Chapter 3

First-Order Logic

First-Order logic is a generalization of propositional logic. Propositional logic can represent propositions, whereas first-order logic can represent individuals and propositions about individuals. For example, in propositional logic from "Socrates is a man" and "If Socrates is a man then Socrates is mortal" the conclusion "Socrates is mortal" can be drawn. In first-order logic this can be represented much more fine-grained. From "Socrates is a man" and "All man are mortal" the conclusion "Socrates is mortal" can be drawn.

This chapter introduces first-order logic with equality. However, all calculi presented here, namely Tableau and Free-Variable Tableau (Sections 3.6, 3.8), Resolution (Section 3.10), and Superposition (Section 3.12) are presented only for its restriction without equality. Purely equational logic and first-order logic with equality are presented separately in Chapter 4 and Chapter 5, respectively.

3.1 Syntax

Most textbooks introduce first-order logic in an unsorted way. Like in programming languages, sorts support distinguishing "apples from oranges" and therefore move part of the reasoning to a more complex syntax of formulas. Many-sorted logic is a generalization of unsorted first-order logic where the universe is separated into disjoint sets of objects, called *sorts*. Functions and predicates are defined with respect to these sorts in a unique way. The resulting language: many-sorted first-order logic has a very simple, but already useful sort structure, sometimes also called *type* structure. It can distinguish apples from oranges by providing two different, respective sorts, but it cannot express relationships between sorts. For example, it cannot express the integers to be a subsort of the reals, because all sorts are assumed to be disjoint. On the other hand, the simple many-sorted language comes at no extra cost when considering inference or simplification rules, whereas more expressive sort languages need extra and sometimes costly reasoning.

Definition 3.1.1 (Many-Sorted Signature)**.** A *many-sorted signature* $\Sigma = (\mathcal{S}, \Omega, \Pi)$ is a triple consisting of a finite non-empty set $\mathcal{S}$ of *sort symbols*, a non-empty set Ω of *operator symbols* (also called *function symbols*) over $\mathcal{S}$ and a set Π of *predicate symbols*. Every operator symbol $f \in \Omega$ has a unique sort declaration $f : S_1 \times \ldots \times S_n \to S$, indicating the sorts of arguments (also called *domain sorts*) and the *range sort* of f, respectively, for some $S_1, \ldots, S_n, S \in \mathcal{S}$ where $n \geq 0$ is called the *arity* of f, also denoted with $\text{arity}(f)$. An operator symbol $f \in \Omega$ with arity 0 is called a *constant*. Every predicate symbol $P \in \Pi$ has a unique sort declaration $P \subseteq S_1 \times \ldots \times S_n$. A predicate symbol $P \in \Pi$ with arity 0 is called a *propositional variable*. For every sort $S \in \mathcal{S}$ there must be at least one constant $a \in \Omega$ with range sort S.

In addition to the signature Σ, a variable set $\mathcal{X}$, disjoint from Ω is assumed, so that for every sort $S \in \mathcal{S}$ there exists a countably infinite subset of $\mathcal{X}$ consisting of variables of the sort S. A variable x of sort S is denoted by x_S.

Definition 3.1.2 (Term)**.** Given a signature $\Sigma = (\mathcal{S}, \Omega, \Pi)$, a sort $S \in \mathcal{S}$ and a variable set $\mathcal{X}$, the set $T_S(\Sigma, \mathcal{X})$ of all *terms* of sort S is recursively defined by (i) $x_S \in T_S(\Sigma, \mathcal{X})$ if $x_S \in \mathcal{X}$, (ii) $f(t_1, \ldots, t_n) \in T_S(\Sigma, \mathcal{X})$ if $f \in \Omega$ and $f : S_1 \times \ldots \times S_n \to S$ and $t_i \in T_{S_i}(\Sigma, \mathcal{X})$ for every $i \in \{1, \ldots, n\}$.

The sort of a term t is denoted by $\text{sort}(t)$, i.e., if $t \in T_S(\Sigma, \mathcal{X})$ then $\text{sort}(t) = S$. A term not containing a variable is called *ground*.

For the sake of simplicity it is often written: $T(\Sigma, \mathcal{X})$ for $\bigcup_{S \in \mathcal{S}} T_S(\Sigma, \mathcal{X})$, the set of all terms, $T_S(\Sigma)$ for the set of all ground terms of sort $S \in \mathcal{S}$, and $T(\Sigma)$ for $\bigcup_{S \in \mathcal{S}} T_S(\Sigma)$, the set of all ground terms over Σ.

A term t is called *shallow* if t is of the form $f(x_1, \ldots, x_n)$. A term t is called *linear* if every variable occurs at most once in t.

Note that the sets $T_S(\Sigma)$ are all non-empty, because there is at least one constant for each sort S in Σ. The sets $T_S(\Sigma, \mathcal{X})$ include infinitely many variables of sort S.

Definition 3.1.3 (Equation, Atom, Literal)**.** If $s, t \in T_S(\Sigma, \mathcal{X})$ then $s \approx t$ is an *equation* over the signature Σ. Any equation is an *atom* (also called *atomic formula*) as well as every $P(t_1, \ldots, t_n)$ where $t_i \in T_{S_i}(\Sigma, \mathcal{X})$ for every $i \in \{1, \ldots, n\}$ and $P \in \Pi$, $\text{arity}(P) = n$, $P \subseteq S_1 \times \ldots \times S_n$. An atom or its negation of an atom is called a *literal*.

The literal $s \mathrel{\dot{\approx}} t$ denotes either $s \approx t$ or $t \approx s$. A literal is *positive* if it is an atom and *negative* otherwise. A negative equational literal $\neg(s \approx t)$ is written as $s \not\approx t$.

C

Non equational atoms can be transformed into equations: For this a given signature is extended for every predicate symbol P as follows: (i) add a distinct sort Bool to $\mathcal{S}$, (ii) introduce a fresh constant true of the sort Bool to Ω, (iii) for every predicate P, $P \subseteq S_1 \times \ldots \times S_n$ add a fresh function $f_P : S_1, \ldots, S_n \to \text{Bool}$ to Ω, and (iv) encode every atom $P(t_1, \ldots, t_n)$ as an equation $f_P(t_1, \ldots, t_n) \approx \text{true}$, see Section 3.4. Definition 3.1.3 implicitly

overloads the equality symbol for all sorts S. An alternative would be to have a separate equality symbol for each sort.

Definition 3.1.4 (Formulas)**.** The set $\mathrm{FOL}(\Sigma, \mathcal{X})$ of *many-sorted first-order formulas with equality* over the signature Σ is defined as follows for formulas $\phi, \psi \in F_\Sigma(\mathcal{X})$ and a variable $x \in \mathcal{X}$:

$\mathrm{FOL}(\Sigma, \mathcal{X})$	Comment
$\bot$	false
$\top$	true
$P(t_1, \ldots, t_n), s \approx t$	atom
$(\neg\phi)$	negation
$(\phi \wedge \psi)$	conjunction
$(\phi \vee \psi)$	disjunction
$(\phi \to \psi)$	implication
$(\phi \leftrightarrow \psi)$	equivalence
$\forall x.\phi$	universal quantification
$\exists x.\phi$	existential quantification

A consequence of the above definition is that $\mathrm{PROP}(\Sigma) \subseteq \mathrm{FOL}(\Sigma', \mathcal{X})$ if the propositional variables of Σ are contained in Σ' as predicates of arity 0. A formula not containing a quantifier is called *quantifier-free*.

Definition 3.1.5 (Positions)**.** It follows from the definitions of terms and formulas that they have a tree-like structure. For referring to a certain subtree, called subterm or subformula, respectively, sequences of natural numbers are used, called *positions* (as introduced in Chapter 2.1.3). The set of positions of a term, formula is inductively defined by:

$$\begin{aligned}
\mathrm{pos}(x) &:= \{\epsilon\} \text{ if } x \in \mathcal{X} \\
\mathrm{pos}(\phi) &:= \{\epsilon\} \text{ if } \phi \in \{\top, \bot\} \\
\mathrm{pos}(\neg\phi) &:= \{\epsilon\} \cup \{1p \mid p \in \mathrm{pos}(\phi)\} \\
\mathrm{pos}(\phi \circ \psi) &:= \{\epsilon\} \cup \{1p \mid p \in \mathrm{pos}(\phi)\} \cup \{2p \mid p \in \mathrm{pos}(\psi)\} \\
\mathrm{pos}(s \approx t) &:= \{\epsilon\} \cup \{1p \mid p \in \mathrm{pos}(s)\} \cup \{2p \mid p \in \mathrm{pos}(t)\} \\
\mathrm{pos}(f(t_1, \ldots, t_n)) &:= \{\epsilon\} \cup \textstyle\bigcup_{i=1}^{n} \{ip \mid p \in pos(t_i)\} \\
\mathrm{pos}(P(t_1, \ldots, t_n)) &:= \{\epsilon\} \cup \textstyle\bigcup_{i=1}^{n} \{ip \mid p \in pos(t_i)\} \\
\mathrm{pos}(\forall x.\phi) &:= \{\epsilon\} \cup \{1p \mid p \in \mathrm{pos}(\phi)\} \\
\mathrm{pos}(\exists x.\phi) &:= \{\epsilon\} \cup \{1p \mid p \in \mathrm{pos}(\phi)\}
\end{aligned}$$

where $\circ \in \{\wedge, \vee, \to, \leftrightarrow\}$ and $t_i \in T(\Sigma, \mathcal{X})$ for all $i \in \{1, \ldots, n\}$.

The *prefix orders* (above, strictly above and parallel), the selection and replacement with respect to positions are defined exactly as in Chapter 2.1.3.

An term t (formula ϕ) is said to *contain* another term s (formula ψ) if $t|_p = s$ ($\phi|_p = \psi$). It is called a *strict subexpression* if $p \neq \epsilon$. The term t (formula ϕ) is called an *immediate subexpression* of s (formula ψ) if $|p| = 1$. For terms a subexpression is called a *subterm* and for formulas a *subformula*, respectively.

The *size* of a term t (formula ϕ), written $|t|$ ($|\phi|$), is the cardinality of $\mathrm{pos}(t)$, i.e., $|t| := |\,\mathrm{pos}(t)|$ ($|\phi| := |\,\mathrm{pos}(\phi)|$). The *depth* of a term, formula is the maximal

length of a position in the term, formula: $\text{depth}(t) := max\{|p| \mid p \in \text{pos}(t)\}$ ($\text{depth}(\phi) := max\{|p| \mid p \in \text{pos}(\phi)\}$).

The set of *all* variables occurring in a term t (formula ϕ) is denoted by $\text{vars}(t)$ ($\text{vars}(\phi)$) and formally defined as $\text{vars}(t) := \{x \in \mathcal{X} \mid x = t|_p, p \in \text{pos}(t)\}$ ($\text{vars}(\phi) := \{x \in \mathcal{X} \mid x = \phi|_p, p \in \text{pos}(\phi)\}$). A term t (formula ϕ) is *ground* if $vars(t) = \emptyset$ ($vars(\phi) = \emptyset$). Note that $\text{vars}(\forall x.a \approx b) = \emptyset$ where a, b are constants. This is justified by the fact that the formula does not depend on the quantifier, see the semantics below. The set of *free* variables of a formula ϕ (term t) is given by $\text{fvars}(\phi, \emptyset)$ ($\text{fvars}(t, \emptyset)$) and recursively defined by $\text{fvars}(\psi_1 \circ \psi_2, B) := \text{fvars}(\psi_1, B) \cup \text{fvars}(\psi_2, B)$ where $\circ \in \{\wedge, \vee, \rightarrow, \leftrightarrow\}$, $\text{fvars}(\forall x.\psi, B) := \text{fvars}(\psi, B \cup \{x\})$, $\text{fvars}(\exists x.\psi, B) := \text{fvars}(\psi, B \cup \{x\})$, $\text{fvars}(\neg\psi, B) := \text{fvars}(\psi, B)$, $\text{fvars}(L, B) := \text{vars}(L) \backslash B$ ($\text{fvars}(t, B) := \text{vars}(t) \backslash B$. For $\text{fvars}(\phi, \emptyset)$ I also write $\text{fvars}(\phi)$.

The function top maps terms to their top symbols, i.e., $\text{top}(f(t_1, \ldots, t_n)) := f$ and $\text{top}(x) := x$ for some variable x.

In $\forall x.\phi$ ($\exists x.\phi$) the formula ϕ is called the *scope* of the quantifier. An occurrence q of a variable x in a formula ϕ ($\phi|_q = x$) is called *bound* if there is some $p < q$ with $\phi|_p = \forall x.\phi'$ or $\phi|_p = \exists x.\phi'$. Any other occurrence of a variable is called *free*. A formula not containing a free occurrence of a variable is called *closed*. If $\{x_1, \ldots, x_n\}$ are the variables freely occurring in a formula ϕ then $\forall x_1, \ldots, x_n.\phi$ and $\exists x_1, \ldots, x_n.\phi$ (abbreviations for $\forall x_1.\forall x_2 \ldots \forall x_n.\phi$, $\exists x_1.\exists x_2 \ldots \exists x_n.\phi$, respectively) are the *universal* and the *existential closure* of ϕ, respectively.

Example 3.1.6. For the literal $\neg P(f(x, g(a)))$ the atom $P(f(x, g(a)))$ is an immediate subformula occurring at position 1. The terms x and $g(a)$ are strict subterms occurring at positions 111 and 112, respectively. The formula $\neg P(f(x, g(a)))[b]_{111} = \neg P(f(b, g(a)))$ is obtained by replacing x with b. $\text{pos}(\neg P(f(x, g(a)))) = \{\epsilon, 1, 11, 111, 112, 1121\}$ meaning its size is 6, its depth 4 and $vars(\neg P(f(x, g(a)))) = \{x\}$.

Definition 3.1.7 (Polarity)**.** The *polarity* of a subformula $\psi = \phi|_p$ at position p is $pol(\phi, p)$ where *pol* is recursively defined by

$$\begin{aligned}
\text{pol}(\phi, \epsilon) &:= 1 \\
\text{pol}(\neg\phi, 1p) &:= -\,\text{pol}(\phi, p) \\
\text{pol}(\phi_1 \circ \phi_2, ip) &:= \text{pol}(\phi_i, p) \text{ if } \circ \in \{\wedge, \vee\} \\
\text{pol}(\phi_1 \rightarrow \phi_2, 1p) &:= -\,\text{pol}(\phi_1, p) \\
\text{pol}(\phi_1 \rightarrow \phi_2, 2p) &:= \text{pol}(\phi_2, p) \\
\text{pol}(\phi_1 \leftrightarrow \phi_2, ip) &:= 0 \\
\text{pol}(P(t_1, \ldots, t_n), p) &:= 1 \\
\text{pol}(t \approx s, p) &:= 1 \\
\text{pol}(\forall x.\phi, 1p) &:= \text{pol}(\phi, p) \\
\text{pol}(\exists x.\phi, 1p) &:= \text{pol}(\phi, p)
\end{aligned}$$

3.2 Semantics

Definition 3.2.1 (Σ-algebra)**.** Let $\Sigma = (\mathcal{S}, \Omega, \Pi)$ be a signature with set of sorts $\mathcal{S}$, operator set Ω and predicate set Π. A Σ-*algebra* $\mathcal{A}$, also called Σ-*interpretation*, is a mapping that assigns (i) a non-empty carrier set $S^{\mathcal{A}}$ to every sort $S \in \mathcal{S}$, so that $(S_1)^{\mathcal{A}} \cap (S_2)^{\mathcal{A}} = \emptyset$ for any distinct sorts $S_1, S_2 \in \mathcal{S}$, (ii) a total function $f^{\mathcal{A}} : (S_1)^{\mathcal{A}} \times \ldots \times (S_n)^{\mathcal{A}} \to (S)^{\mathcal{A}}$ to every operator $f \in \Omega$, $\text{arity}(f) = n$ where $f : S_1 \times \ldots \times S_n \to S$, (iii) a relation $P^{\mathcal{A}} \subseteq ((S_1)^{\mathcal{A}} \times \ldots \times (S_m)^{\mathcal{A}})$ to every predicate symbol $P \in \Pi$, $\text{arity}(P) = m$. (iv) the equality relation becomes $\approx^{\mathcal{A}} = \{(e, e) \mid e \in \mathcal{U}^{\mathcal{A}}\}$ where the set $\mathcal{U}^{\mathcal{A}} := \bigcup_{S \in \mathcal{S}} (S)^{\mathcal{A}}$ is called the *universe* of $\mathcal{A}$.

A (variable) *assignment*, also called a *valuation* for an algebra $\mathcal{A}$ is a function $\beta : \mathcal{X} \to \mathcal{U}_{\mathcal{A}}$ so that $\beta(x) \in S_{\mathcal{A}}$ for every variable $x \in \mathcal{X}$, where $S = \text{sort}(x)$. A *modification* $\beta[x \mapsto e]$ of an assignment β at a variable $x \in \mathcal{X}$, where $e \in S_{\mathcal{A}}$ and $S = \text{sort}(x)$, is the assignment defined as follows:

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

Informally speaking, the assignment $\beta[x \mapsto e]$ is identical to β for every variable except x, which is mapped by $\beta[x \mapsto e]$ to e.

The homomorphic extension $\mathcal{A}(\beta)$ of β onto terms is a mapping $T(\Sigma, \mathcal{X}) \to \mathcal{U}_{\mathcal{A}}$ defined as (i) $\mathcal{A}(\beta)(x) = \beta(x)$, where $x \in \mathcal{X}$ and (ii) $\mathcal{A}(\beta)(f(t_1, \ldots, t_n)) = f_{\mathcal{A}}(\mathcal{A}(\beta)(t_1), \ldots, \mathcal{A}(\beta)(t_n))$, where $f \in \Omega$, $\text{arity}(f) = n$.

Given a term $t \in T(\Sigma, \mathcal{X})$, the value $\mathcal{A}(\beta)(t)$ is called the *interpretation* of t under $\mathcal{A}$ and β. If the term t is ground, the value $\mathcal{A}(\beta)(t)$ does not depend on a particular choice of β, for which reason the interpretation of t under $\mathcal{A}$ is denoted by $\mathcal{A}(t)$.

An algebra $\mathcal{A}$ is called *term-generated*, if every element e of the universe $\mathcal{U}_{\mathcal{A}}$ of $\mathcal{A}$ is the image of some ground term t, i.e., $\mathcal{A}(t) = e$.

