infty \cup Red(N_\infty)$. Hence N_∞ is saturated up to redundancy. □

Theorem 5.4.16 (Refutational Completeness: Dynamic View) *Let $N_0 \vdash N_1 \vdash N_2 \vdash \cdots$ be a fair run, let N_∞ be its limit. Then N_0 has a model if and only if $\bot \notin N_\infty$.*

Proof. ($\Leftarrow$): By fairness, N_∞ is saturated up to redundancy. If $\bot \notin N_\infty$, then it has a term-generated model. Since every clause in N_0 is contained in N_∞ or redundant w.r.t. N_∞, this model is also a model of $G_\Sigma(N_0)$ and therefore a model of N_0.

($\Rightarrow$): Obvious, since $N_0 \models N_\infty$. □

5.5 Improvements and Refinements

The superposition calculus as described so far can be improved and refined in several ways.

Concrete Redundancy and Simplification Criteria

Redundancy is undecidable.

Even decidable approximations are often expensive (experimental evaluations are needed to see what pays off in practice).

Often a clause can be *made* redundant by adding another clause that is entailed by the existing ones.

This process is called *simplification*.

Examples:

Subsumption:
If N contains clauses D and $C = C' \vee D\sigma$, where C' is nonempty, then D subsumes C and C is redundant.
Example: $f(x) \approx g(x)$ subsumes $f(y) \approx a \vee f(h(y)) \approx g(h(y))$.

Trivial literal elimination:
Duplicated literals and trivially false literals can be deleted: A clause $C' \vee L \vee L$ can be simplified to $C' \vee L$; a clause $C' \vee s \not\approx s$ can be simplified to C'.

Condensation:
If we obtain a clause D from C by applying a substitution, followed by deletion of duplicated literals, and if D subsumes C, then C can be simplified to D.
Example: By applying $\{y \to g(x)\}$ to $C = f(g(x)) \approx a \vee f(y) \approx a$ and deleting the duplicated literal, we obtain $f(g(x)) \approx a$, which subsumes C.

Semantic tautology deletion:
Every clause that is a tautology is redundant. Note that in the nonequational case, a clause is a tautology if and only if it contains two complementary literals, whereas in the equational case we need a congruence closure algorithm to detect that a clause like $x \not\approx y \vee f(x) \approx f(y)$ is tautological.

Rewriting:
If N contains a unit clause $D = s \approx t$ and a clause $C[s\sigma]$, such that $s\sigma \succ t\sigma$ and $C \succ_{\mathrm{C}} D\sigma$, then C can be simplified to $C[t\sigma]$.
Example: If $D = f(x, x) \approx g(x)$ and $C = h(f(g(y), g(y))) \approx h(y)$, and $\succ$ is an lpo with the precedence $h \succ f \succ g$, then C can be simplified to $h(g(g(y))) \approx h(y)$.

Selection Functions

Like the ordered resolution calculus, superposition can be used with a selection function that overrides the ordering restrictions for negative literals.

A *selection function* is a mapping

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

We indicate selected literals by a box:

$$\boxed{\neg f(x) \approx a} \vee\ g(x, y) \approx g(x, z)$$

The second ordering condition for inferences is replaced by

- Either the last literal in each premise is selected or there is no selected literal in the premise and the literal is maximal in the premise (strictly maximal for positive literals in superposition inferences).

In particular, clauses with selected literals can only be used in equality resolution inferences and as the second premise in negative superposition inferences.

Refutational completeness is proved essentially as before:

> We assume that each ground clause in $G_\Sigma(N)$ inherits the selection of one of the clauses in N of which it is a ground instance (there may be several ones).
>
> In the proof of the model construction theorem, we replace case 3 by "$C\theta$ contains a selected or maximal negative literal" and case 4 by "$C\theta$ contains neither a selected nor a maximal negative literal."
>
> In addition, for the induction proof of this theorem we need one more property, namely: (iv) If $C\theta$ has selected literals then $E_{C\theta} = \emptyset$.