Definition 3.2.2 (Semantics)**.** An algebra $\mathcal{A}$ and an assignment β are extended to formulas $\phi \in \text{FOL}(\Sigma, \mathcal{X})$ by

$$\begin{array}{rcl}
\mathcal{A}(\beta)(\bot) & := & 0 \\
\mathcal{A}(\beta)(\top) & := & 1 \\
\mathcal{A}(\beta)(s \approx t) & := & 1 \text{ if } \mathcal{A}(\beta)(s) = \mathcal{A}(\beta)(t) \text{ and 0 otherwise} \\
\mathcal{A}(\beta)(P(t_1, \ldots, t_n)) & := & 1 \text{ if } (\mathcal{A}(\beta)(t_1), \ldots, \mathcal{A}(\beta)(t_n)) \in P^{\mathcal{A}} \text{ and 0 otherwise} \\
\mathcal{A}(\beta)(\neg\phi) & := & 1 - \mathcal{A}(\beta)(\phi) \\
\mathcal{A}(\beta)(\phi \wedge \psi) & := & \min(\{\mathcal{A}(\beta)(\phi), \mathcal{A}(\beta)(\psi)\}) \\
\mathcal{A}(\beta)(\phi \vee \psi) & := & \max(\{\mathcal{A}(\beta)(\phi), \mathcal{A}(\beta)(\psi)\}) \\
\mathcal{A}(\beta)(\phi \to \psi) & := & \max(\{(1 - \mathcal{A}(\beta)(\phi)), \mathcal{A}(\beta)(\psi)\}) \\
\mathcal{A}(\beta)(\phi \leftrightarrow \psi) & := & \text{if } \mathcal{A}(\beta)(\phi) = \mathcal{A}(\beta)(\psi) \text{ then 1 else 0} \\
\mathcal{A}(\beta)(\exists x_S.\phi) & := & 1 \text{ if } \mathcal{A}(\beta[x \mapsto e])(\phi) = 1 \text{ for some } e \in S_{\mathcal{A}} \text{ and 0 otherwise} \\
\mathcal{A}(\beta)(\forall x_S.\phi) & := & 1 \text{ if } \mathcal{A}(\beta[x \mapsto e])(\phi) = 1 \text{ for all } e \in S_{\mathcal{A}} \text{ and 0 otherwise}
\end{array}$$

A formula ϕ is called *satisfiable by* $\mathcal{A}$ *under* β (or *valid in* $\mathcal{A}$ *under* β) if $\mathcal{A}, \beta \models \phi$; in this case, ϕ is also called *consistent*; *satisfiable by* $\mathcal{A}$ if $\mathcal{A}, \beta \models \phi$ for some assignment β; *satisfiable* if $\mathcal{A}, \beta \models \phi$ for some algebra $\mathcal{A}$ and some assignment β; *valid in* $\mathcal{A}$, written $\mathcal{A} \models \phi$, if $\mathcal{A}, \beta \models \phi$ for any assignment β; in this case, $\mathcal{A}$ is called a *model* of ϕ; *valid*, written $\models \phi$, if $\mathcal{A}, \beta \models \phi$ for any algebra $\mathcal{A}$ and any assignment β; in this case, ϕ is also called a *tautology*; *unsatisfiable* if $\mathcal{A}, \beta \not\models \phi$ for any algebra $\mathcal{A}$ and any assignment β; in this case ϕ is also called *inconsistent*.

Note that $\bot$ is inconsistent whereas $\top$ is valid. If ϕ is a sentence that is a formula not containing a free variable, it is valid in $\mathcal{A}$ if and only if it is satisfiable by $\mathcal{A}$. This means the truth of a sentence does not depend on the choice of an assignment.

Given two formulas ϕ and ψ, ϕ *entails* ψ, or ψ is a *consequence* of ϕ, written $\phi \models \psi$, if for any algebra $\mathcal{A}$ and assignment β, if $\mathcal{A}, \beta \models \phi$ then $\mathcal{A}, \beta \models \psi$. The formulas ϕ and ψ are called *equivalent*, written $\phi \models\!\!\models \psi$, if $\phi \models \psi$ and $\psi \models \phi$. Two formulas ϕ and ψ are called *equisatisfiable*, if ϕ is satisfiable iff ψ is satisfiable (not necessarily in the same models). Note that if ϕ and ψ are equivalent then they are equisatisfiable, but not the other way around. The notions of "entailment", "equivalence" and "equisatisfiability" are naturally extended to sets of formulas, that are treated as conjunctions of single formulas. Thus, given formula sets M_1 and M_2, the set M_1 entails M_2, written $M_1 \models M_2$, if for any algebra $\mathcal{A}$ and assignment β, if $\mathcal{A}, \beta \models \phi$ for every $\phi \in M_1$ then $\mathcal{A}, \beta \models \psi$ for every $\psi \in M_2$. The sets M_1 and M_2 are equivalent, written $M_1 \models\!\!\models M_2$, if $M_1 \models M_2$ and $M_2 \models M_1$. Given an arbitrary formula ϕ and formula set M, $M \models \phi$ is written to denote $M \models \{\phi\}$; analogously, $\phi \models M$ stands for $\{\phi\} \models M$.

Clauses are implicitly universally quantified disjunctions of literals. A clause C is satisfiable by an algebra $\mathcal{A}$ if for every assignment β there is a literal $L \in C$ with $\mathcal{A}, \beta \models L$. Note that if $C = \{L_1, \ldots, L_k\}$ is a ground clause, i.e., every L_i is a ground literal, then $\mathcal{A} \models C$ if and only if there is a literal L_j in C so that $\mathcal{A} \models L_j$. A clause set N is satisfiable iff all clauses $C \in N$ are satisfiable by the same algebra $\mathcal{A}$. Accordingly, if N and M are two clause sets, $N \models M$ iff every model $\mathcal{A}$ of N is also a model of M.

Definition 3.2.3 (Congruence)**.** Let $\Sigma = (\mathcal{S}, \Omega, \Pi)$ be a signature and $\mathcal{A}$ a Σ-algebra. A *congruence* $\sim$ is an equivalence relation on $(S_1)^{\mathcal{A}} \cup \ldots \cup (S_n)^{\mathcal{A}}$ such that

1. if $a \sim b$ then there is an $S \in \mathcal{S}$ such that $a \in S^{\mathcal{A}}$ and $b \in S^{\mathcal{A}}$
2. for all $a_i \sim b_i$, $a_i, b_i \in (S_i)^{\mathcal{A}}$ and all functions $f : S_1 \times \ldots \times S_n \to S$ it holds $f^{\mathcal{A}}(a_1, \ldots, a_n) \sim f^{\mathcal{A}}(b_1, \ldots, b_n)$
3. for all $a_i \sim b_i$, $a_i, b_i \in (S_i)^{\mathcal{A}}$ and all predicates $P \subseteq S_1 \times \ldots \times S_n$ it holds $(a_1, \ldots, a_n) \in P^{\mathcal{A}}$ iff $(b_1, \ldots, b_n) \in P^{\mathcal{A}}$

The first condition guarantees that a congruence $\sim$ respects the disjoint sort structure. The second requires compatibility with function applications and the third compatibility with predicate definitions. Actually, for any Σ-algebra $\mathcal{A}$ the

interpretation of equality $\approx^{\mathcal{A}}$ is a congruence, Exercise **??**. Further on in this chapter I will also show that the other way round can hold as well: given a suitable congruence on some set, the equivalence classes of the congruence can then serve as the domain of a Σ-algebra providing a suitable interpretation for equality.

3.3 Substitutions

For a concrete propositional logic interpretation, it is sufficient select a valuation, i.e., truth values for the propositional variables, see Section 2.2. In first-order logic this becomes more versatile. The truth values for propositional variables correspond to n-ary relations on the domain with respect to valuations for the first-order variables, see Section 3.2. So in addition to the 0-relations for propositional variables, n-ary relations need to be considered under an assignment β for the first-order variables. When calculi for propositional logic considered partial interpretations, e.g., Tableau (Section 2.4) or CDCL (Section **??**)), they are presented by sets of propositional literals taken from the processed clause set. For first-order logic this corresponds to taking first-order literals from the clause set and then instantiating the variables in these literals with terms in order to detect conflicts or for propagation. For example, a first-order clause $\neg P(x) \vee T(x)$ with universally quantified x propagates the literal $T(f(y))$ under the partial interpretation $P(f(y))$ where x is instantiated with $f(y)$. This instantiation is the syntactic counterpart of an assignment and represented by *substitutions* represented below.

Definition 3.3.1 (Substitution (well-sorted))**.** A *well-sorted substitution* is a mapping $\sigma : \mathcal{X} \to T(\Sigma, \mathcal{X})$ so that

1. $\sigma(x) \neq x$ for only finitely many variables x and
2. $\mathrm{sort}(x) = \mathrm{sort}(\sigma(x))$ for every variable $x \in \mathcal{X}$.

The application $\sigma(x)$ of a substitution σ to a variable x is often written in postfix notation as $x\sigma$. The variable set $\mathrm{dom}(\sigma) := \{x \in \mathcal{X} \mid x\sigma \neq x\}$ is called the *domain* of σ. The term set $\mathrm{codom}(\sigma) := \{x\sigma \mid x \in \mathrm{dom}(\sigma)\}$ is called the *codomain* of σ. From the above definition it follows that $\mathrm{dom}(\sigma)$ is finite for any substitution σ. The composition of two substitutions σ and τ is written as a juxtaposition $\sigma\tau$, i.e., $t\sigma\tau = (t\sigma)\tau$. A substitution σ is called *idempotent* if $\sigma\sigma = \sigma$. A substitution σ is idempotent iff $\mathrm{dom}(\sigma) \cap \mathrm{vars}(\mathrm{codom}(\sigma)) = \emptyset$.

Substitutions are often written as sets of pairs $\{x_1 \mapsto t_1, \ldots, x_n \mapsto t_n\}$ if $dom(\sigma) = \{x_1, \ldots, x_n\}$ and $x_i\sigma = t_i$ for every $i \in \{1, \ldots, n\}$. The *modification* of a substitution σ at a variable x is defined as follows:

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

A substitution σ is identified with its extension to formulas and defined as follows:

1. $\bot\sigma = \bot$,
2. $\top\sigma = \top$,
3. $(f(t_1, \ldots, t_n))\sigma = f(t_1\sigma, \ldots, t_n\sigma)$,
4. $(P(t_1, \ldots, t_n))\sigma = P(t_1\sigma, \ldots, t_n\sigma)$,
5. $(s \approx t)\sigma = (s\sigma \approx t\sigma)$,
6. $(\neg\phi)\sigma = \neg(\phi\sigma)$,
7. $(\phi \circ \psi)\sigma = \phi\sigma \circ \psi\sigma$ where $\circ \in \{\vee, \wedge\}$,
8. $(Qx\phi)\sigma = Qz(\phi\sigma[x \mapsto z])$ where $Q \in \{\forall, \exists\}$, z and x are of the same sort and z is a fresh variable.

The result $t\sigma$ ($\phi\sigma$) of applying a substitution σ to a term t (formula ϕ) is called an *instance* of t (ϕ). The substitution σ is called *ground* if it maps every domain variable to a ground term, i.e., the codomain of σ consists of ground terms only. If the application of a substitution σ to a term t (formula ϕ) produces a ground term $t\sigma$ (a variable-free formula, $\mathrm{vars}(\phi\sigma) = \emptyset$), then $t\sigma$ ($\phi\sigma$) is called *ground instance* of t (ϕ) and σ is called *grounding* for t (ϕ). The set of ground instances of a clause set N is given by $\mathrm{grd}(\Sigma, N) = \{C\sigma \mid C \in N, \sigma \text{ is grounding for } C\}$ is the set of *ground instances* of N. A substitution σ is called a *variable renaming* if $\mathrm{codom}(\sigma) \subseteq \mathcal{X}$ and for any $x, y \in \mathcal{X}$, if $x \neq y$ then $x\sigma \neq y\sigma$.

The following lemma establishes the relationship between substitutions and assignments.

Lemma 3.3.2 (Substitutions and Assignments)**.** Let β be an assignment of some interpretation $\mathcal{A}$ of a term t and σ a substitution. Then

$$\beta(t\sigma) = \beta[x_1 \mapsto \beta(x_1\sigma), \ldots, x_n \mapsto \beta(x_n\sigma)](t)$$

where $\mathrm{dom}(\sigma) = \{x_1, \ldots, x_n\}$.

Proof. By structural induction on t. If $t = a$ is a constant, then $\beta(a\sigma) = a^{\mathcal{A}} = \beta[x_1 \mapsto \beta(x_1\sigma), \ldots, x_n \mapsto \beta(x_n\sigma)](a)$. The case $t = x$ is a variable and $x \notin \mathrm{dom}(\sigma)$ is identical to the case that t is a constant. So $t = x_i$ is a variable and $x_i \in \mathrm{dom}(\sigma)$, where $x_i\sigma = s$. If s is a variable, then $\beta(t\sigma) = \beta(x_i\sigma) = \beta(s) = \beta[x_i \mapsto \beta(s)](x_i) = \beta[x_1 \mapsto \beta(x_1\sigma), \ldots, x_n \mapsto \beta(x_n\sigma)](t)$. The case s is a constant is analogous to the case t is a constant. So let $x_i\sigma = s = f(s_1, \ldots, s_m)$. $\beta(x_i\sigma) = \beta(f(s_1, \ldots, s_m)) = f^{\mathcal{A}}(\beta(s_1), \ldots, \beta(s_m)) = \beta[x_i \mapsto f(s_1, \ldots, s_m)](x_i) = \beta[x_1 \mapsto \beta(x_1\sigma), \ldots, x_n \mapsto \beta(x_n\sigma)](t)$.
For the inductive case let $t = f(t_1, \ldots, t_m)$. Then $\beta(t\sigma) = f^{\mathcal{A}}(\beta(t_1\sigma), \ldots, \beta(t_m\sigma)) = f^{\mathcal{A}}(\beta[x_1 \mapsto \beta(x_1\sigma), \ldots, x_n \mapsto \beta(x_n\sigma)](t_1), \ldots, \beta[x_1 \mapsto \beta(x_1\sigma), \ldots, x_n \mapsto \beta(x_n\sigma)](t_m)) = \beta[x_1 \mapsto \beta(x_1\sigma), \ldots, x_n \mapsto \beta(x_n\sigma)](t)$. □

3.4 Equality

The equality predicate is build into the first-order language in Section 3.1 and not part of the signature. It is a first class citizen. This is the case although it can be actually axiomatized in the language. The motivation is that firstly, many real world problems naturally contain equations. They are a means to define functions. Then predicates over terms model properties of the functions. Secondly, without special treatment in a calculus, it is almost impossible to automatically prove non-trivial properties of a formula containing equations.

In this section I firstly show that any formula can be transformed into a formula where all atoms are equations. Secondly, that any formula containing equations can be transformed into a formula where the equality predicate is replaced by a fresh predicate together with some axioms. In the first case the respective clause sets are equivalent, in the second case the transformation is satisfiability preserving. For the replacement of any predicate R by equations over a fresh function f_R we assume an additional fresh sort Bool with a fresh constant true.

InjEq $\chi[R(t_{1,1},\ldots,t_{1,n})]_{p_1}\ldots[R(t_{m,1},\ldots,t_{m,n})]_{p_m} \Rightarrow_{\mathrm{IE}} \chi[f_R(t_{1,1},\ldots,t_{1,n}) \approx \mathrm{true}]_{p_1}\ldots[f_R(t_{m,1},\ldots,t_{m,n}) \approx \mathrm{true}]_{p_m}$
provided R is a predicate occurring in χ, $\{p_1,\ldots,p_m\}$ are all positions of atoms with predicate R in χ and f_R is new with appropriate sorting

Proposition 3.4.1. Let $\chi \Rightarrow^*_{\mathrm{IE}} \chi'$ then χ is satisfiable (valid) iff χ' is satisfiable (valid).

Proof. (Sketch) The basic proof idea is to establish the relation $(t_1^{\mathcal{A}},\ldots,t_n^{\mathcal{A}}) \in R^{\mathcal{A}}$ iff $f_R^{\mathcal{A}}(t_1^{\mathcal{A}},\ldots,t_n^{\mathcal{A}}) = \mathrm{true}^{\mathcal{A}}$. Furthermore, the sort of true is fresh to χ and the equations $f_R(t_1,\ldots,t_n) \approx \mathrm{true}$ do not interfere with any term t_i because the f_R are all fresh and only occur on top level of the equations. □

When removing equality from a formula it needs to be axiomatized. For simplicity, I assume here that the considered formula χ is one-sorted, i.e., there is only one sort occurring for functions, relations in χ. The extension to formulas with many sorts is straightforward and discussed below.

RemEq $\chi[l_1 \approx r_1]_{p_1}\ldots[l_m \approx r_m]_{p_m} \Rightarrow_{\mathrm{RE}} \chi[E(l_1,r_1)]_{p_1}\ldots[E(l_m,r_m)]_{p_m} \wedge \mathrm{def}(\chi,E)$
provided $\{p_1,\ldots,p_m\}$ are all positions of equations $l_i = r_i$ in χ and E is a new binary predicate

The formula $\mathrm{def}(\chi,E)$ is the axiomatization of equality for χ and it consists of a conjunction of the equivalence relation axioms for E
$\forall x.E(x,x)$
$\forall x,y.(E(x,y) \rightarrow E(y,x))$
$\forall x,y,z.((E(x,y) \wedge E(x,z)) \rightarrow E(x,z))$
plus the congruence axioms for E for every n-ary function symbol f

$$\forall x_1, y_1, \ldots, x_n, y_n.((E(x_1, y_1) \wedge \ldots \wedge E(x_n, y_n)) \\ \rightarrow E(f(x_1, \ldots, x_n), f(y_1, \ldots, y_n)))$$

plus the congruence axioms for E for every m-ary predicate symbol P

$$\forall x_1, y_1, \ldots, x_m, y_m.((E(x_1, y_1) \wedge \ldots \wedge E(x_m, y_m) \wedge P(x_1, \ldots, x_m)) \\ \rightarrow P(y_1, \ldots, y_m)$$

Proposition 3.4.2. Let $\chi \Rightarrow_{\mathrm{RE}} \chi'$ then χ is satisfiable iff χ' is satisfiable.

Proof. (Sketch) The identity on an algebra (see Definition 3.2.2) is a congruence relation proving the direction from left to right. The direction from right to left is more involved. □

Note that $\Rightarrow_{\mathrm{RE}}$ is not validity preserving. Consider the simple example formula $a \approx a$ which is valid for any constant a. Its translation $E(a, a) \wedge \mathrm{def}(a \approx a, E)$ is not valid, e.g., consider an algebra with $E^{\mathcal{A}} = \emptyset$.

Now in case χ has many different sorts then for each sort S one new fresh predicate E_S is needed for the translation. For each of these predicates equivalence relation and congruence axioms need to be generated where for every function f only one axiom using E_S is needed, where S is the range sort of S. Similar for the domain sorts of f and accordingly for predicates.

3.5 Herbrand's Theorem

There are substantial differences between propositional logic and its generalization first-order logic. There are only finitely many formulas in propositional logic that can be semantically distinguished for some finite signature. Given a finite propositional signature Σ there are "only" $2^{|\Sigma|}$ different valuations. In first-order logic there are infinitely many different interpretations for formulas over some finite first-order signature Σ. As we will see, this moves the satisfiability problem for some set of clauses from NP (propositional) to undecidable (first-order), see Section 3.15. In this section I present two results that are the basis for most first-order calculi. Firstly, I show that when considering satisfiability of a clause set, it is not necessary to consider arbitrary interpretations. Instead, one specific interpretation, called *Herbrand interpretation*, is sufficient for establishing satisfiability. Secondly, interpretations for first-order clause sets, including Herbrand interpretations, typically consider an infinite domain. This implies infinitely many different assignments defining the semantics for a clause set. Still, if some clause set is unsatisfiable, then finitely many assignments are sufficient to prove unsatisfiability. This property is called *Compactness* of first-order logic. Putting the two results together, it is sufficient to consider finitely many assignments from the Herbrand interpretation in order to prove unsatisfiability of a set of clauses: the basis for all modern automated reasoning calculi for first-order logic.

Definition 3.5.1 (Herbrand Interpretation)**.** A *Herbrand Interpretation* (over Σ) is a Σ-algebra $\mathcal{H}$ such that

1. $S^{\mathcal{H}} := T_S(\Sigma)$ for every sort $S \in \mathcal{S}$
2. $f^{\mathcal{H}} : (s_1, \ldots, s_n) \mapsto f(s_1, \ldots, s_n)$ where $f \in \Omega$, $\text{arity}(f) = n$, $s_i \in S_i^{\mathcal{H}}$ and $f : S_1 \times \ldots \times S_n \to S$ is the sort declaration for f
3. $P^{\mathcal{H}} \subseteq (S_1^{\mathcal{H}} \times \ldots \times S_m^{\mathcal{H}})$ where $P \in \Pi$, $\text{arity}(P) = m$ and $P \subseteq S_1 \times \ldots \times S_m$ is the sort declaration for P

Lemma 3.5.2 (Herbrand Interpretations are Well-Defined)**.** Every Herbrand Interpretation is a Σ-algebra.

Proof. (i) the carriers are non-empty because every signature contains a constant declaration for each sort. If $S^{\mathcal{H}} \cap T^{\mathcal{H}} \neq \emptyset$, then there must be two declarations for the same function symbol in Σ which is forbidden. Furthermore, $\sim$ is well-sorted.
(ii) functions are total by definition.
(iii) relations are assigned. □

In other words, values for ground terms are fixed to be the ground terms itself and functions are fixed to be the term constructors. Predicate symbols may be freely interpreted as relations over ground terms.

Proposition 3.5.3 (Representing Herbrand Interpretations)**.** A Herbrand interpretation $\mathcal{A}$ can be uniquely determined by a set of ground atoms I

$$(s_1, \ldots, s_n) \in P^{\mathcal{A}} \text{ iff } P(s_1, \ldots, s_n) \in I$$

Thus Herbrand interpretations (over Σ) can be identified with sets of Σ-ground atoms. A Herbrand interpretation I is called a *Herbrand model* of ϕ, where I assume ϕ does not contain equations, if $I \models \phi$.

Historically, Herbrand interpretations have been defined for first-order logic without equality. These are exactly the definitions above. Later on, I'll extend these notions such that they also cover the case of equations.

Example 3.5.4. Consider the signature $\Sigma = (\{S\}, \{a, b\}, \{P, Q\})$, where a, b are constants, $\text{arity}(P) = 1$, $\text{arity}(Q) = 2$, and all constants, predicates are defined over the sort S. Then the following are examples of Herbrand interpretations over Σ, where for all interpretations $S_{\mathcal{A}} = \{a, b\}$.

$I_1 := \emptyset$
$I_2 := \{P(a), Q(a, a), Q(b, b)\}$
$I_3 := \{P(a), P(b), Q(a, a), Q(b, b), Q(a, b), Q(b, a)\}$

Now consider the extension Σ' of Σ by one unary function symbol $g : S \to S$. Then the following are examples of Herbrand interpretations over Σ', where for all interpretations $S_{\mathcal{A}} = \{a, b, g(a), g(b), g(g(a)), \ldots\}$.

$I'_1 := \emptyset$
$I'_2 := \{P(a), Q(a, g(a)), Q(b, b)\}$
$I'_3 := \{P(a), P(g(a)), P(g(g(a))), \ldots, Q(a, a), Q(b, b), Q(b, g(b)), Q(b, g(g(b))), \ldots\}$

Theorem 3.5.5 (Herbrand's Theorem)**.** Let N be a finite set of Σ-clauses without equality. Then N is satisfiable iff N has a Herbrand model over Σ iff $\text{grd}(\Sigma, N)$ has a Herbrand model over Σ.

Proof. Firstly, I prove that if N has a model, then it has a Herbrand model over Σ. So let $\mathcal{A}$ be a model for N. Since N is finite let's consider exactly the subsignature of N. Then $P^{\mathcal{H}} = \{(t_1, \ldots, t_n) \mid (t_1^{\mathcal{A}}, \ldots, t_n^{\mathcal{A}}) \in P^{\mathcal{A}}, t_i \in T(\Sigma)\}$. Finally, I need to prove that $\mathcal{H}$ is a model for N. Assume not. Then there is a clause $C \in N$ and an assignment $\beta_{\mathcal{H}}$ such that $\mathcal{H}(\beta_{\mathcal{H}})(C) = 0$ where $\beta_{\mathcal{H}}(x_i) = t_i$ for all $x_i \in \text{vars}(C)$ with $t_i \in T_{\text{sort}(x_i)}(\Sigma)$. Let $\sigma = \{x_1 \mapsto t_1, \ldots, x_m \mapsto t_m\}$. Now consider an assignment $\beta_{\mathcal{A}}$ where $\beta_{\mathcal{A}}(x_i) = t_i^{\mathcal{A}}$. Since $\mathcal{A} \models N$ also $\mathcal{A}(\beta_{\mathcal{A}}) \models C$, in particular, there is a literal $L \in C$ with $\mathcal{A}(\beta_{\mathcal{A}})(L) = 1$. If it is an atom $P(l_1, \ldots, l_n)$ with $(\mathcal{A}(\beta_{\mathcal{A}})(l_1), \ldots, \mathcal{A}(\beta_{\mathcal{A}})(l_n)) \in P^{\mathcal{A}}$, but then $(l_1\sigma, \ldots, l_n\sigma) \in P^{\mathcal{H}}$ by definition of $\mathcal{H}$ and Lemma 3.3.2. Hence $(\mathcal{H}(\beta_{\mathcal{H}})(l_1), \ldots, \mathcal{H}(\beta_{\mathcal{H}})(l_n)) \in P^{\mathcal{H}}$, a contradiction. The case where L is negative is dual.
Secondly, due to Lemma 3.5.2 the existence of a Herbrand model implies satisfiability.
It remains to be shown that N has a Herbrand model over Σ iff $\text{grd}(\Sigma, N)$ has a Herbrand model. Firstly, assume N has a Herbrand model $\mathcal{H}$ over Σ. Then $\mathcal{H}$ is also a model for $\text{grd}(\Sigma, N)$. Assume not. Then there is a clause $C\sigma \in \text{grd}(\Sigma, N)$, $C \in N$, such that $\mathcal{H} \not\models C\sigma$. But then $\mathcal{H}(\beta_{\mathcal{H}}[x_1 \mapsto (x_1\sigma), \ldots, x_n \mapsto (x_n\sigma)])(C) = 0$, $\text{dom}(\sigma) = \{x_1, \ldots, x_n\}$, contradicting $\mathcal{H}$ is a model for N. Secondly, assume $\mathcal{H}$ is a model for $\text{grd}(\Sigma, N)$. Then $\mathcal{H}$ is also a model for N. Assume not, then there is a clause $C \in N$ and ad assignment $\beta_{\mathcal{H}}[x_1 \mapsto (x_1\sigma), \ldots, x_n \mapsto (x_n\sigma)]$, $vars(C) = \{x_1, \ldots, x_n\}$, such that $\mathcal{H}(\beta_{\mathcal{H}}[x_1 \mapsto (x_1\sigma), \ldots, x_n \mapsto (x_n\sigma)])(C) = 0$. But then $\mathcal{H} \not\models C\sigma$, contradicting $\mathcal{H}$ is a model for $\text{grd}(\Sigma, N)$.

□

Example 3.5.6 (Example of a $\text{grd}(\Sigma, N)$)**.** Consider Σ' from Example 3.5.4 and the clause set $N = \{Q(x, x) \vee \neg P(x), \neg P(x) \vee P(g(x))\}$. Then the set of ground instances $\text{grd}(\Sigma', N) = \{$

$Q(a, a) \vee \neg P(a)$
$Q(b, b) \vee \neg P(b)$
$Q(g(a), g(a)) \vee \neg P(g(a))$
$\ldots$
$\neg P(a) \vee P(g(a))$
$\neg P(b) \vee P(g(b))$
$\neg P(g(a)) \vee P(g(g(a)))$
$\ldots\}$

is satisfiable. For example by the Herbrand models

$I_1 := \emptyset$
$I_2 := \{P(b), Q(b, b), P(g(b)), Q(g(b), g(b)), \ldots\}$

Definition 3.5.7 (Herbrand Interpretation with Equality)**.** A *Herbrand Interpretation* (over Σ) is a Σ-algebra $\mathcal{H}$ such that

1. a well-sorted equivalence relation $\sim$ on $T(\Sigma)$, i.e., if $s \sim t$ then $s, t \in T_S(\Sigma)$ for some S where $[s]$ denotes the equivalence class containing s

2. $S^{\mathcal{H}} := T_S(\Sigma)/\sim$ for every sort $S \in \mathcal{S}$
3. $f^{\mathcal{H}} : ([s_1], \ldots, [s_n]) \mapsto [f(s_1, \ldots, s_n)]$ where $f \in \Omega$, $\text{arity}(f) = n$, $s_i \in T_{S_i}(\Sigma)$ and $f : S_1 \times \ldots \times S_n \to S$ is the sort declaration for f
4. $P^{\mathcal{H}} \subseteq (S_1^{\mathcal{H}} \times \ldots \times S_m^{\mathcal{H}})$ where $P \in \Pi$, $\text{arity}(P) = m$ and $P \subseteq S_1 \times \ldots \times S_m$ is the sort declaration for P

Lemma 3.5.8 (Herbrand Interpretations are Well-Defined)**.** Every Herbrand Interpretation is a Σ-algebra.

Proof. (i) the carriers are non-empty because every signature contains a constant declaration for each sort. If $S^{\mathcal{H}} \cap T^{\mathcal{H}} \neq \emptyset$, then there must be two declarations for the same function symbol in Σ which is forbidden. Furthermore, $\sim$ is well-sorted.

(ii) functions are total by definition.

(iii) relations are assigned. □

In other words, values are fixed to be equivalence classes of ground terms and functions are fixed to be the term constructors. Predicate symbols may be freely interpreted as relations over equivalence classes of ground terms.

Proposition 3.5.9. A Herbrand interpretation $\mathcal{A}$ can be uniquely determined by a set of ground atoms I

$$\begin{array}{rl} (s_1, \ldots, s_n) \in P_{\mathcal{A}} & \text{iff } P(s_1, \ldots, s_n) \in I \\ t \sim s & \text{iff } s \approx t \in I \end{array}$$

Thus Herbrand interpretations (over Σ) can be identified with sets of Σ-ground atoms. A Herbrand interpretation I is called a *Herbrand model* of ϕ, if $I \models \phi$.

Historically, Herbrand interpretations have been defined for first-order logic without equality. The above definition and the below Herbrand theorem are generalizations. If no equality atoms are present, then they coincide with the classical definitions. However, I chose to include equality, because the definition now already suggests what is needed for a calculus in order to cope explicitly with equality.

Example 3.5.10. Consider the signature $\Sigma = (\{S\}, \{a, b\}, \{P, Q\})$, where a, b are constants, $\text{arity}(P) = 1$, $\text{arity}(Q) = 2$, and all constants, predicates are defined over the sort S. Then the following are examples of Herbrand interpretations over Σ, where for all interpretations $S_{\mathcal{A}} = \{a, b\}$.

$I_1 := \emptyset$
$I_2 := \{P(a), Q(a, a), Q(b, b)\}$
$I_3 := \{P(a), P(b), Q(a, a), Q(b, b), Q(a, b), Q(b, a)\}$

Now consider the extension Σ' of Σ by one unary function symbol $g : S \to S$. Then the following are examples of Herbrand interpretations over Σ', where for all interpretations $S_{\mathcal{A}} = \{a, b, g(a), g(b), g(g(a)), \ldots\}$.

Proof. If N is unsatisfiable, saturation via the tableau calculus generates a closed tableau. So there is an i such that $N \Rightarrow^i_{\text{TAB}} N'$ and N' is closed. Every closed branch is the result of finitely many tableau rule applications on finitely many clauses $\{C_1, \ldots, C_n\} \subseteq N$. Let M be the union of all these finite clause sets, so $M \subseteq N$. Tableau is sound, so M is a finite, unsatisfiable subset of N. □

3.7 Unification

Definition 3.7.1 (Unifier)**.** Two terms s and t of the same sort are said to be *unifiable* if there exists a well-sorted substitution σ so that $s\sigma = t\sigma$, the substitution σ is then called a well-sorted *unifier* of s and t. The unifier σ is called *most general unifier*, written $\sigma = mgu(s, t)$, if any other well-sorted unifier τ of s and t it can be represented as $\tau = \sigma\tau'$, for some well-sorted substitution τ'.

Obviously, two terms of different sort cannot be made equal by well-sorted instantiation. Since well-sortedness is preserved by all rules of the unification calculus, we assume from now an that all equations, terms, and substitutions are well-sorted.

The first calculus is the naive standard unification calculus that is typically found in the (old) literature on automated reasoning [**?**]. A state of the naive standard unification calculus is a set of equations E or $\bot$, where $\bot$ denotes that no unifier exists. The set E is also called a *unification problem*. The start state for checking whether two terms s, t, $\text{sort}(s) = \text{sort}(t)$, (or two non-equational atoms A, B) are unifiable is the set $E = \{s = t\}$ ($E = \{A = B\}$). A variable x is *solved* in E if $E = \{x = t\} \uplus E'$, $x \notin \text{vars}(t)$ and $x \notin \text{vars}(E)$.

A variable $x \in \text{vars}(E)$ is called *solved* in E if $E = E' \uplus \{x = t\}$ and $x \notin \text{vars}(t)$ and $x \notin \text{vars}(E')$.

Tautology $E \uplus \{t = t\} \Rightarrow_{\text{SU}} E$

Decomposition $E \uplus \{f(s_1, \ldots, s_n) = f(t_1, \ldots, t_n)\} \Rightarrow_{\text{SU}} E \cup \{s_1 = t_1, \ldots, s_n = t_n\}$

Clash $E \uplus \{f(s_1, \ldots, s_n) = g(s_1, \ldots, s_m)\} \Rightarrow_{\text{SU}} \bot$
if $f \neq g$

Substitution $E \uplus \{x = t\} \Rightarrow_{\text{SU}} E\{x \mapsto t\} \cup \{x = t\}$
if $x \in vars(E)$ and $x \notin vars(t)$

Occurs Check $E \uplus \{x = t\} \Rightarrow_{\text{SU}} \bot$

if $x \neq t$ and $x \in vars(t)$

Orient $E \uplus \{t = x\} \Rightarrow_{\mathrm{SU}} E \cup \{x = t\}$
if $t \notin \mathcal{X}$

Theorem 3.7.2 (Soundness, Completeness and Termination of $\Rightarrow_{\mathrm{SU}}$)**.** If s, t are two terms with $\mathrm{sort}(s) = \mathrm{sort}(t)$ then

1. if $\{s = t\} \Rightarrow^*_{\mathrm{SU}} E$ then any equation $(s' = t') \in E$ is well-sorted, i.e., $\mathrm{sort}(s') = \mathrm{sort}(t')$.
2. $\Rightarrow_{\mathrm{SU}}$ terminates on $\{s = t\}$.
3. if $\{s = t\} \Rightarrow^*_{\mathrm{SU}} E$ then σ is a unifier (mgu) of E iff σ is a unifier (mgu) of $\{s = t\}$.
4. if $\{s = t\} \Rightarrow^*_{\mathrm{SU}} \bot$ then s and t are not unifiable.
5. if $\{s = t\} \Rightarrow^*_{\mathrm{SU}} \{x_1 = t_1, \ldots, x_n = t_n\}$ and this is a normal form, then $\{x_1 \mapsto t_1, \ldots, x_n \mapsto t_n\}$ is an mgu of s, t.

Proof. 1. by induction on the length of the derivation and a case analysis for the different rules.
2. for a state $E = \{s_1 = t_1, \ldots, s_n = t_n\}$ take the measure $\mu(E) := (n, M, k)$ where n is the number of unsolved variables, M the multiset of all term depths of the s_i, t_i and k the number of equations $t = x$ in E where t is not a variable. The state $\bot$ is mapped to $(0, \emptyset, 0)$. Then the lexicographic combination of $>$ on the naturals and its multiset extension shows that any rule application decrements the measure.
3. by induction on the length of the derivation and a case analysis for the different rules. Clearly, for any state where Clash, or Occurs Check generate $\bot$ the respective equation is not unifiable.
4. a direct consequence of 3.
5. if $E = \{x_1 = t_1, \ldots, x_n = t_n\}$ is a normal form, then for all $x_i = t_i$ we have $x_i \notin \mathrm{vars}(t_i)$ and $x_i \notin \mathrm{vars}(E \setminus \{x_i = t_i\})$, so $\{x_1 = t_1, \ldots, x_n = t_n\}\{x_1 \mapsto t_1, \ldots, x_n \mapsto t_n\} = \{t_1 = t_1, \ldots, t_n = t_n\}$ and hence $\{x_1 \mapsto t_1, \ldots, x_n \mapsto t_n\}$ is an mgu of $\{x_1 = t_1, \ldots, x_n = t_n\}$. By 3. it is also an mgu of s, t. □

Example 3.7.3 (Size of Standard Unification Problems)**.** Any normal form of the unification problem E given by
$\{f(x_1, g(x_1, x_1), x_3, \ldots, g(x_n, x_n)) = f(g(x_0, x_0), x_2, g(x_2, x_2), \ldots, x_{n+1})\}$
with respect to $\Rightarrow_{\mathrm{SU}}$ is exponentially larger than E.

The second calculus, polynomial unification, prevents the problem of exponential growth by introducing an implicit representation for the mgu. For this calculus the size of a normal form is always polynomial in the size of the input unification problem.

Tautology $E \uplus \{t = t\} \Rightarrow_{\mathrm{PU}} E$

Decomposition $E \uplus \{f(s_1, \ldots, s_n) = f(t_1, \ldots, t_n)\} \Rightarrow_{\mathrm{PU}} E \uplus \{s_1 = t_1, \ldots, s_n = t_n\}$

Clash $E \uplus \{f(t_1, \ldots, t_n) = g(s_1, \ldots, s_m)\} \Rightarrow_{\mathrm{PU}} \bot$
if $f \neq g$

Occurs Check $E \uplus \{x = t\} \Rightarrow_{\mathrm{PU}} \bot$
if $x \neq t$ and $x \in vars(t)$

Orient $E \uplus \{t = x\} \Rightarrow_{\mathrm{PU}} E \uplus \{x = t\}$
if $t \notin \mathcal{X}$

Substitution $E \uplus \{x = y\} \Rightarrow_{\mathrm{PU}} E\{x \mapsto y\} \uplus \{x = y\}$
if $x \in vars(E)$ and $x \neq y$

Cycle $E \uplus \{x_1 = t_1, \ldots, x_n = t_n\} \Rightarrow_{\mathrm{PU}} \bot$
if there are positions p_i with $t_i|_{p_i} = x_{i+1}, t_n|_{p_n} = x_1$ and some $p_i \neq \epsilon$

Merge $E \uplus \{x = t, x = s\} \Rightarrow_{\mathrm{PU}} E \uplus \{x = t, t = s\}$
if $t, s \notin \mathcal{X}$ and $|t| \leq |s|$

Theorem 3.7.4 (Soundness, Completeness and Termination of $\Rightarrow_{\mathrm{PU}}$)**.** If s, t are two terms with $\mathrm{sort}(s) = \mathrm{sort}(t)$ then

1. if $\{s = t\} \Rightarrow^*_{\mathrm{PU}} E$ then any equation $(s' = t') \in E$ is well-sorted, i.e., $\mathrm{sort}(s') = \mathrm{sort}(t')$.
2. $\Rightarrow_{\mathrm{PU}}$ terminates on $\{s = t\}$.
3. if $\{s = t\} \Rightarrow^*_{\mathrm{PU}} E$ then σ is a unifier (mgu) of E iff σ is a unifier (mgu) of $\{s = t\}$.
4. if $\{s = t\} \Rightarrow^*_{\mathrm{PU}} \bot$ then s and t are not unifiable.

Theorem 3.7.5 (Normal Forms generated by $\Rightarrow_{\mathrm{PU}}$)**.** Let $\{s = t\} \Rightarrow^*_{\mathrm{PU}} \{x_1 = t_1, \ldots, x_n = t_n\}$ be a normal form. Then

1. $x_i \neq x_j$ for all $i \neq j$ and without loss of generality $x_i \notin \mathrm{vars}(t_{i+k})$ for all i, k, $1 \leq i < n$, $i + k \leq n$.
2. the substitution $\{x_1 \mapsto t_1\}\{x_2 \mapsto t_2\} \ldots \{x_n \mapsto t_n\}$ is an mgu of $s = t$.