Redundant Inferences

So far, we have defined saturation in terms of redundant clauses:

> N is *saturated up to redundancy* if the conclusion of every inference from clauses in $N \setminus \mathit{Red}(N)$ is contained in $N \cup \mathit{Red}(N)$.

This definition ensures that in the proof of the model construction theorem, the conclusion $C_0\theta$ of a ground inference follows from clauses in $G_\Sigma(N)$ that are smaller than or equal to itself, hence they are smaller than the premise $C\theta$ of the inference, hence they are true in $R_{C\theta}$ by induction.

However, a closer inspection of the proof shows that it is actually sufficient that the clauses from which $C_0\theta$ follows are smaller than $C\theta$—it is *not* necessary that they are smaller than $C_0\theta$ itself. This motivates the following definition of redundant *inferences*:

A ground inference with conclusion C_0 and right (or only) premise C is called *redundant w.r.t. a set of ground clauses* N if one of its premises is redundant w.r.t. N, or if C_0 follows from clauses in N that are smaller than C.

An inference is *redundant w.r.t. a set of clauses* N if all its ground instances are redundant w.r.t. $G_\Sigma(N)$.

Recall that a clause can be redundant w.r.t. N without being contained in N. Analogously, an inference can be redundant w.r.t. N without being an inference from clauses in N.

The set of all inferences that are redundant w.r.t. N is denoted by $\mathit{RedInf}(N)$.

Saturation is then redefined in the following way:

N is *saturated up to redundancy* if every inference from clauses in N is redundant w.r.t. N.

Using this definition, the model construction theorem can be proved essentially as before.

The connection between redundant inferences and clauses is given by the following lemmas. They are proved in the same way as the corresponding lemmas for redundant clauses:

Lemma 5.5.1 *If $N \subseteq N'$, then $\mathit{RedInf}(N) \subseteq \mathit{RedInf}(N')$.*

Lemma 5.5.2 *If $N' \subseteq \mathit{Red}(N)$, then $\mathit{RedInf}(N) \subseteq \mathit{RedInf}(N \setminus N')$.*

Literature

Leo Bachmair, Harald Ganzinger: Completion of First-Order Clauses with Equality by Strict Superposition (Extended Abstract). Conditional and Typed Rewriting Systems, 2nd International Workshop, LNCS 516, pp. 162–180, Springer, 1990.

Leo Bachmair, Harald Ganzinger: Rewrite-based Equational Theorem Proving with Selection and Simplification. Journal of Logic and Computation, 4(3):217–247, 1994.

Leo Bachmair, Harald Ganzinger: Resolution Theorem Proving. Handbook of Automated Reasoning, Vol. 1, Ch. 2, pp. 19–99, Elsevier Science B.V., 2001.

Christoph Weidenbach: Combining Superposition, Sorts and Splitting. Handbook of Automated Reasoning, Vol. 2, Ch. 27, pp. 1965–2013, Elsevier Science B.V., 2001.

6 Efficient 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.

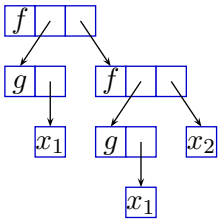
Design decisions depend on each other.

Design decisions depend on the particular class of problems we want to solve (first-order logic without or with equality/unit equations, size of the signature, special algebraic properties like associativity and commutativity, etc.).

6.1 Term Representations

The obvious data structure for terms: Trees

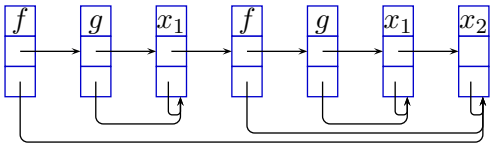
$f(g(x_1), f(g(x_1), x_2))$



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.

6.2 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.