Proof. 1. If $x_i = x_j$ for some $i \neq j$ then Merge is applicable. If $x_i \in \mathrm{vars}(t_i)$ for some i then Occurs Check is applicable. If the x_i cannot be ordered in the described way, then either Substitution or Cycle is applicable.
2. Since $x_i \notin \mathrm{vars}(t_{i+k})$ the composition yields the mgu. □

Lemma 3.7.6 (Size of Unifiers)**.** Let $\{s = t\}$ be a unification problem between two non-variable terms. Then

1. if s and t are linear then for any unifier σ and any term $r \in \text{codom}(\sigma)$, $|r| < |s|$ and $|r| < |t|$ as well as $\text{depth}(r) < \text{depth}(s)$ and $\text{depth}(r) < \text{depth}(t)$,

2. if s is shallow and linear, then the mgu σ of s and t is also a matcher from s to t, i.e., $s\sigma = t$

Proof. Both parts follow directly from the structure of the terms s, t: if they are both linear then the substitution rule is never applied. If s is shallow and linear, it has the form $f(x_1, \ldots, x_n)$, all x_i different, then the unifier is $\sigma = \{x_i \mapsto t|_i \mid 1 \leq i \leq n\}$. □

3.8 First-Order Free-Variable Tableau

An important disadvantage of standard first-order tableau is that the γ ground term instances need to be guessed. The main complexity in proving a formula to be valid lies in this guessing as for otherwise tableau terminates with a proof. Guessing useless ground terms may result in infinite branches. A natural idea is to guess ground terms that can eventually be used to close a branch. Of course, it is not known which ground term will close a branch. Therefore, it would be great to postpone the γ instantiations. This is the idea of free-variable first-order tableau. Instead of guessing a ground term for a γ formula, free-variable tableau introduces a fresh variable. Then a branch can be closed if two complementary literals have a common ground instance, i.e., their atoms are unifiable. The instantiation is delayed until a branch is closed for two literals via unification. As a consequence, for δ formulas no longer constants are introduced but shallow, so called *Skolem* terms in the formerly universally quantified variables that had the δ formula in their scope.

The new calculus needs to keep track of scopes of variables, so I move from a state as a set of pairs of a sequence and a set of constants, see standard first-order tableau Section 3.6, to a set of sequences of pairs (M_i, X_i) where X_i is a set of variables.

Definition 3.8.1 (Direct Free-Variable Tableau Descendant)**.** Given a γ- or δ-formula ϕ, Figure 3.2 shows its direct descendants.

The notion of closedness, Section 3.6, transfers exactly from standard to free-variable tableau. For α- and β-formulas the definition of an *open* formula remains unchanged as well. A γ- or δ-formula is called *open* in (M, X) if no direct descendant is contained in M. Note that instantiation of a tableau may remove direct descendants of γ- or δ-formulas by substituting terms for variables. Then a branch, pair (M, X), sequence M, is *open* if it is not closed and there is an open formula in M or there is pair of unifiable, complementary literals in M.

γ	Descendant $\gamma(y)$
$\forall x_S.\psi$	$\psi\{x_S \mapsto y_S\}$
$\neg\exists x_S.\psi$	$\neg\psi\{x_S \mapsto y_S\}$
	for a fresh variable y_S

δ	Descendant $\delta(f(y_1,\ldots,y_n))$
$\exists x_S.\psi$	$\psi\{x_S \mapsto f(y_1,\ldots,y_n)\}$
$\neg\forall x_S.\psi$	$\neg\psi\{x_S \mapsto f(y_1,\ldots,y_n)\}$
	for some fresh Skolem function f,
	$f : \text{sort}(y_1) \times \ldots \times \text{sort}(y_n) \to S$

Figure 3.2: γ- and δ-Formulas

γ-Expansion $N \uplus \{((\phi_1,\ldots,\psi,\ldots,\phi_n),X)\} \Rightarrow_{\text{FT}} N \cup \{((\phi_1,\ldots,\psi,\ldots,\phi_n,\psi'),X \cup \{y\})\}$

provided ψ is a γ-formula, ψ' a $\gamma(y)$ descendant where y is fresh to the overall tableau and the sequence is not closed.

δ-Expansion $N \uplus \{((\phi_1,\ldots,\psi,\ldots,\phi_n),X)\} \Rightarrow_{\text{FT}} N \cup \{(\phi_1,\ldots,\psi,\ldots,\phi_n,\psi'),X)\}$

provided ψ is an open δ-formula, $X = \{y_1,\ldots,y_n\}$, ψ' a $\delta(f(y_1,\ldots,y_n))$ descendant where f is fresh to the sequence, and the sequence is not closed.

Branch-Closing $N \uplus \{((\phi_1,\ldots,\phi_n),X)\} \Rightarrow_{\text{FT}} (N \cup \{((\phi_1,\ldots,\phi_n),X)\})\sigma$

provided there are complementary literals ϕ_i and ϕ_j, $\text{atom}(\phi_i)\sigma = \text{atom}(\phi_j)\sigma$ for an mgu σ, and the sequence is not closed.

The first-order free-variable tableau calculus consists of the rules α-, and β-expansion, see Section 3.6, which are adapted to pairs of sequences and variable sets, and the above three rules γ-Expansion, δ-Expansion and Branch-Closing. It remains to define the instantiation of a tableau by a substitution. As usual the application of a substitution to a set means application to the elements. For a pair $((\phi_1,\ldots,\phi_n),X)$ it is defined by $((\phi_1,\ldots,\phi_n),X)\sigma := ((\phi_1\sigma,\ldots,\phi_n\sigma),X \setminus \text{dom}(\sigma))$.

For free-varianle tableau, the γ rule has to be applied several times to the same formula as well in order to close a tableau, see the below example in Section 3.6. Constructing a closed tableau from initial state

$$\{((\forall x_S.(P(x_S) \to P(f(x_S))),\ P(b),\ \neg P(f(f(b)))),\emptyset)\}$$

is impossible without applying γ-Expansion twice to $\forall x_S.(P(x_S) \to P(f(x_S)))$ on some branch, where $b :\to S$, $f : S \to S$ and $P \subseteq S$. Below is the derivation of a closed tableau where I only show the added formulas and often abbreviate the parent sequence with an indexed M.

$\{((\forall x_S.(P(x_S) \to P(f(x_S))),\ P(b),\ \neg P(f(f(b)))), \emptyset)$
$\Rightarrow^{\gamma}_{\mathrm{FT}}$ $\{((M_1, P(y_S) \to P(f(y_S))), \{y_s\})$
$\Rightarrow^{\beta}_{\mathrm{FT}}$ $\{((M_2, \neg P(y_S)), \{y_s\}), ((M_2, P(f(y_S))), \{y_s\})\}$
$\Rightarrow^{\mathrm{Closing}}_{\mathrm{FT}}$ $\{((M_2\sigma, \neg P(b)), \emptyset), ((M_2\sigma, P(f(b))), \emptyset)\}$
the unifier is $\sigma = \{y_S \mapsto b\}$, of the literals $\neg P(y_S)$ and $P(b)$
$\Rightarrow^{\gamma}_{\mathrm{FT}}$ $\{((M_2\sigma, P(f(b)), P(f(z_S)) \to P(f(f(z_S)))), \{z_S\})\}$
$\Rightarrow^{\beta}_{\mathrm{FT}}$ $\{((M_3, \neg P(f(z_S))), \{z_S\}), ((M_3, P(f(f(z_S)))), \{z_S\})\}$
$\Rightarrow^{\mathrm{Closing}}_{\mathrm{FT}}$ $\{((M_3\delta, \neg P(f(b))), \emptyset), ((M_3\delta, P(f(f(b)))), \emptyset)\}$
the unifier is $\delta = \{z_S \mapsto f(b)\}$, of the literals $\neg P(z_S)$ and $P(f(b))$
now the tableau is closed

where $M_1 = (\forall x_S.(P(x_S) \to P(f(x_S))),\ P(b),\ \neg P(f(f(b))))$, $M_2 = M_1, (P(y_S) \to P(f(y_S)))$ and $M_3 = M_2\sigma, (P(f(b)), P(f(z_S)) \to P(f(f(z_S))))$.

A possibly infinite tableau derivation $s_0 \Rightarrow_{\mathrm{FT}} s_1 \Rightarrow_{\mathrm{FT}} \ldots$ is called *saturated* if for all its open sequences M_i of some pair $(M_i, X_i) \in s_i$ where not all successor sequences of M_i are closed and all formulas ϕ occurring in M_i, there is an index $j > i$ and some pair $(M_j, X_j) \in s_j$, M_i is a prefix of M_j, if in case ϕ is an α-formula then both direct descendants are part of M_j, if it is a β-formula then one of its descendants is part of M_j, if it is a δ- or γ-formula then one direct descendant is part of M_j, and if Branch-Closing is applicable to M_i then M_j is closed.

Theorem 3.8.2 (Free-variable First-Order Tableau is Sound and Complete)**.** A formula ϕ is valid iff free-variable tableau computes a closed state out of $\{(\neg\phi, \emptyset)\}$.

Proof Idea: By lifting from standard first-order tableau. □

Here is another example including δ-Expansion applications. I assume the exsitence of exactly one sort with the respective defintions for the constants, functions, variables, and predicates.

$\{((\neg[\exists w \forall x R(x, w, f(x, w)) \to \exists w \forall x \exists y R(x, w, y)]), \emptyset)$
$\Rightarrow^{\alpha,*}_{\mathrm{FT}}$ $\{((M_1, \exists w \forall x\ R(x, w, f(x, w))), \neg\exists w \forall x \exists y\ R(x, w, y)), \emptyset)\}$
$\Rightarrow^{\delta}_{\mathrm{FT}}$ $\{((M_2, \forall x\ R(x, c, f(x, c))), \emptyset)\}$
$\Rightarrow^{\gamma}_{\mathrm{FT}}$ $\{((M_2, \forall x\ R(x, c, f(x, c)), \neg\forall x \exists y\ R(x, v_1, y)), \{v_1\})\}$
$\Rightarrow^{\delta}_{\mathrm{FT}}$ $\{((M_2, \forall x\ R(x, c, f(x, c)), \neg\forall x \exists y\ R(x, v_1, y), \forall x\ R(x, c, f(x, c)), \neg\exists y\ R(g(v_1), v_1, y)), \{v_1\})\}$
$\Rightarrow^{\gamma}_{\mathrm{FT}}$ $\{((M_3, R(v_2, c, f(v_2, c))), \{v_1, v_2\})\}$
$\Rightarrow^{\gamma}_{\mathrm{FT}}$ $\{((M_3, R(v_2, c, f(v_2, c)), \neg R(g(v_1), v_1, v_3)), \{v_1, v_2, v_3\})\}$
$\Rightarrow^{\mathrm{Closing}}_{\mathrm{FT}}$ $\{((M_3\sigma, R(g(c), c, f(g(c), c)), \neg R(g(c), c, f(g(c), c))), \emptyset)\}$
the unifier is $\sigma = \{v_1 \mapsto c,\ v_2 \mapsto g(c),\ v_3 \mapsto f(g(c), c)\}$
now the tableau is closed

where $M_1 = \neg[\exists w \forall x R(x, w, f(x, w)) \rightarrow \exists w \forall x \exists y R(x, w, y)]$,
$M_2 = M_1, \exists w \forall x\ R(x, w, f(x, w)), \neg\exists w \forall x \exists y\ R(x, w, y)$, and
$M_3 = M_2, \forall x\ R(x, c, f(x, c)), \neg\forall x \exists y\ R(x, v_1, y), \forall x\ R(x, c, f(x, c)), \neg\exists y\ R(g(v_1), v_1, y)$.

Semantic Tableau vs. Resolution

1. Tableau: global, goal-oriented, "backward".
2. Resolution: local, "forward".
3. 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.)
4. Resolution can be combined with more powerful redundancy elimination methods; because of its global nature this is more difficult for the tableau method.
5. Resolution can be refined to work well with equality; for tableau this seems to be impossible.
6. On the other hand tableau calculi can be easily extended to other logics; in particular tableau provers are very successful in modal and description logics.

3.9 First-Order CNF Transformation

Similar to the propositional case, first-order resolution and superposition operate on clauses. In this section I show how any first-order sentence can be efficiently transformed into a CNF, preserving satisfiability. To this end all existentially quantified variables are replaced with so called Skolem functions. Similar to the renaming of subformulas this replacement preserves satisfiability only. Eventually, all variables in clauses are implicitly universally quantified.

More concretely, the acnf CNF transformation is algorithm from Section 2.5.3 is generalized to first-order logic with equality. The adiitional complications are: (i) additional rules for the quantifiers, (ii) the formula renaming technique is extended to cope with variables and (iii) removal of existential quantifiers through the introduction of *Skolem* functions. Basically, all rules known from the propositional case apply.

The first two extra rules eliminate $\top$ and $\bot$ from first-order formula starting with a quantifier.

ElimTB13 $\chi[\{\forall, \exists\}x.\top]_p \Rightarrow_{\text{ACNF}} \chi[\top]_p$

ElimTB14 $\chi[\{\forall,\exists\}x.\bot]_p \Rightarrow_{\text{ACNF}} \chi[\bot]_p$

Next, in order to obtain a negation normal form with negation symbols in front of atoms only, the respective rules for pushing negations over the quantifiers are needed as well.

PushNeg4 $\chi[\neg\forall x.\phi]_p \Rightarrow_{\text{ACNF}} \chi[\exists x.\neg\phi]_p$

PushNeg5 $\chi[\neg\exists x.\phi]_p \Rightarrow_{\text{ACNF}} \chi[\forall x.\neg\phi]_p$

where the expression $\{\forall,\exists\}x.\phi$ covers both cases $\forall x.\phi$ and $\exists x.\phi$. The next step is to rename all variables such that different quantifiers bind different variables. This step is necessary to prevent a later on confusion of variables, once the quantifiers are dropped.

RenVar $\phi \Rightarrow_{\text{ACNF}} \phi\sigma$
for $\sigma = \{\}$

In first-order logic, the renaming of subformulas has to take care of variables as well. The notion of an obvious position remains unchanged. Therefore, the basic mechanism of renaming and the concept of a beneficial subformula is exactly the same as in propositional logic. The only difference is that renaming does introduce an atom in the free variables of the respective subformula. When some formula ψ is renamed at position p an atom $P(\vec{x}_n)$, $\vec{x}_n = x_1,\ldots,x_n$ replaces $\psi|_p$ where $\text{fvars}(\psi|_p) = \{x_1\ldots,x_n\}$. The respective definition of $P(\vec{x}_n)$ becomes

$$\text{def}(\psi,p,P(\vec{x}_n)) := \begin{cases} \forall\vec{x}_n.(P(\vec{x}_n) \rightarrow \psi|_p) & \text{if } \text{pol}(\psi,p) = 1 \\ \forall\vec{x}_n.(\psi|_p \rightarrow P(\vec{x}_n)) & \text{if } \text{pol}(\psi,p) = -1 \\ \forall\vec{x}_n.(P(\vec{x}_n) \leftrightarrow \psi|_p) & \text{if } \text{pol}(\psi,p) = 0 \end{cases}$$

and the rule SimpleRenaming is changed accordingly.

SimpleRenaming $\phi \Rightarrow_{\text{ACNF}} \phi[P_1(\vec{x_{1,j_1}})]_{p_1}[P_2(\vec{x_{2,j_2}})]_{p_2}\ldots[P_n(\vec{x_{n,j_n}})]_{p_n} \wedge$
$\text{def}(\phi,p_1,P_1(\vec{x_{1,j_1}})) \wedge \ldots \wedge \text{def}(\phi[P_1(\vec{x_{1,j_1}})]_{p_1}[P_2(\vec{x_{2,j_2}})]_{p_2}\ldots[P_{n-1}(\vec{x_{n-1,j_{n-1}}})]_{p_{n-1}},p_n,P_n(\vec{x_{n,j_n}}))$
provided $\{p_1,\ldots,p_n\} \subset \text{pos}(\phi)$ and for all $i, i+j$ either $p_i \parallel p_{i+j}$ or $p_i > p_{i+j}$
and where $\text{fvars}(\phi|_{p_i}) = \{x_{i,1},\ldots,x_{i,j_i}\}$ and all P_i are different and new to ϕ

SimpleRenaming shares the variables of ϕ with the variables used for the definitions of the new predicates. This does not cause any confusion, because there will never be a clause consisting of literals from the remaning ϕ after renaming and literals from a definition. In propositional logic after subformula renaming, removal of equivalences and implications, and pushing negations down in front

of atoms, the CNF can be generated using distributivity. In first-order logic the existential quantifiers are eliminated first by the introduction of Skolem functions. In order to receive Skolem functions with few arguments, the quantifiers are first moved inwards as far as passible. This step is called *mini-scoping*.

MiniScope1 $\chi[\forall x.(\psi_1 \circ \psi_2)]_p \Rightarrow_{\text{ACNF}} \chi[(\forall x.\psi_1) \circ \psi_2]_p$
provided $\circ \in \{\wedge, \vee\}$, $x \notin \text{fvars}(\psi_2)$

MiniScope2 $\chi[\exists x.(\psi_1 \circ \psi_2)]_p \Rightarrow_{\text{ACNF}} \chi[(\exists x.\psi_1) \circ \psi_2]_p$
provided $\circ \in \{\wedge, \vee\}$, $x \notin \text{fvars}(\psi_2)$

MiniScope3 $\chi[\forall x.(\psi_1 \wedge \psi_2)]_p \Rightarrow_{\text{ACNF}} \chi[(\forall x.\psi_1) \wedge (\forall x.\psi_2)\sigma]_p$
where $\sigma = \{\}$, $x \in (\text{fvars}(\psi_1) \cap \text{fvars}(\psi_2))$

MiniScope4 $\chi[\exists x.(\psi_1 \vee \psi_2)]_p \Rightarrow_{\text{ACNF}} \chi[(\exists x.\psi_1) \vee (\exists x.\psi_2)\sigma]_p$
where $\sigma = \{\}, x \in (\text{fvars}(\psi_1) \cap \text{fvars}(\psi_2))$

The rules MiniScope1, MiniScope2 are applied modulo the commutativity of $\wedge$, $\vee$. Once the quantifiers are moved inwards Skolemization can take place. Skolemization replaces all existentially quantified variables by shallow Skolem function terms.

Skolemization $\chi[\exists x.\phi]_p \Rightarrow_{\text{ACNF}} \chi[\phi\{x \mapsto f(y_1, \ldots, y_n)\}]_p$
provided there is no q, $q < p$ with $\phi|_q = \exists x'.\psi'$, $\text{fvars}(\exists x.\psi) = \{y_1, \ldots, y_n\}$, $f : \text{sort}(y_1) \times \ldots \times \text{sort}(y_n) \to \text{sort}(x)$ is a new function symbol

Theorem 3.9.1 (Skolemization Preserves Satisfiability)**.** A formula $\chi[\exists x.\phi]_p$ is satisfiable iff the formula $\chi[\phi\{x \mapsto f(y_1, \ldots, y_n)\}]_p$ is, where χ is in negation normal form, p the maximal position of an existential quantifier, $\text{fvars}(\exists x.\psi) = \{y_1, \ldots, y_n\}$, and $\text{arity}(f) = n$ is a new function symbol to ϕ, $f : \text{sort}(y_1) \times \ldots \times \text{sort}(y_n) \to \text{sort}(x)$.

Proof. Both directions of the proof are done by induction on the length of p and then by a case analysis on the structure of the formula. I only show the relevant cases.

$\Rightarrow$: If $\mathcal{A}, \beta \models \exists x.\phi$ then there exists an $a \in (\text{sort}(x))^{\mathcal{A}}$ such that $\mathcal{A}, \beta[x \mapsto a] \models \phi$. Now define $f^{\mathcal{A}}(\beta(y_1), \ldots, \beta(y_n)) := a$. Then obviously $\mathcal{A}, \beta \models \chi[\phi\{x \mapsto f(y_1, \ldots, y_n)\}]_p$. The function $f^{\mathcal{A}}$ is well-defined, because the truth value of $\exists x.\phi$ under $\mathcal{A}, \beta$ depends only on the values β assigns to the free variables of $\exists x.\phi$, i.e., the free variables $\text{fvars}(\exists x.\psi) = \{y_1, \ldots, y_n\}$ the function f depends on.

$\Leftarrow$: If $\mathcal{A}, \beta \models \chi[\phi\{x \mapsto f(y_1, \ldots, y_n)\}]_p$ then $\mathcal{A}, \beta[x \mapsto f^{\mathcal{A}}(\beta(y_1), \ldots, \beta(y_n))] \models \phi$ and therefore $\mathcal{A}, \beta \models \exists x.\phi$. □

Example 3.9.2 (Mini-Scoping and Skolemization)**.** Consider the simple formula $\forall x.\exists y.(R(x,x) \wedge P(y)$. Applying Skolemization directly to this formula, without mini-scoping results in

$$\forall x.\exists y.(R(x,x) \wedge P(y)) \Rightarrow^{\text{Skolem}}_{\text{ACNF}} \forall x.(R(x,x) \wedge P(g(x))$$

for a unary Skolem function g because $\text{fvars}(\exists y.(R(x,x) \wedge P(y))) = \{x\}$. Applying mini-scoping and then Skolemization generates

$$\begin{array}{lll} \forall x.\exists y.(R(x,x) \wedge P(y)) & \Rightarrow^{\text{MiniScope},*}_{\text{ACNF}} & \forall x.R(x,x) \wedge \exists y.P(y) \\ & \Rightarrow^{\text{Skolem}}_{\text{ACNF}} & \forall x.R(x,x) \wedge P(a) \end{array}$$

for some Skolem constant $a :\rightarrow \text{sort}(y)$ because $\text{fvars}(\exists y.P(y)) = \emptyset$. Now the former formula after Skolemization is seriously more complex than the latter. The former belongs to an undecidable fragment of first-order logic while the latter belongs to a decidable one (see Section 3.15).

Finally, the universal quantifiers are removed. In a first-order logic CNF any variable is universally quantified by default. Furthermore, the variables of two different clauses are always assumed to be different.

RemForall $\quad \chi[\forall x.\psi]_p \Rightarrow_{\text{ACNF}} \chi[\psi]_p$

The actual CNF is then done by distributivity, exactly as it is done in propositional logic.

Algorithm 11: acnf(ϕ)

Input : A first-order formula ϕ.
Output: A formula ψ in CNF satisfiability preserving to ϕ.

```
1  whilerule (ElimTB1(φ),...,ElimTB14(φ)) do ;
2  RenVar(φ);
3  SimpleRenaming(φ) on obvious positions;
4  whilerule (ElimEquiv1(φ),ElimEquiv2(φ)) do ;
5  whilerule (ElimImp(φ)) do ;
6  whilerule (PushNeg1(φ),...,PushNeg5(φ)) do ;
7  whilerule (MiniScope1(φ),...,MiniScope4(φ)) do ;
8  whilerule (Skolemization(φ)) do ;
9  whilerule (RemForall(φ)) do ;
10 whilerule (PushDisj(φ)) do ;
11 return φ;
```

Theorem 3.9.3 (Properties of the ACNF Transformation)**.** Let ϕ be a first-order sentence, then

1. $\text{acnf}(\phi)$ terminates

2. ϕ is satisfiable iff acnf(ϕ) is satisfiable

Proof. (Idea) 1. is a straightforward extension of the propositional case. It is easy to define a measure for any line of Algorithm 11.
2. can also be established separately for all rule applications. The rules SimpleRenaming and Skolemization need separate proofs, the rest is straightforward or copied from the propositional case. □

In addition to the consideration of repeated subformulas, discussed in Section 2.5, for first-order renaming another technique can pay off: generalization. Consider the formula $[\phi_1 \vee (Q_1(a_1) \wedge Q_2(a_1))] \wedge [\phi_2 \vee (Q_1(a_2) \wedge Q_2(a_2))] \wedge \ldots \wedge [\phi_n \vee (Q_1(a_n) \wedge Q_2(a_n)]$. SimpleRenaming on obvious renamings applied to this formula will independently rename any occurrences of a formula $(Q_1(a_i) \wedge Q_2(a_i))$. However generalization pays off here. By adding the definition $\forall x, y\,(R(x,y) \rightarrow (Q_1(x) \wedge Q_2(y)))$ and replacing the i^{th} occurrence of the conjunct by $R(x,y)\{x \mapsto a_i, y \mapsto a_i\}$ one definition for all subformula occurrences suffices.

3.10 First-Order Resolution

As already mentioned, I still consider first-order logic without equality. First-order resolution on ground clauses corresponds to propositional resolution. Each ground atom becomes a propositional variable. However, since there are up to infinitely many ground instances for a first-oder clause set with variables and it is not a priori known which ground instances are needed in a proof, the first-order resolution calculus operates on clauses with variables. Roughly, the relationship between ground resolution and first-order resolution corresponds to the relationship between standard tableau and free-variable tableau. However, the variables in free-variable tablea can only be instantiated once, thereas in resolution they can be instantiated arbitrarily often.

Propositional (or first-order ground) resolution is refutationally complete, without reduction rules it is not guaranteed to terminate for satisfiable sets of clauses, and inferior to the CDCL calculus. However, in contrast to the CDCL calculus, resolution can be easily extended to non-ground clauses via unification. The problem to lift the CDCL calculus lies in the lifting of the model representation of the trail. I'll discuss this in more detail in Section 3.15.

Lemma 3.10.1. Let $\mathcal{A}$ be a $\Sigma - algebra$ and let ϕ be a $\Sigma - formula$ with free variables $x_1, \ldots, x_n$. Then $\mathcal{A} \models \forall x_1, \ldots, x_n \phi$ iff $\mathcal{A} \models \phi$

Lemma 3.10.2. Let ϕ be a $\Sigma - formula$ with free variables $x_1, \ldots, x_n$, let σ be a substitution and let $y_1, \ldots, y_m$ be free variables of $\phi\sigma$. Then $\mathcal{A} \models \forall x_1, \ldots, x_n \phi$ implies $\mathcal{A} \models \forall y_1, \ldots, y_m \phi\sigma$.

In particular, if $\mathcal{A}$ is a model of an (implicitly universally quantified) clause C then it is also a model of all (implicitly universally quantified) instances $C\sigma$ of C. Consequently, if it is shown that some instances of clauses in a set N are unsatisfiable then it is also shown that N itself is unsatisfiable.

General Resolution through Instantiation

The approach is to instantiate clauses appropriately. An example is shown in Figure 3.3. However, this may lead to several problems. First of all, more than one instance of a clause can participate in a proof and secondly, which is even worse, there are infinitely many possible instances. Due to the fact that instantiation must produce complementary literals so that inferences become possible, the idea is to not instantiate more than necessary to get complementary literals. An instantiation of the clause set from Figure 3.3 is again shown in Figure 3.4 with the difference that the latter instantiates only as much as necessary, inevitably reducing the number of substitutions.

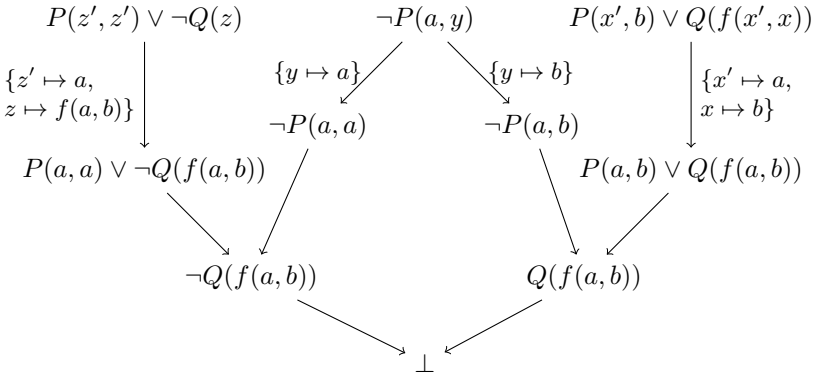

Figure 3.3: Instantiation of the clause set $N = P(z', z') \vee \neg Q(z), \neg P(a, y), P(x', b) \vee Q(f(x', x))$

Lifting Principle

In order to overcome the problem of effectively and efficiently saturating infinite sets of clauses as they arise from taking the (ground) instances of finitely many *general* clauses (with variables), the general idea is to lift the resolution principle as proposed by Robinson [**?**]. The lifting is as follows: For the resolution of general clauses, *equality* of ground atoms is generalized to *unifiability* of general atoms and only the *most general* (minimal) unifiers (mgu) are computed.

The advantage of the method in Robinson [**?**] compared with Gilmore [**?**] is that unification enumerates only those instances of clauses that participate in an inference. Moreover, clauses are not right away instantiated into ground clauses. Rather they are instantiated only as far as required for an inference. Inferences with non-ground clauses in general represent infinite sets of ground

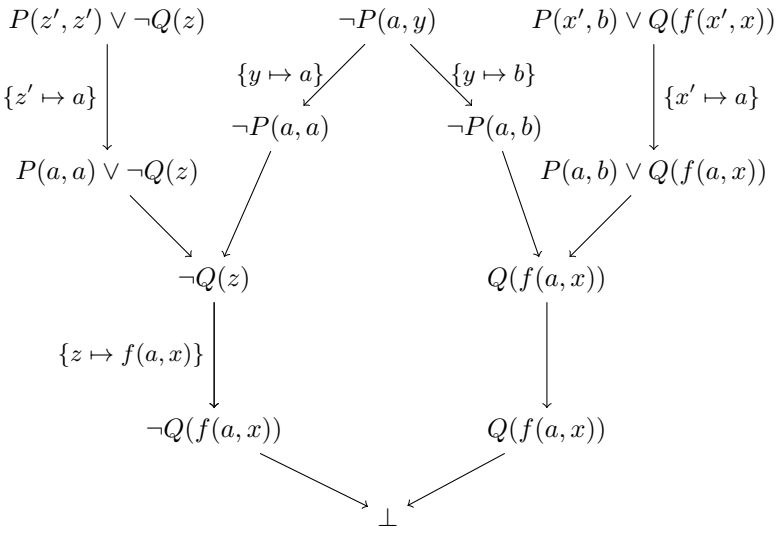

Figure 3.4: Instantiation of the clause set $N = P(z', z') \vee \neg Q(z), \neg P(a, y), P(x', b) \vee Q(f(x', x))$ with a reduced number of instantiations.

inferences which are computed simultaneously in a single step.

The *first-order resolution calculus* consists of the inference rules *Resolution* and *Factoring* and generalizes the propositional resolution calculus (Section 2.6). Variables in clauses are implicitely universally quantified, so they can be instantiated in an arbitrary way. For the application of any inference or reduction rule, I can therefore assume that the involved clauses don't share any variables, i.e., variables are a priori renamed. Furthermore, clauses are assumed to be unique with respect to renaming in a set.

Resolution $(N \uplus \{D \vee A, \neg B \vee C\}) \Rightarrow_{\text{RES}} (N \cup \{D \vee A, \neg B \vee C\} \cup \{(D \vee C)\sigma\})$
if $\sigma = mgu(A, B)$ for atoms A, B

Factoring $(N \uplus \{C \vee L \vee K\}) \Rightarrow_{\text{RES}} (N \cup \{C \vee L \vee K\} \cup \{(C \vee L)\sigma\})$
if $\sigma = mgu(L, K)$ for literals L, K

The reduction rules are

Subsumption $(N \uplus \{C_1, C_2\}) \Rightarrow_{\text{RES}} (N \cup \{C_1\})$
provided $C_1\sigma \subset C_2$ for some matcher σ

Tautology Deletion $(N \uplus \{C \vee A \vee \neg A\}) \Rightarrow_{\text{RES}} (N)$

Condensation $(N \uplus \{C\}) \Rightarrow_{\text{RES}} (N \cup \{C'\})$
where C' is the result of removing duplicate literals from $C\sigma$ for some matcher σ and C' subsumes C

Subsumption Resolution $(N \uplus \{C_1 \vee L, C_2 \vee K\}) \Rightarrow_{\text{RES}} (N \cup \{C_1 \vee L, C_2\})$
where $L\sigma = \text{comp}(K)$ and $C_1\sigma \subseteq C_2$

Lifting Lemma

Lemma 3.10.3. Let C and D be variable-disjoint clauses. If

Propositional Resolution $(N \uplus \{D\sigma, C\rho\}) \Rightarrow (N \cup \{D\sigma, C\rho\} \cup \{C'\})$
where σ and ρ are substitutions then there exists a substitution τ so that

General Resolution $(N \uplus \{D, C\}) \Rightarrow (N \cup \{D, C\} \cup \{C''\tau = C'\})$
An analogous lifting lemma holds for factorization.

Saturation of Sets of General Clauses

Definition 3.10.4 (Resolution Saturation)**.** A set of clauses N is saturated up to redundancy

Corollary 3.10.5. Let N be a set of general clauses saturated under Res, i.e., $\text{Res}(N) \subseteq N$. Then also $G_\Sigma(N)$ is saturated, that is, $\text{Res}(G_\Sigma(N)) \subseteq G_\Sigma(N)$.

Proof. W.l.o.g. assume that clauses in N are pairwise variable-disjoint. (Otherwise they have to be made disjoint and this renaming process changes neither $Res(N)$ nor $G_\Sigma(N)$.) Let $C' \in Res(G_\Sigma(N))$, meaning (i) there exist resolvable ground instances $D\sigma$ and $C\rho$ of N with resolvent C', or else (ii) C' is a factor of a ground instance $C\sigma$ of C.

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

Case (ii): Similar. □

Herbrand's Theorem

Lemma 3.10.6. 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 3.10.7. Let N be a set of Σ-clauses, let $\mathcal{A}$ be a *Herbrand* interpretation. Then $\mathcal{A} \models G_\Sigma(N)$ implies $\mathcal{A} \models N$.

Theorem 3.10.8 (Herbrand)**.** A set N of Σ-clauses is satisfiable if and only if it has a Herbrand model over Σ.

Proof. ($\Leftarrow$) Assume N has a Herbrand model I over Σ, i.e., $I \models N$. Then N is satisfiable.

($\Rightarrow$) Let $N \not\models \bot$.

$$\begin{array}{lcll} N \not\models \bot & \Rightarrow & \bot \notin Res^*(N) & \text{(resolution is sound)} \\ & \Rightarrow & \bot \notin G_\Sigma(Res^*(N)) & \\ & \Rightarrow & I_{G_\Sigma(Res^*(N))} \models G_\Sigma(Res^*(N)) & \text{(Theorem ; Corollary 3.10.5)} \\ & \Rightarrow & I_{G_\Sigma(Res^*(N))} \models Res^*(N) & \text{(Lemma 3.10.7)} \\ & \Rightarrow & I_{G_\Sigma(Res^*(N))} \models N & (N \subseteq Res^*(N)) \end{array}$$

□

The Theorem of Löwenheim-Skolem

Theorem 3.10.9 (Löwenheim–Skolem)**.** Let Σ be a countable signature and let S be a set of closed Σ-formulas. Then S is satisfiable iff S has a model over a countable universe.

Proof. If both X and Σ are countable, then S can be at most countably infinite. Now generate, maintaining satisfiability, a set N of clauses from S. This extends Σ by at most countably many new Skolem functions to Σ'. As Σ' is countable, so is $T_{\Sigma'}$, the universe of Herbrand-interpretations over Σ'. Now apply Theorem 3.10.8. □

Refutational Completeness of General Resolution

Theorem 3.10.10 (Soundness and Completenss of Resolution)**.** The resolution calculus is sound and complete:

$$N \text{ is unsatisfiable iff } N \Rightarrow^*_{\text{RES}} N' \text{ and } \bot \in N' \text{ for some } N'$$

Theorem 3.10.11 (Soundness and Completenss of Resolution)**.** Let N be a set of first-clauses where $Res(N) \subseteq N$. Then

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

Proof. Let $Res(N) \subseteq N$. By Corollary 3.10.5: $Res(G_\Sigma(N)) \subseteq G_\Sigma(N)$

$$\begin{array}{lll} N \models \bot & \Leftrightarrow G_\Sigma(N) \models \bot & \text{(Lemma 3.10.6/3.10.7; Theorem 3.10.8)} \\ & \Leftrightarrow \bot \in G_\Sigma(N) & \text{(propositional resolution sound and complete)} \\ & \Leftrightarrow \bot \in N & \end{array}$$

□

Compactness of First-Order Logic

Theorem 3.10.12 (Compactness Theorem for First-Order Logic)**.** Let S be a set of first-order formulas. S is unsatisfiable if and only if some finite subset $S' \subseteq S$ is unsatisfiable.

Proof. ($\Leftarrow$) Assume that S' is unsatisfiable. Then there exists at least one unsatisfiable formula $\phi \in S'$. Since $S' \subseteq S$, S is also unsatisfiable.

($\Rightarrow$) 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 3.10.11, $\bot \in Res^*(N)$, 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 so that the corresponding clauses contain the assumptions (leaves) of B. □

3.11 Orderings

Propositional superposition is based on an ordering on the propositional variables, Section 2.7. The ordering is total and well-founded. Basically, propositional variables correspond to ground atoms in first-order logic. This section generalizes the ideas of the propositional superposition ordering to first-order logic. In first-order logic the ordering has to also consider terms and variables and operations on terms like the application of a substitution.

Definition 3.11.1 (Σ-Operation Compatible Relation)**.** A binary relation $\sqsupset$ over $T(\Sigma, \mathcal{X})$ is called *compatible with Σ-operations,* if $s \sqsupset s'$ implies $f(t_1, \ldots, s, \ldots, t_n) \sqsupset f(t_1, \ldots, s', \ldots, t_n)$ for all $f \in \Omega$ and $s, s', t_i \in T(\Sigma, \mathcal{X})$.

Lemma 3.11.2 (Σ-Operation Compatible Relation)**.** A relation $\sqsupset$ is compatible with Σ-operations iff $s \sqsupset s'$ implies $t[s]_p \sqsupset t[s']_p$ for all $s, s', t \in T(\Sigma, \mathcal{X})$ and $p \in pos(t)$.

In the literature *compatible with Σ-operations* is sometimes also called *compatible with contexts.*

Definition 3.11.3 (Substitution Stable Relation, Rewrite Relation)**.** A binary relation $\sqsupset$ over $T(\Sigma, \mathcal{X})$ is called *stable under substitutions*, if $s \sqsupset s'$ implies $s\sigma \sqsupset s'\sigma$ for all $s, s' \in T(\Sigma, \mathcal{X})$ and substitutions σ. A binary relation $\sqsupset$ is called a *rewrite relation*, if it is compatible with Σ-operations and stable under substitutions.

A *rewrite ordering* is then an ordering that is a rewrite relation.

Definition 3.11.4 (Subterm Ordering)**.** The *proper subterm ordering* $s > t$ is defined by $s > t$ iff $s|_p = t$ for some position $p \neq \epsilon$ of s.

Definition 3.11.5 (Simplification Ordering)**.** A rewrite ordering $\succ$ over $T(\Sigma, \mathcal{X})$ is called *simplification ordering*, if it enjoys the *subterm property* $s \succ t$ implies $s > t$ for all $s, t \in T(\Sigma, \mathcal{X})$ of the same sort.

Definition 3.11.6 (Lexicographical Path Ordering (LPO))**.** Let $\Sigma = (\mathcal{S}, \Omega, \Pi)$ be a signature and let $\succ$ be a strict partial ordering on operator symbols in Ω, called *precedence*. The *lexicographical path ordering* $\succ_{lpo}$ on $T(\Sigma, \mathcal{X})$ is defined as follows: if s, t are terms in $T_S(\Sigma, \mathcal{X})$ then $s \succ_{lpo} t$ iff

1. $t = x \in \mathcal{X}$, $x \in vars(s)$ and $s \neq t$ or
2. $s = f(s_1, \ldots, s_n)$, $t = g(t_1, \ldots, t_m)$ and
   (a) $s_i \succeq_{lpo} t$ for some $i \in \{1, \ldots, n\}$ or
   (b) $f \succ g$ and $s \succ_{lpo} t_j$ for every $j \in \{1, \ldots, m\}$ or
   (c) $f = g$, $s \succ_{lpo} t_j$ for every $j \in \{1, \ldots, m\}$ and $(s_1, \ldots, s_n)(\succ_{lpo})_{lex}(t_1, \ldots, t_m)$.

Theorem 3.11.7 (LPO Properties)**.**
1. The LPO is a rewrite ordering.
2. LPO enjoys the subterm property, hence is a simplification ordering.
3. If the precedence $\succ$ is total on Ω then $\succ_{lpo}$ is total on the set of ground terms $T(\Sigma)$.
4. If Ω is finite then $\succ_{lpo}$ is well-founded.