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:

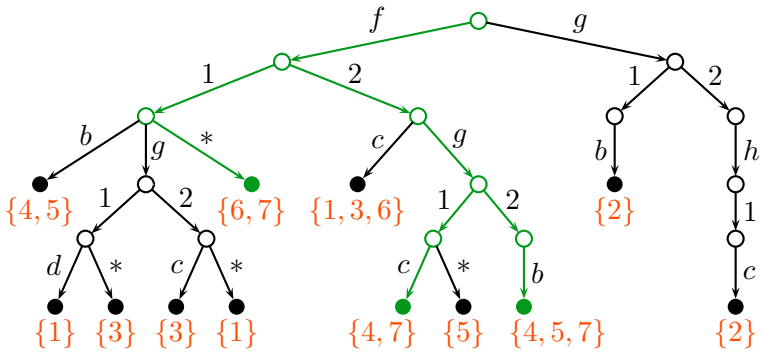
Paths of terms are encoded in a trie ("retrieval tree").

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 $\{f(g(d,*),c),\ g(b,h(c)),\ f(g(*,c),c),\ f(b,g(c,b)),\ f(b,g(*,b)),\ f(*,c),\ f(*,g(c,b))\}$



Advantages:

Uses little space.
No backtracking for retrieval.
Efficient insertion and deletion.
Good for finding instances, also usable for finding generalizations.

Disadvantages:

Retrieval requires combining intermediate results for all paths.

Discrimination Trees

Discrimination trees:

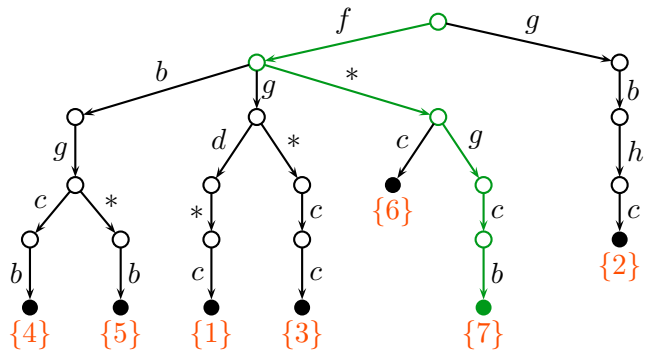
Preorder traversals of terms are encoded in a trie.

A star $*$ represents arbitrary variables.

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

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

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



Advantages:

Each leaf yields one term, hence retrieval does not require intersections of intermediate results for all paths.

Good for finding generalizations, not so good for finding instances.

Disadvantage:

Uses more storage than path indexing (due to less sharing).

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

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.

Stephan Schulz: Simple and Efficient Clause Subsumption with Feature Vector Indexing, in Bonacina and Stickel (eds.), *Automated Reasoning and Mathematics*, LNCS 7788, Springer, 2013.

Christoph Weidenbach: Combining Superposition, Sorts and Splitting, Ch. 27 in Robinson and Voronkov (eds.), *Handbook of Automated Reasoning, Vol. II*, Elsevier, 2001.

7 Outlook

7.1 Satisfiability Modulo Theories (SMT)

DPLL and CDCL check satisfiability of propositional formulas.

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

Ground first-order atoms are treated like propositional variables.

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

For ground formulas with equality, independence does not hold:

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 CDCL, but we must combine it with a decision procedure for the theory part T:

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

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 $\mathrm{var}(C_1) \cap \mathrm{var}(C_2) = \emptyset$.

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

The splitting tree is explored using intelligent backtracking.

Improvement:

Use a SAT solver to manage the selection of split clauses.

$\Rightarrow$ AVATAR.

7.4 Higher-Order Logics

What changes if we switch to higher-order logics?

Applied variables: $x\ b$.

Partially applied functions: $\mathit{times}\ z$.

Lambda-expressions with $\alpha\beta\eta$-conversion: $(\lambda x.\ f\ (x\ b)\ c)\ (\lambda y.\ d) = f\ d\ c$.

Embedded booleans: $(\lambda x.\,\text{if } x \text{ then } f \text{ else } g)\ (p \vee q)$

Problems:

Orderings cannot have all desired compatibility properties.
$\Rightarrow$ additional inferences

Most general unifiers need not exist anymore.
$\Rightarrow$ interleave enumeration of unifiers and computation of inferences

CNF transformation by preprocessing is no longer sufficient.
$\Rightarrow$ need calculus with embedded clausification