Example 3.11.8. Consider the terms $g(x)$, $g(y)$, $g(g(a))$, $g(b)$, $g(a)$, b, a. With respect to the precedence $g \succ b \succ a$ the ordering on the ground terms is $g(g(a)) \succ_{lpo} g(b) \succ_{lpo} g(a) \succ_{lpo} b \succ_{lpo} a$. The terms $g(x)$ and $g(y)$ are not comparable. Note that the terms $g(g(a))$, $g(b)$, $g(a)$ are all instances of both $g(x)$ and $g(y)$.

With respect to the precedence $b \succ a \succ g$ the ordering on the ground terms is $g(b) \succ_{lpo} b \succ_{lpo} g(g(a)) \succ_{lpo} g(a) \succ_{lpo} a$.

Definition 3.11.9 (The Knuth-Bendix Ordering)**.** Let $\Sigma = (\mathcal{S}, \Omega, \Pi)$ be a finite signature, let $\succ$ be a strict partial ordering *("precedence")* on Ω, let $w : \Omega \cup \mathcal{X} \to \mathbb{R}^+$ be a *weight function*, so that the following admissibility condition is satisfied: $w(x) = w_0 \in \mathbb{R}^+$ for all variables $x \in \mathcal{X}$; $w(c) \geq w_0$ for all constants $c \in \Omega$.

Then, 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

$$\sum w(t) = \sum_{x \in vars(t)} w(x) \cdot \#(x, t) + \sum_{f \in \Omega} w(f) \cdot \#(f, t)$$

where $\#(a, t)$ is the number of occurrences of a in t.

The *Knuth-Bendix ordering* $\succ_{kbo}$ on $T(\Sigma, \mathcal{X})$ induced by $\succ$ and admissible w is defined by: $s \succ_{kbo} t$ iff

1. $\#(x, s) \geq \#(x, t)$ for all variables x and $w(s) > w(t)$, or
2. $\#(x, s) \geq \#(x, t)$ for all variables x, $w(s) = w(t)$, and
   (a) $s = f(s_1, \ldots, s_m)$, $t = g(t_1, \ldots, t_n)$, and $f \succ g$, or

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

Theorem 3.11.10 (KBO Properties)**.**

1. The KBO is a rewrite ordering.
2. KBO enjoys the subterm property, hence is a simplification ordering.
3. If the precedence $\succ$ is total on Ω then $\succ_{kbo}$ is total on the set of ground terms $T(\Sigma)$.
4. If Ω is finite then $\succ_{kbo}$ is well-founded.

The KBO ordering can be extended to contain unary function symbols with weight zero. This was motivated by completion of the group axioms, see Chapter 4.

Definition 3.11.11 (The Knuth-Bendix Ordering Extended)**.** The additional requirements added to Definition 3.11.9 are

1. Extend w to $w : \Omega \cup \mathcal{X} \to \mathbb{R}_0^+$
2. If $w(f) = 0$ for some $f \in \Omega$ with arity$(f) = 1$, then $f \succeq g$ for all $g \in \Omega$.
3. As a first case to the disjunction of 3.11.9-2.
   (a') $t = x$, $s = f^n(x)$ for some $n \geq 1$

The LPO ordering as well as the KBO ordering can be extended to atoms in a straightforward way. The precedence $\succ$ is extended to Π. For LPO atoms are then compared according to Definition 3.11.6-2. For KBO the weight function w is also extended to atoms by giving predicates a non-zero positive weight and then atoms are compared according to terms.

Actually, since atoms are never substituted for variables in first-order logic, an alternative to the above would be to first compare the predicate symbols and let $\succ$ decide the ordering. Only if the atoms share the same predicate symbol, the argument terms are considered, e.g., in a lexicographic way and are then compared with respect to KBO or LPO, respectively.

3.12 First-Order Ground Superposition

Propositional clauses and ground clauses are essentially the same, as long as equational atoms are not considered. This section deals only with ground clauses and recalls mostly the material from Section 2.7 for first-order ground clauses. The main difference is that the atom ordering is more complicated, see Section 3.11. Let N be a possibly infinite set of ground clauses.

Definition 3.12.1 (Ground Clause Ordering)**.** Let $\prec$ be a strict rewrite ordering total on ground terms and ground atoms. Then $\prec$ can be lifted to a total ordering $\prec_L$ on literals by its multiset extension $\prec_{\text{mul}}$ where a positive literal

$P(t_1, \ldots, t_n)$ is mapped to the multiset $\{P(t_1, \ldots, t_n)\}$ and a negative literal $\neg P(t_1, \ldots, t_n)$ to the multiset $\{P(t_1, \ldots, t_n), P(t_1, \ldots, t_n)\}$. The ordering $\prec_L$ is further lifted to a total ordering on clauses $\prec_C$ by considering the multiset extension of $\prec_L$ for clauses.

Proposition 3.12.2 (Properties of the Ground Clause Ordering)**.**

1. The orderings on literals and clauses are total and well-founded.
2. Let C and D be clauses with $P(t_1, \ldots, t_n) = \text{atom}(\max(C))$, $Q(s_1, \ldots, s_m) = \text{atom}(\max(D))$, where $\max(C)$ denotes the maximal literal in C.
   (a) If $Q(s_1, \ldots, s_m) \prec_L P(t_1, \ldots, t_n)$ then $D \prec_C C$.
   (b) If $P(t_1, \ldots, t_n) = Q(s_1, \ldots, s_m)$, $P(t_1, \ldots, t_n)$ occurs negatively in C but only positively in D, then $D \prec_C C$.

Eventually, as I did for propositional logic, I overload $\prec$ with $\prec_L$ and $\prec_C$. So if $\prec$ is applied to literals it denotes $\prec_L$, if it is applied to clauses, it denotes $\prec_C$. Note that $\prec$ is a total ordering on literals and clauses as well. For superposition, inferences are restricted to maximal literals with respect to $\prec$. For a clause set N, I define $N^{\prec C} = \{D \in N \mid D \prec C\}$.

Definition 3.12.3 (Abstract Redundancy)**.** A ground clause C is *redundant* with respect to a set of ground clauses N if $N^{\prec C} \models C$.

Tautologies are redundant. Subsumed clauses are redundant if $\subseteq$ is strict. Duplicate clauses are anyway eliminated quietly because the calculus operates on sets of clauses.

Note that for finite N, and any $C \in N$ redundancy $N^{\prec C} \models C$ can be decided but is as hard as testing unsatisfiability for a clause set N. So the goal is to invent redundancy notions that can be efficiently decided and that are useful.

Definition 3.12.4 (Selection Function)**.** The selection function sel maps clauses to one of its negative literals or $\bot$. If $\text{sel}(C) = \neg P(t_1, \ldots, t_n)$ then $\neg P(t_1, \ldots, t_n)$ is called *selected* in C. If $\text{sel}(C) = \bot$ then no literal in C is *selected*.

The selection function is, in addition to the ordering, a further means to restrict superposition inferences. If a negative literal is selected in a clause, any superposition inference must be on the selected literal.

Definition 3.12.5 (Partial Model Construction)**.** Given a clause set N and an ordering $\prec$ we can construct a (partial) model $N_\mathcal{I}$ for N inductively as follows:

$$N_C := \bigcup_{D \prec C} \delta_D$$

$$\delta_D := \begin{cases} \{P(t_1, \ldots, t_n)\} & \text{if } D = D' \vee P(t_1, \ldots, t_n), P(t_1, \ldots, t_n) \text{ strictly maximal, no literal} \\ & \text{selected in } D \text{ and } N_D \not\models D \\ \emptyset \quad \text{otherwise} \end{cases}$$

$$N_\mathcal{I} := \bigcup_{C \in N} \delta_C$$

Clauses C with $\delta_C \neq \emptyset$ are called *productive*.

Proposition 3.12.6 (Propertied of the Model Operator)**.** Some properties of the partial model construction.

1. For every D with $(C \vee \neg P(t_1, \ldots, t_n)) \prec D$ we have $\delta_D \neq \{P(t_1, \ldots, t_n)\}$.
2. If $\delta_C = \{P(t_1, \ldots, t_n)\}$ then $N_C \cup \delta_C \models C$.
3. If $N_C \models D$ and $D \prec C$ then for all C' with $C \prec C'$ we have $N_{C'} \models D$ and in particular $N_{\mathcal{I}} \models D$.
4. There is no clause C with $P(t_1, \ldots, t_n) \vee P(t_1, \ldots, t_n) \prec C$ such that $\delta_C = \{P(t_1, \ldots, t_n)\}$.

Please properly distinguish: N is a set of clauses interpreted as the conjunction of all clauses. $N^{\prec C}$ is of set of clauses from N strictly smaller than C with respect to $\prec$. $N_{\mathcal{I}}$, N_C are Herbrand interpretations (see Proposition 3.5.3). $N_{\mathcal{I}}$ is the overall (partial) model for N, whereas N_C is generated from all clauses from N strictly smaller than C.

Superposition Left $(N \uplus \{C_1 \vee P(t_1, \ldots, t_n), C_2 \vee \neg P(t_1, \ldots, t_n)\}) \Rightarrow_{\text{SUP}} (N \cup \{C_1 \vee P(t_1, \ldots, t_n), C_2 \vee \neg P(t_1, \ldots, t_n)\} \cup \{C_1 \vee C_2\})$
where (i) $P(t_1, \ldots, t_n)$ is strictly maximal in $C_1 \vee P(t_1, \ldots, t_n)$ (ii) no literal in $C_1 \vee P(t_1, \ldots, t_n)$ is selected (iii) $\neg P(t_1, \ldots, t_n)$ is maximal and no literal selected in $C_2 \vee \neg P(t_1, \ldots, t_n)$, or $\neg P(t_1, \ldots, t_n)$ is selected in $C_2 \vee \neg P(t_1, \ldots, t_n)$

Factoring $(N \uplus \{C \vee P(t_1, \ldots, t_n) \vee P(t_1, \ldots, t_n)\}) \Rightarrow_{\text{SUP}} (N \cup \{C \vee P(t_1, \ldots, t_n) \vee P(t_1, \ldots, t_n)\} \cup \{C \vee P(t_1, \ldots, t_n)\})$
where (i) $P(t_1, \ldots, t_n)$ is maximal in $C \vee P(t_1, \ldots, t_n) \vee P(t_1, \ldots, t_n)$ (ii) no literal is selected in $C \vee P(t_1, \ldots, t_n) \vee P(t_1, \ldots, t_n)$

Note that the superposition factoring rule differs from the resolution factoring rule in that it only applies to positive literals.

Definition 3.12.7 (Saturation)**.** A set N of clauses is called *saturated up to redundancy*, if any inference from non-redundant clauses in N yields a redundant clause with respect to N or is contained in N.

Examples for specific redundancy rules that can be efficiently decided are

Subsumption $(N \uplus \{C_1, C_2\}) \Rightarrow_{\text{SUP}} (N \cup \{C_1\})$
provided $C_1 \subset C_2$

Tautology Deletion $(N \uplus \{C \vee P(t_1, \ldots, t_n) \vee \neg P(t_1, \ldots, t_n)\}) \Rightarrow_{\text{SUP}} (N)$

Condensation $(N \uplus \{C_1 \vee L \vee L\}) \Rightarrow_{\text{SUP}} (N \cup \{C_1 \vee L\})$

Subsumption Resolution where $C_1 \subseteq C_2$

$$(N \uplus \{C_1 \vee L, C_2 \vee \neg L\}) \Rightarrow_{\text{SUP}} (N \cup \{C_1 \vee L, C_2\})$$

Proposition 3.12.8 (Completeness of the Reduction Rules)**.** All clauses removed by Subsumption, Tautology Deletion, Condensation and Subsumption Resolution are redundant with respect to the kept or added clauses.

Theorem 3.12.9 (Completeness)**.** Let N be a, possibly countably infinite, set of ground clauses. If N is saturated up to redundancy and $\bot \notin N$ then N is satisfiable and $N_{\mathcal{I}} \models N$.

Proof. The proof is by contradiction. So I assume: (i) for any clause D derived by Superposition Left or Factoring from N that D is redundant, i.e., $N^{\prec D} \models D$, (ii) $\bot \notin N$ and (iii) $N_{\mathcal{I}} \not\models N$. Then there is a minimal, with respect to $\prec$, clause $C \vee L \in N$ such that $N_{\mathcal{I}} \not\models C \vee L$ and L is a selected literal in $C \vee L$ or no literal in $C \vee L$ is selected and L is maximal. This clause must exist because $\bot \notin N$.

The clause $C \vee L$ is not redundant. For otherwise, $N^{\prec C \vee L} \models C \vee L$ and hence $N_{\mathcal{I}} \models C \vee L$, because $N_{\mathcal{I}} \models N^{\prec C \vee L}$, a contradiction.

I distinguish the case L is a positive and no literal selected in $C \vee L$ or L is a negative literal. Firstly, assume L is positive, i.e., $L = P(t_1, \ldots, t_n)$ for some ground atom $P(t_1, \ldots, t_n)$. Now if $P(t_1, \ldots, t_n)$ is strictly maximal in $C \vee P(t_1, \ldots, t_n)$ then actually $\delta_{C \vee P} = \{P(t_1, \ldots, t_n)\}$ and hence $N_{\mathcal{I}} \models C \vee P$, a contradiction. So $P(t_1, \ldots, t_n)$ is not strictly maximal. But then actually $C \vee P(t_1, \ldots, t_n)$ has the form $C'_1 \vee P(t_1, \ldots, t_n) \vee P(t_1, \ldots, t_n)$ and Factoring derives $C'_1 \vee P(t_1, \ldots, t_n)$ where $(C'_1 \vee P(t_1, \ldots, t_n)) \prec (C'_1 \vee P(t_1, \ldots, t_n) \vee P(t_1, \ldots, t_n))$. Now $C'_1 \vee P(t_1, \ldots, t_n)$ is not redundant, strictly smaller than $C \vee L$, we have $C'_1 \vee P(t_1, \ldots, t_n) \in N$ and $N_{\mathcal{I}} \not\models C'_1 \vee P(t_1, \ldots, t_n)$, a contradiction against the choice that $C \vee L$ is minimal.

Secondly, let us assume L is negative, i.e., $L = \neg P(t_1, \ldots, t_n)$ for some ground atom $P(t_1, \ldots, t_n)$. Then, since $N_{\mathcal{I}} \not\models C \vee \neg P(t_1, \ldots, t_n)$ we know $P(t_1, \ldots, t_n) \in N_{\mathcal{I}}$. So there is a clause $D \vee P(t_1, \ldots, t_n) \in N$ where $\delta_{D \vee P(t_1, \ldots, t_n)} = \{P(t_1, \ldots, t_n)\}$ and $P(t_1, \ldots, t_n)$ is strictly maximal in $D \vee P(t_1, \ldots, t_n)$ and $(D \vee P(t_1, \ldots, t_n)) \prec (C \vee \neg P(t_1, \ldots, t_n))$. So Superposition Left derives $C \vee D$ where $(C \vee D) \prec (C \vee \neg P(t_1, \ldots, t_n))$. The derived clause $C \vee D$ cannot be redundant, because for otherwise either $N^{\prec D \vee P(t_1, \ldots, t_n)} \models D \vee P(t_1, \ldots, t_n)$ or $N^{\prec C \vee \neg P(t_1, \ldots, t_n)} \models C \vee \neg P(t_1, \ldots, t_n)$. So $C \vee D \in N$ and $N_{\mathcal{I}} \not\models C \vee D$, a contradiction against the choice that $C \vee L$ is the minimal false clause. □

So the proof actually tells us that at any point in time we need only to consider either a superposition left inference between a minimal false clause and a productive clause or a factoring inference on a minimal false clause.

Theorem 3.12.10 (Compactness of First-Order Logic)**.** Let N be a, possibly countably infinite, set of first-order logic ground clauses. Then N is unsatisfiable iff there is a finite subset $N' \subseteq N$ such that N' is unsatisfiable.

Proof. If N is unsatisfiable, saturation via superposition generates $\bot$. So there is an i such that $N \Rightarrow^{i}_{\mathrm{SUP}} N'$ and $\bot \in N'$. The clause $\bot$ is the result of at most i-many superposition inferences, reductions on clauses $\{C_1, \ldots, C_n\} \subseteq N$. Superposition is sound, so $\{C_1, \ldots, C_n\}$ is a finite, unsatisfiable subset of N. □

Corollary 3.12.11 (Compactness of First-Order Logic: Classical)**.** A set N of clauses is satisfiable iff all finite subsets of N are satisfiable.

Theorem 3.12.12 (Soundness and Completeness of Ground Superposition)**.** A first-order Σ-sentence ϕ is valid iff there exists a ground superposition refutation for $\mathrm{grd}(\Sigma, \mathrm{cnf}(\neg\phi))$.

Proof. A first-order sentence ϕ is valid iff $\neg\phi$ is unsatisfiable iff $\mathrm{acnf}(\neg\phi)$ is unsatisfiable iff $\mathrm{grd}(\Sigma, \mathrm{cnf}(\neg\phi))$ is unsatisfiable iff superposition provides a refutation of $\mathrm{grd}(\Sigma, \mathrm{cnf}(\neg\phi))$. □

Theorem 3.12.13 (Semi-Decidability of First-Order Logic by Ground Superposition)**.** If a first-order Σ-sentence ϕ is valid then a ground superposition refutation can be computed.

Proof. In a fair way enumerate $\mathrm{grd}(\Sigma, \mathrm{acnf}(\neg\phi))$ and perform superposition inference steps. The enumeration can, e.g., be done by considering Herbrand terms of increasing size. □

Example 3.12.14 (Ground Superposition)**.** Consider the below clauses 1-4 and superposition refutation with respect a KBO with precedence $P \succ Q \succ g \succ f \succ c \succ b \succ a$ where the weight function w returns 1 for all signature symbols. Maximal literals are marked with a *.

1.	$\neg P(f(c))^* \vee \neg P(f(c))^* \vee Q(b)$	(Input)
2.	$P(f(c))^* \vee Q(b)$	(Input)
3.	$\neg P(g(b,c))^* \vee \neg Q(b)$	(Input)
4.	$P(g(b,c))^*$	(Input)
5.	$\neg P(f(c))^* \vee Q(b)$	(Cond(1))
6.	$Q(b)^* \vee Q(b)^*$	(Sup(5, 2)))
7.	$Q(b)^*$	(Fact(6))
8.	$\neg Q(b)^*$	(Sup(3, 4))
10.	$\bot$	(Sup(8, 7))

Note that clause 5 cannot be derived by Factoring whereas clause 7 can also be derived by Condensation. Clause 8 is also the result of a Subsumption Resolution application to clauses 3, 4.

Theorem 3.12.15 (Craig Theorem [**?**])**.** Let ϕ and ψ be two propositional (first-order ground) formulas so that $\phi \models \psi$. Then there exists a formula χ (called the *interpolant* for $\phi \models \psi$), so that χ contains only propositional variables (first-order signature symbols) occurring both in ϕ and in ψ so that $\phi \models \chi$ and $\chi \models \psi$.

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

3.13 First-Order Superposition

Now the result for ground superposition are lifted to superposition on first-order clauses with variables, still without equality. The completeness proof of ground superposition above talks about (strictly) maximal literals of *ground* clauses. The non-ground calculus considers those literals that correspond to (strictly) maximal literals of ground instances.

The used ordering is exactly the ordering of Definition 3.12.1 where clauses with variables are projected to their ground instances for ordering computations.

Definition 3.13.1 (Maximal Literal)**.** A literal L is called *maximal* in a clause C if and only if there exists a grounding substitution σ so that $L\sigma$ is maximal in $C\sigma$, i.e., there is no different $L' \in C$: $L\sigma \prec L'\sigma$. The literal L is called *strictly maximal* if there is no different $L' \in C$ such that $L\sigma \preceq L'\sigma$.

Note that the orderings KBO and LPO cannot be total on atoms with variables, because they are stable under substitutions. Therefore, maximality can also be defined on the basis of absence of greater literals. A literal L is called *maximal* in a clause C if $L \not\prec L'$ for all other literals $L' \in C$. It is called *strictly maximal* in a clause C if $L \not\preceq L'$ for all other literals $L' \in C$.

Superposition Left $(N \uplus \{C_1 \vee P(t_1,\ldots,t_n), C_2 \vee \neg P(s_1,\ldots,s_n)\}) \Rightarrow_{\mathrm{SUP}}$ $(N \cup \{C_1 \vee P(t_1,\ldots,t_n), C_2 \vee \neg P(s_1,\ldots,s_n)\} \cup \{(C_1 \vee C_2)\sigma\})$

where (i) $P(t_1,\ldots,t_n)\sigma$ is strictly maximal in $(C_1 \vee P(t_1,\ldots,t_n))\sigma$ (ii) no literal in $C_1 \vee P(t_1,\ldots,t_n)$ is selected (iii) $\neg P(s_1,\ldots,s_n)\sigma$ is maximal and no literal selected in $(C_2 \vee \neg P(s_1,\ldots,s_n))\sigma$, or $\neg P(s_1,\ldots,s_n)$ is selected in $(C_2 \vee \neg P(s_1,\ldots,s_n))\sigma$ (iv) σ is the mgu of $P(t_1,\ldots,t_n)$ and $P(s_1,\ldots,s_n)$

Factoring $(N \uplus \{C \vee P(t_1,\ldots,t_n) \vee P(s_1,\ldots,s_n)\}) \Rightarrow_{\mathrm{SUP}}$ $(N \cup \{C \vee P(t_1,\ldots,t_n) \vee P(s_1,\ldots,s_n)\} \cup \{(C \vee P(t_1,\ldots,t_n))\sigma\})$

where (i) $P(t_1,\ldots,t_n)\sigma$ is maximal in $(C \vee P(t_1,\ldots,t_n) \vee P(s_1,\ldots,s_n))\sigma$ (ii) no literal is selected in $C \vee P(t_1,\ldots,t_n) \vee P(s_1,\ldots,s_n)$ (iii) σ is the mgu of $P(t_1,\ldots,t_n)$ and $P(s_1,\ldots,s_n)$

Note that the above inference rules Superposition Left and Factoring are generalizations of their respective counterparts from the ground superposition calculus above. Therefore, on ground clauses they coincide. Therefore, we can safely overload them in the sequel.

Definition 3.13.2 (Abstract Redundancy)**.** A clause C is *redundant* with respect to a clause set N if for all ground instances $C\sigma$ there are clauses $\{C_1, \ldots, C_n\} \subseteq N$ with ground instances $C_1\tau_1, \ldots, C_n\tau_n$ such that $C_i\tau_i \prec C\sigma$ for all i and $C_1\tau_1, \ldots, C_n\tau_n \models C\sigma$.

Definition 3.13.3 (Saturation)**.** A set N of clauses is called *saturated up to redundancy*, if any inference from non-redundant clauses in N yields a redundant clause with respect to N or is contained in N.

In contrast to the ground case, the above abstract notion of redundancy is not effective, i.e., it is undecidable for some clause C whether it is redundant, in general. Nevertheless, the concrete ground redundancy notions carry over to the non-ground case. Note also that a clause C is contained in N modulo renaming of variables.

Let rdup be a function from clauses to clauses that removes duplicate literals, i.e., $\mathrm{rdup}(C) = C'$ where $C' \subseteq C$, C' does not contain any duplicate literals, and for each $L \in C$ also $L \in C'$.

Subsumption $(N \uplus \{C_1, C_2\}) \Rightarrow_{\mathrm{SUP}} (N \cup \{C_1\})$
provided $C_1\sigma \subset C_2$ for some σ

Tautology Deletion $(N \uplus \{C \vee P(t_1, \ldots, t_n) \vee \neg P(t_1, \ldots, t_n)\}) \Rightarrow_{\mathrm{SUP}} (N)$

Condensation $(N \uplus \{C_1 \vee L \vee L'\}) \Rightarrow_{\mathrm{SUP}} (N \cup \{\mathrm{rdup}((C_1 \vee L \vee L')\sigma)\})$
provided $L\sigma = L'$ and $\mathrm{rdup}((C_1 \vee L \vee L')\sigma)$ subsumes $C_1 \vee L \vee L'$ for some σ

Subsumption Resolution $(N \uplus \{C_1 \vee L, C_2 \vee L'\}) \Rightarrow_{\mathrm{SUP}} (N \cup \{C_1 \vee L, C_2\})$
where $L\sigma = \neg L'$ and $C_1\sigma \subseteq C_2$ for some σ

Lemma 3.13.4. All reduction rules are instances of the abstract redundancy criterion.

Proof. Do it □

Lemma 3.13.5 (Subsumption is NP-complete)**.** Subsumption is NP-complete.

Proof. Let C_1 subsume C_2 with substitution σ Subsumption is in NP because the size of σ is bounded by the size of C_2 and the subset relation can be checked in time at most quadratic in the size of C_1 and C_2.

Propositional SAT can be reduced as follows. Assume a 3-SAT clause set N. Consider a 3-place predicate R and a unary function g and a mapping from propositional variables P to first order variables x_P. □

Lemma 3.13.6 (Lifting)**.** Let $D \vee L$ and $C \vee L'$ be variable-disjoint clauses and σ a grounding substitution for $C \vee L$ and $D \vee L'$. If there is a superposition left inference

$$(N \uplus \{(D \vee L)\sigma, (C \vee L')\sigma\}) \Rightarrow_{\text{SUP}} (N \cup \{(D \vee L)\sigma, (C \vee L')\sigma\} \cup \{D\sigma \vee C\sigma\})$$

and if $\text{sel}((D \vee L)\sigma) = \text{sel}((D \vee L))\sigma$, $\text{sel}((C \vee L')\sigma) = \text{sel}((C \vee L'))\sigma$, then there exists a mgu τ such that

$$(N \uplus \{D \vee L, C \vee L'\}) \Rightarrow_{\text{SUP}} (N \cup \{D \vee L, C \vee L'\} \cup \{(D \vee C)\tau\}).$$

Let $C \vee L \vee L'$ be a clause and σ a grounding substitution for $C \vee L \vee L'$. If there is a factoring inference

$$(N \uplus \{(C \vee L \vee L')\sigma\}) \Rightarrow_{\text{SUP}} (N \cup \{(C \vee L \vee L')\sigma\} \cup \{(C \vee L)\sigma\})$$

and if $\text{sel}((C \vee L \vee L')\sigma) = \text{sel}((C \vee L \vee L'))\sigma$, then there exists a mgu τ such that

$$(N \uplus \{C \vee L \vee L'\}) \Rightarrow_{\text{SUP}} (N \cup \{C \vee L \vee L'\} \cup \{(C \vee L)\tau\})$$

Note that in the above lemma the clause $D\sigma \vee C\sigma$ is an instance of the clause $(D \vee C)\tau$. The reduction rules cannot be lifted in the same way as the following example shows.

Example 3.13.7 (First-Order Reductions are not Liftable)**.** Consider the two clauses $P(x) \vee Q(x)$, $P(g(y))$ and grounding substitution $\{x \mapsto g(a), y \mapsto a\}$. Then $P(g(y))\sigma$ subsumes $(P(x) \vee Q(x))\sigma$ but $P(g(y))$ does not subsume $P(x) \vee Q(x)$. For all other reduction rules similar examples can be constructed.

Lemma 3.13.8 (Soundness and Completeness)**.** First-Order Superposition is sound and complete.

Proof. Soundness is obvious. For completeness, Theorem 3.12.12 proves the ground case. Now by applying Lemma 3.13.6 to this proof it can be lifted to the first-order level, as argued in the following.

Let N be a an unsatisfiable set of first-order clauses. By Theorem 3.5.5 and Lemma 3.6.10 there exist a finite unsatisfiable set N' of ground instances from clauses from N such that for each clause $C\sigma \in N'$ there is a clause $C \in N$. Now ground superposition is complete, Theorem 3.12.12, so there exists a derivation of the empty clause by ground superposition from N': $N' = N'_0 \Rightarrow_{\text{SUP}} \ldots \Rightarrow_{\text{SUP}} N'_k$ and $\bot \in N'_k$. Now by an inductive argument on the length of the derivation k this derivation can be lifted to the first-order level. The invariant is: for any ground clause $C\sigma \in N'_i$ used in the ground proof, there is a clause $C \in N_i$ on the first-order level. The induction base holds for N and N' by construction. For the induction step Lemma 3.13.6 delivers the result. □

There are questions left open by Lemma 3.13.8. It just says that a ground refutation can be lifted to a first-order refutation. But what about abstract redundancy, Definition 3.13.2? Can first-order redundant clauses be deleted

without harming completeness? And what about the ground model operator with respect to clause sets N saturated on the first-order level. Is in this case $\mathrm{grd}(\Sigma, N)_{\mathcal{I}}$ a model? The next two lemmas answer these questions positively.

Lemma 3.13.9 (Redundant Clauses are Obsolete)**.** If a clause set N is unsatisfiable, then there is a derivation $N \Rightarrow^*_{\mathrm{SUP}} N'$ such that $\bot \in N'$ and no clause in the derivation of $\bot$ is redundant.

Proof. If N is unsatisfiable then there is a ground superposition refutation of $\mathrm{grd}(\Sigma, N)$ such that no ground clause in the refutation is redundant. Now according to Lemma 3.13.8 this proof can be lifted to the first-order level. Now assume some clause C in the first-order proof is redundant that is the lifting of some clause $C\sigma$ from the ground proof with respect to a grounding substitution σ. The clause C is redundant by Definition 3.13.2 if all its ground instances are, in particular, $C\sigma$. But this contradicts the fact that the lifted ground proof does not contain redundant clauses. □

Lemma 3.13.10 (Model Property)**.** If N is a saturated clause set and $\bot \notin N$ then $\mathrm{grd}(\Sigma, N)_{\mathcal{I}} \models N$.

Proof. As usual we assume that selection on the ground and respective non-ground clauses is identical. Assume $\mathrm{grd}(\Sigma, N)_{\mathcal{I}} \not\models N$. Then there is a minimal ground clause $C\sigma$, $C \neq \bot$, $C \in N$ such that $\mathrm{grd}(\Sigma, N)_{\mathcal{I}} \not\models C\sigma$. Note that $C\sigma$ is not redundant as for otherwise $\mathrm{grd}(\Sigma, N)_{\mathcal{I}} \models C\sigma$. So $\mathrm{grd}(\Sigma, N)$ is not saturated. If $C\sigma$ is productive, i.e., $C\sigma = (C' \vee L)\sigma$ such that L is positive, $L\sigma$ strictly maximal in $(C' \vee L)\sigma$ then $L\sigma \in \mathrm{grd}(\Sigma, N)_{\mathcal{I}}$ and hence $\mathrm{grd}(\Sigma, N)_{\mathcal{I}} \models C\sigma$ contradicting $\mathrm{grd}(\Sigma, N)_{\mathcal{I}} \not\models C\sigma$.

If $C\sigma = (C' \vee L \vee L')\sigma$ such that L is positive, $L\sigma$ maximal in $(C' \vee L \vee L')\sigma$ then, because N is saturated, there is a clause $(C' \vee L)\tau \in N$ such that $(C' \vee L)\tau\sigma = (C' \vee L)\sigma$. Now $(C' \vee L)\tau$ is not redundant, $\mathrm{grd}(\Sigma, N)_{\mathcal{I}} \not\models (C' \vee L)\tau$, contradicting the minimal choice of $C\sigma$.

If $C\sigma = (C' \vee L)\sigma$ such that L is selected, or negative and maximal then there is a clause $(D' \vee L') \in N$ and grounding substitution ρ, such that $L'\rho$ is a strictly maximal positive literal in $(D' \vee L')\rho$, $L'\rho \in \mathrm{grd}(\Sigma, N)_{\mathcal{I}}$ and $L'\rho = \neg L\sigma$. Again, since N is saturated, there is variable disjoint clause $(C' \vee D')\tau \in N$ for some unifier τ, $(C' \vee D')\tau\sigma\rho \prec C\sigma$, and $\mathrm{grd}(\Sigma, N)_{\mathcal{I}} \not\models (C' \vee D')\tau\sigma\rho$ contradicting the minimal choice of $C\sigma$. □

Dynamic stuff: a clause C is called *persistent* in a derivation $N \rightarrow^*_{\mathrm{SUP}} N'$ if there is some i such that $C \in N_i$ for $N \rightarrow^i_{\mathrm{SUP}} N_i$ and for all $j > i$, $N \rightarrow^j_{\mathrm{SUP}} N_j$ then $C \in N_j$. A derivation $N \rightarrow^*_{\mathrm{SUP}} N'$ is called *fair* if any two persistent clauses C, D and any superposition inference C' out of the two clauses, there is an index j such with $N \rightarrow^j_{\mathrm{SUP}} N_j \rightarrow^*_{\mathrm{SUP}} N'$ such that $C' \in N_j$.

Definition 3.13.11 (Persistent Clause)**.** Let $N_0 \Rightarrow_{\mathrm{SUP}} N_1 \Rightarrow_{\mathrm{SUP}} \ldots$ be a, possibly infinite, superposition derivation. A clause C is called *persistent* in this derivation if $C \in N_i$ for some i and for all $j > i$ also $C \in N_j$.

Definition 3.13.12 (Fair Derivation)**.** A derivation $N_0 \Rightarrow_{\text{SUP}} N_1 \Rightarrow_{\text{SUP}} \ldots$ is called *fair* if for any persistent clause $C \in N_i$ where factoring is applicable to C, there is a j such that the factor of $C' \in N_j$ or $\bot \in N_j$. If $\{C, D\} \subseteq N_i$ are persistent clauses such that superposition left is applicable to C, D then the superposition left result is also in N_j for some j or $\bot \in N_j$.

Theorem 3.13.13 (Dynamic Superposition Completeness)**.** If N is unsatisfiable and $N = N_0 \Rightarrow_{\text{SUP}} N_1 \Rightarrow_{\text{SUP}} \ldots$ is a fair derivation, then there is $\bot \in N_j$ for some j.

Proof. If N is unsatisfiable, then by Lemma 3.13.8 there is a derivation of $\bot$ by superposition. Furthermore, no clause contributing to the derivation of $\bot$ is redundant (Lemma 3.13.9). So all clauses in the derivation of $\bot$ are persistent. The derivation $N_0 \Rightarrow_{\text{SUP}} N_1 \Rightarrow_{\text{SUP}} \ldots$ is fair, hence $\bot \in N_j$ for some j. □

Algorithm 12: SupProver(N)

```
   Input  : A set of clauses N.
   Output: A saturated set of clauses N', equivalent to N.
 1 WO := ∅;
 2 US := N;
 3 while (US ≠ ∅ and ⊥ ∉ US) do
 4 |   Given:= pick a clause from US;
 5 |   WO := WO ∪ {Given};
 6 |   New := SupLeft(WO,Given) ∪ Fact(Given);
 7 |   while (New ≠ ∅) do
 8 |   |   Given:= pick a clause from New;
 9 |   |   if (!TautDel(Given)) then
10 |   |   |   if (!SubDel(Given,WO ∪US)) then
11 |   |   |   |   Given:= Cond(Given);
12 |   |   |   |   Given:= SubRes(Given,WO);
13 |   |   |   |   WO:= SubDel(WO,Given);
14 |   |   |   |   US:= SubDel(US,Given);
15 |   |   |   |   New:= New ∪ SubRes(WO ∪US,Given);
16 |   |   |   |   US:= US ∪ {Given };
17 |   |   |   |
18 |   |   |
19 |   end
20 end
21 return WO;
```

Lemma 3.13.14. Let red(N) be all clauses that are redundant with respect to the clauses in N and N, M be clause sets. Then

1. if $N \subseteq M$ then red(N) $\subseteq$ red(M)

clause $Q(f(x_1, \ldots, x_n))$ is the only strictly maximal literal. The second form is

$$\neg P_1(x) \vee \ldots \vee \neg P_n(x) \vee Q(x)$$

where maximality depends on the precedence on the predicates. Importantly, there are only finitely many different clauses of the above two forms with respect to condensation and subsumption. If the positive Q contains a non-constant ground term initially, this can also be transformed into a finite set of equivalent clauses of the above first form. In addition, only for the above two forms a positive literal can become maximal and therefore be used in a superposition inference. Then any superposition inference generates either a clause of the obove form, and there are only finitely many, or the resulting clause is strictly smaller with respect to the multiset of all subterms of the parent clause that has not the above form. □

3.16 Decision Procedures for the Bernays-Schönfinkel (BS) Fragment

In Section 3.15 I showed that unsatisfiability (validity) of first-order logic (clause sets) is undecidable, Theorem 3.15.1. So decision procedures can only exists for fragments, e.g., the Bernays-Schönfinkel (BS) or the MSLH fragment introduced in Section 3.15. This section presents several decision procedures for the BS fragment. Some of them can be extended to complete calculi for full first-order logic and some are refinements of full first-order logic calculi.

Historically, the BS fragment has been defined as all first-order sentences of the form $\exists\vec{x}.\forall\vec{y}.\phi$ where ϕ does not contain constant or function symbols, Definition 3.15.3. After Skolemization, satisfiability is equivalent to the formula $\forall^*\vec{y}.(\phi\{x_1 \mapsto a_1, \ldots, x_n \mapsto a_n\})$ for (fresh) constants $a_1, \ldots, a_n$ which can then be further transformed into CNF.

Thus the Herbrand domain of a BS clause set N is finite, consisting of all constants in N. So is the equisatisfiable set $\mathrm{grd}(N)$ where satisfiability can then be decided by any decision procedure for propositional logic. However, the set $\mathrm{grd}(N)$ is exponentially larger than N, in general. If k is the maximal number of variables of a clause in N, and n the number of different constants in N, then worst-case $|\mathrm{grd}(N)| = \mathrm{O}(|N| \cdot n^k)$. This motivates research for more flexible calculi without a worst-case initial blow-up.

3.16.1 Superposition

The superposition calculus, Section 3.13, is not a decision procedure for the BS fragment, i.e., it does not necessarily terminate on a clause set N of BS clauses, see also Example 3.15.5. Consider a BS clause set consisting of the following two clauses

$$1 \quad \neg R(x, y) \vee \neg R(y, z) \vee R(x, z)$$

$$2 \quad R(x, y) \vee R(y, x)$$

describing a transitive and total relation R. With respect to any ordering stable under substitution, the R literals are all incomparable in their respective clauses. The only way to restrict inferences via the superposition calculus is to select one of the negative literals in the transitivity clause, clause 1. The superposition calculus generates an infinite number of clauses including

$$\begin{aligned}
&N_0 = \{1 : \neg R(x,y) \vee \neg R(y,z) \vee R(x,z),\ 2 : R(x,y) \vee R(y,x)\}\\
&\Rightarrow^{1.1,2.1}_{\text{SUP}} \quad N_0 \cup \{3 : \neg R(y,z) \vee R(x,z) \vee R(y,x)\}\\
&\Rightarrow^{3.1,2.1}_{\text{SUP}} \quad N_1 \cup \{4 : R(x,z) \vee R(y,x) \vee R(z,y)\}\\
&\Rightarrow^{4.1,4.2}_{\text{SUP}} \quad N_2 \cup \{5 : R(x,x)\}\\
&\Rightarrow^{4.1,3.1}_{\text{SUP}} \quad N_3 \cup \{6 : R(x,z) \vee R(y,x) \vee R(y',y) \vee R(z,y')\}\\
&\vdots
\end{aligned}$$

The crucial point is that neither clause 4 nor clause 6 is redundant because of the underlying variable chains. The variable chain can be extended generating clauses of length five, six, ... Obviously, such a clause containing a variable chain contributes a refutation at most of the length of square of the number of different constant symbols. Otherwise, it becomes redundant by the existence of shorter clauses. So one way to turn superposition into a decision procedure for the BS class is to add an additional condensation rule that unifies literals in clauses as soon as all potential ground instantiations with constants yield duplicates.

Condensation-BS $(N \uplus \{L_1 \vee \cdots \vee L_n\}) \Rightarrow_{\text{SUP}} (N \cup \{\text{rdup}((L_1 \vee \ldots L_n)\sigma_{i,j}) \mid \sigma_{i,j} = \text{mgu}(L_i, L_j) \text{ and } \sigma_{i,j} \neq \bot\})$
provided any ground instance $(L_1 \vee \cdots \vee L_n)\delta$ contains at least two duplicate literals

Another way to prevent non-termination is by preventing the generation of arbitrary long clauses. This can be done by a special splitting rule, that splits non-Horn clauses into their Horn parts through instantiation. Assuming two constants a, b for the above example, then clause 2 is replaced by clauses

$$\begin{aligned}
&2.1 \quad R(a,b) \vee R(b,a)\\
&2.2 \quad R(a,a) \qquad .\\
&2.3 \quad R(b,b)
\end{aligned}$$

Next the clause $R(a,b) \vee R(b,a)$ can be split, similar to a β-rule application of tableau, resulting in two clause sets

$$\begin{aligned}
M_1 &= \{\neg R(x,y) \vee \neg R(y,z) \vee R(x,z),\ R(a,a),\ R(b,b),\ R(a,b)\}\\
M_2 &= \{\neg R(x,y) \vee \neg R(y,z) \vee R(x,z),\ R(a,a),\ R(b,b),\ R(b,a)\}.
\end{aligned}$$

Now the original clause set N_0 is satisfiable iff M_1 or M_2 are satisfiable. Employing a rigourous selection strategy where in every clause containing negative literals one negative literal is selected, the superposition calculus will always infer from a positive unit clause and a clause containing at least one negative literal a shorter clause. So it will terminate.

A state is now a set of clause sets. Let k be the number of different constants $a_1, \ldots, a_k$ in the initial clause set N. Then the initial state is the set $M = \{N\}$, Superposition Left is adopted to the new setting, Factoring is no longer needed and the rules Instantiate and Split are added. The variables $x_1, \ldots, x_k$ constitute a *variable chain* between literals L_1, L_k inside a clause C, if there are literals $\{L_1, \ldots, L_k\} \subseteq C$ such that $x_i \in (\mathrm{vars}(L_i) \cap \mathrm{vars}(L_{i+1}))$, $1 \leq i < k$.

Superposition-BS $M \uplus \{N \uplus \{P(t_1, \ldots, t_n), C \vee \neg P(s_1, \ldots, s_n)\}\} \Rightarrow_{\mathrm{SUPBS}} M \cup \{N \cup \{P(t_1, \ldots, t_n), C \vee \neg P(s_1, \ldots, s_n)\} \cup \{C\sigma\}\}$
where (i) $\neg P(s_1, \ldots, s_n)$ is selected in $(C \vee \neg P(s_1, \ldots, s_n))\sigma$ (ii) σ is the mgu of $P(t_1, \ldots, t_n)$ and $P(s_1, \ldots, s_n)$ (iii) $C \vee \neg P(s_1, \ldots, s_n)$ is a Horn clause

Instantiation $M \uplus \{N \uplus \{C \vee A_1 \vee A_2\}\} \Rightarrow_{\mathrm{SUPBS}} M \cup \{N \cup \{(C \vee A_1 \vee A_2)\sigma_i \mid \sigma_i = \{x \mapsto a_i\}, 1 \leq i \leq k\}\}$
where x occurs in a variable chain between A_1 and A_2

Split $M \uplus \{N \uplus \{C_1 \vee A_1 \vee C_2 \vee A_2\}\} \Rightarrow_{\mathrm{SUPBS}} M \cup \{N \cup \{C_1 \vee A_1\}, N \cup \{C_2 \vee A_2\}\}$
where $\mathrm{vars}(C_1 \vee A_1) \cap \mathrm{vars}(C_2 \vee A_2) = \emptyset$

As usual, the clause parts C, C_1, C_2 may be empty. Note that after exhaustive application of Instantiation and Split, every clause purely containing positive literals is a unit clause. This together with the below rigorous selection strategy justifies the strong side conditions of Superposition BS compared to Superposition Left and explains whi Factoring is not needed.

Definition 3.16.1 (Rigorous Selection Strategy)**.** A selection strategy is *rigorous* of in any clause containing a negative literal, a negative literal is selected.

Lemma 3.16.2 (SUPBS Basic Properties)**.** The SUPBS rules have the following properties:

1. Superposition BS is sound.
2. Instantiation is sound and complete.
3. Split is sound and complete.

Proof. 1. Follows from the soundness of Superposition Left.

2. Soundness follows from the soundness of variable instantiation. Completeness follows from the fact that $\mathrm{grd}(C \vee A_1 \vee A_2) = \mathrm{grd}(\{(C \vee A_1 \vee A_2)\sigma_i \mid \sigma_i = \{x \mapsto a_i\}, 1 \leq i \leq k\})$.

3. I prove $N \uplus \{C_1 \vee A_1 \vee C_2 \vee A_2\}$ is satisfiable iff $N \cup \{C \vee A_1\}$ or $N \cup \{C_2 \vee A_2\}$ is satisfiable. The direction from right to left is obvious, because both $C_1 \vee A_1$ and $C_2 \vee A_2$ subsume $C_1 \vee A_1 \vee C_2 \vee A_2$. For the other direction assume an interpretation $\mathcal{A}$ satisfying $N \uplus \{C_1 \vee A_1 \vee C_2 \vee A_2\}$, in partiular $\mathcal{A} \models C_1 \vee A_1 \vee C_2 \vee A_2$. This means for any valuation β we have $\mathcal{A}, \beta \models C_1 \vee A_1 \vee C_2 \vee A_2$. The sub-clauses $C_1 \vee A_1$, $C_2 \vee A_2$ are variable disjoint so β can be split into β_1 assigning values to variables in $C_1 \vee A_1$ and β_2 assigning values to variables in $C_2 \vee A_2$. Then $\mathcal{A}, \beta \models C_1 \vee A_1 \vee C_2 \vee A_2$ for all β iff $\mathcal{A}, \beta_1\beta_2 \models C_1 \vee A_1 \vee C_2 \vee A_2$ for all β_1, β_2 iff $\mathcal{A}, \beta_1 \models C_1 \vee A_1$ or $\mathcal{A}, \beta_2 \models C_2 \vee A_2$ for all β_1, β_2.

□

3.16.2 Non-Redundant Clause Learning (NRCL)

The NRCL calculus is a generalization of the CDCL calculus, Section **??**, to the BS fragment. The BS fragment can be finitely grounded, but with a worst-case exponential blow up. So an obvious procedure would be to perform the grounding and then run CDCL on the resulting first-order ground clauses. Every first-order ground atom then corresponds to a propositional variable. However, in general, it is not wise to incorporate into an automated reasoning calculus a worst-case exponential preprocessing step. Therefore, NRCL rather lifts the CDCL calculus to the first-order BS fragment.

Similar to a CDCL state, an *NRCL state* is a five tuple $(\Gamma; N; U; j; C)$, where Γ is a (partial) model assumption build from ground literals, N the initial BS clause set, U the set of learned BS clauses, j the current level and C is either $\top$, $\bot$ or a BS clause. Literals $L \in \Gamma$ are either annotated with a number, a level, i.e., they have the form L^k meaning that L is the $k-th$ guessed decision literal, or they are annotated with a pair consisting of a clause and a (ground) substitution $L\sigma^{(C \vee L)\cdot\sigma}$ that forced the literal to become true. A pair $(C \cdot \sigma)$ is called a *closure*. A literal L is of *level* k with respect to a problem state $(\Gamma; N; U; j; C)$ if L or $\mathrm{comp}(L)$ occurs in Γ and the first decision literal left from L ($\mathrm{comp}(L)$) in Γ is annotated with k or L ($\mathrm{comp}(L)$) is a decision literal in Γ annotated with k. If there is no such decision literal then $k = 0$. A clause D is of *level* k with respect to a problem state $(\Gamma; N; U; j; C)$ if k is the maximal level of a literal in D.

The initial state for testing satisfiability of a BS clause set N is $(\epsilon; N; \emptyset; 0; \top)$, the final state $(\Gamma; N; U; 0; \bot)$ indicates unsatsifiability of the clause set and a final state $(\Gamma; N; U; k; \top)$ where if $\mathcal{H} = \{A \mid \text{ atom } A \in \Gamma\}$ then $\mathcal{H} \models N$.

The rules are:

Propagate $(\Gamma; N; U; k; \top) \Rightarrow_{\mathrm{NRCL}} (\Gamma L\sigma^{(C \setminus \{L_1 \ldots, L_n\} \vee L)\delta\cdot\sigma}; N; U; k; \top)$
provided $C \vee L \in (N \cup U)$, $C\sigma$ is false under Γ for some grounding substitution σ, $L\sigma$ is undefined in Γ, let $L_1\sigma, \ldots, L_n\sigma$ be all copies of $L\sigma$ in $C\sigma$ and δ be the mgu of the $L_1, \ldots, L_n$

Decide $(\Gamma; N; U; k; \top) \Rightarrow_{\mathrm{NRCL}} (\Gamma, L^{k+1}; N; U; k+1; \top)$

provided L is a ground literal undefined under Γ

Conflict $(\Gamma; N; U; k; \top) \Rightarrow_{\text{NRCL}} (\Gamma; N; U; k; D \cdot \sigma)$
provided $D \in (N \cup U)$, $D\sigma$ false in Γ for grounding σ

Skip $(\Gamma L\delta^{(C \vee L)\cdot\delta}; N; U; k; D \cdot \sigma) \Rightarrow_{\text{NRCL}} (\Gamma; N; U; k; D \cdot \sigma)$
provided $\text{comp}(L\delta)$ does not occur in $D\sigma$

Factorize $(\Gamma; N; U; k; (D \vee L \vee L') \cdot \sigma) \Rightarrow_{\text{NRCL}} (\Gamma; N; U; k; ((D \vee L)\tau) \cdot \sigma)$
provided $L\sigma = L'\sigma$, $\tau = \text{mgu}(L, L')$

Resolve $(\Gamma L\delta^{(C \vee L)\cdot\delta}; N; U; k; (D \vee L') \cdot \sigma) \Rightarrow_{\text{NRCL}} (\Gamma; N; U; k; ((D \vee C)\gamma) \cdot \sigma\delta)$
provided $D\sigma$ is of level k, $L'\sigma \notin D\sigma$, $L\delta = \text{comp}(L'\sigma)$, and $\gamma = \text{mgu}(L, \text{comp}(L'))$

Backtrack $(\Gamma K^{i+1}\Gamma'; N; U; k; (D \vee L) \cdot \sigma) \Rightarrow_{\text{NRCL}} (\Gamma L\sigma^{(D \vee L)\cdot\sigma}; N; U \cup \{D \vee L\}; i; \top)$
provided $L\sigma$ is of level k and $D\sigma$ is of level i.

Compared to the propositional CDCL calculs the ground literals on the trail correspond to propositional variables. However, the clauses in N, the learned clauses in U and also the annotated clauses for propagating literals from the trail contain first-order variables. While the model assumption is always a ground model, inferences are performed via the Resolve rule on non-ground clauses, in general. Because of this, an additional rule is needed: Factorize and the Propagate rule also factorizes all occurrences of the propagated literal in the respective clause. In the propositional CDCL calculus factorization happened silently via duplicate literal elimination, here it has to be done explicitely, because for two literals L, K where $L\sigma = K\sigma$ for some grounding substitution σ, the literals are unfiable but not necesserily identical.

Theorem 3.16.3 (NRCL Overall Properties)**.** NRCL is sound, complete and terminating on a set of BS clauses.

3.16.3 Instance Generaction (InstGen)

A substitution σ is a *proper instantiator* with respect to a literal L (clause C), if for some variable $x \in \text{vars}(L)$ ($x \in \text{vars}(C)$), $x\sigma$ is not a variable. Let $\succ$ be a well-founded closure ordering satifying $C \cdot \sigma \succ D \cdot \gamma\tau$ if (i) $C\sigma = D\gamma\tau$ but $C\rho = D$ for some proper instantiator ρ, or, (ii) $D\gamma \subset C$, or (iii) $D\gamma = C$ where γ is not a renaming, nor a proper instantiator for D.

Similar to superposition, Section 3.13, a model operator explicitely constructs a model in case of a saturated clause set. For otherwise, if the model out of the model operator does not satisfy all clauses, the minimal false clause indicates the next minimal inference, in case of the instance generation calculus, the generation of additional clause instances.

The candidate model $\mathcal{I}_N$ is inductively defined over the well-founded closure ordering $\succ$ with respect to a ground model $\mathcal{I}_{N_\alpha}$ of the grounded clause set N_α. The ground clause set N_α is constructed by mapping all variables in N to a distinguished single constant α. Then if N_α is unsatisfiable, so is N. If N_α is satisfiable, N is not necessarily satisfiable and $\mathcal{I}_N$ lifts the model $\mathcal{I}_{N_\alpha}$ of N_α to a candidate model for N. Satisfiability of the clause set N_α can be more efficiently decided by a procedure for SAT (NP $\neq$ NEXPTIME), e.g., a CDCL-based SAT solver.

Similar to the model construction for superposition, suppose the sets $\delta D \cdot \sigma$ have been defined for all closures $D \cdot \sigma$ smaller than $C \cdot \gamma$.

$$\mathcal{I}_{C\cdot\gamma} := \bigcup_{D\cdot\sigma \prec C\cdot\gamma} \delta_{D\cdot\sigma}$$

$$\delta_{C\cdot\gamma} := \begin{cases} \{L\gamma\} & \text{if } C\gamma \text{ is false in } \mathcal{I}_{C\cdot\gamma} \\ & C\cdot\gamma \text{ is the minimal representation of } C\gamma \text{ in } N \\ & L \in C \text{ and } L\gamma \text{ undefined in } \mathcal{I}_{C\cdot\gamma} \text{ and } L_\alpha \in \mathcal{I}_{N_\alpha} \\ \emptyset & \text{otherwise} \end{cases}$$

$$\mathcal{I}_N := \bigcup_{C\in N} \delta_{C\cdot\gamma}$$

Falsify $(N, \top) \Rightarrow_{\text{IGEN}} (N, M)$
where $M = \bot$ if N_α is unsatisfiable and $M = \{L_1, \ldots, L_n\}$ if $\{L_1, \ldots, L_n\}$ is a model for N_α

Instantiate $(N \uplus \{C \vee A, D \vee \neg B\}, M) \Rightarrow_{\text{IGEN}} (N \uplus \{C \vee A, D \vee \neg B, (C \vee A)\sigma, (D \vee \neg B)\sigma\}, \top)$
where $M = \{L_1, \ldots, L_n\}$, $\sigma = \text{mgu}(A, B)$, and σ is a proper instantiator of A or B

It is important that the grounding of N_α is obtained by substituting the same constant α for all variables, for otherwise the calculus becomes incomplete. For example, the two unit clauses $P(x, y); \neg P(x, x)$ are unsatisfiable. A grounding $P(a, b); \neg P(a, a)$ results in the model $\mathcal{I}_{N_\alpha} = \{P(a, b); \neg P(a, a)\}$ but Instantiate is not applicable, because the unifier $\{x \mapsto y\}$ is not a proper instantiater for both literals.

The model M is actially not used in rule Instantiate. The proof of the therem below, however, shows that it is sufficient to consider a minimal false clause $C \vee A$ or $D \vee \neg B$ with respect to $\mathcal{I}_N$, for the inference.

Theorem 3.16.4 (Completeness of InstGen)**.** Let $(N, \top) \Rightarrow^*_{\text{IGEN}} (N', M)$ and let (N', M) be a final state. If N is satisfiable then $M \neq \bot$ and $\mathcal{I}_{N'} \models N'$.

Proof. Suppose $\mathcal{I}_{N'}$ is not a model for N'. Then there exists a minimal ground closure $C \cdot \gamma$ such that $\mathcal{I}_{N'} \not\models C\gamma$. Obviously, $C \cdot \gamma$ was not productive in $\mathcal{I}_{N'}$. So there is no literal $L \in C$ such that $L\gamma$ is undefined in $\mathcal{I}_{C\cdot\gamma}$ and $L_\alpha \in$

$\mathcal{I}_{N_\alpha}$. However, $K_\alpha \in \mathcal{I}_{N_\alpha}$ for some $K \in C$ because $\mathcal{I}_{N_\alpha} \models N_\alpha$. So $C = C' \vee K$ and $\text{comp}(K\gamma) \in \mathcal{I}_{C\cdot\gamma}$. Therefore, there is some ground instance $D\sigma = (D' \vee K')\sigma$ that produces $\text{comp}(K\gamma)$ in $\mathcal{I}_{C\cdot\gamma}$, i.e., $\text{comp}(K')\sigma = K\gamma$. Let $\rho = \text{mgu}(K, \text{comp}(K'))$ and γ' be a substitution such that $(D' \vee K')\rho\gamma' = D\sigma$ and $(C' \vee K)\rho\gamma' = C\gamma$.

Firstly, ρ is a proper instatiator for K' or K. Assume not, then $K_\alpha = \text{comp}(K'_\alpha)$ so both $K'_\alpha \in \mathcal{I}_{N_\alpha}$ and $\text{comp}(K'_\alpha) \in \mathcal{I}_{N_\alpha}$, a contradiction.

Therefore, secondly, ρ is a proper instatiator for K' or K leading to two cases. If ρ is a proper instatiator for K', i.e., it instantiates a variable in K' with some constant then $(D' \vee K')\rho \cdot \gamma' \prec (D' \vee K') \cdot \sigma$ contradicting that N' is saturated or $(D' \vee K') \cdot \sigma$ is a minimal representation.

If ρ is a proper instatiator for K, it instantiates a variable in K with some constant then $(C' \vee K)\rho \cdot \gamma' \prec (C' \vee K) \cdot \gamma$ contradicting that that N' is saturated or $C \cdot \gamma$ was a minimal ground closure. □

Redundancy can be defined analogoulsy to superposition as well. A ground closure $C \cdot \sigma$ is *redundant* in a clause set N, if there are closures $C_1 \cdot \sigma_1, \ldots, C_n \cdot \sigma_n$ from clauses $C_1, \ldots, C_n$ from N such that $C_i \cdot \sigma_i \prec C \cdot \sigma$ for all i and $C_1\sigma_1, \ldots, C_n\sigma_n \models C\sigma$. A clause C from N is *redundant* if all its ground closures $C \cdot \sigma$ are redundant.

3.17 First-Order Logic Theories

In Section 3.2 the semantics of a first-order formula is defined with respect to all algebras that assign meaning to the symbols of the signature. For many applications this is too crude. For example, let us assume we consider the signature of simple linear integer arithmetic without divisibility relations, $\Sigma_{\text{LIA}} = (\{\text{LIA}\}, \{0, 1, +, -\} \cup \mathbb{Z}, \{\leq, <, >, \geq\})$, see also Section 6.2.4. Then a standard first-order algebra $\mathcal{A}$ is, e.g., $\text{LIA}^{\mathcal{A}} = \{0, 1\}$, $0^{\mathcal{A}} = 0$, $1^{\mathcal{A}} = 1$, $k^{\mathcal{A}} = (|k| \mod 2)$ for all $k \in \mathbb{Z}$, $+^{\mathcal{A}}(0,0) = 0$, $+^{\mathcal{A}}(1,0) = +^{\mathcal{A}}(0,1) = +^{\mathcal{A}}(1,1) = 1$, $-^{\mathcal{A}}(0,0) = -^{\mathcal{A}}(1,1) = -^{\mathcal{A}}(0,1) = 0$, $-^{\mathcal{A}}(1,0) = 1$, and the relations $\leq$, $<$, $>$, $\geq$ are interpreted as usual over the domain $\{0, 1\}$. Obviously, $\mathcal{A}$ is not the standard interpretation of linear integer arithmetic, because the domain is not the integers, and , e.g., $\mathcal{A} \models 8 < 9$ but also $\mathcal{A} \models 10 < 9$.

Is there a way to fix the semantics to the intended interpretation? Actually, there are two: the syntactic way by requiring any algebra $\mathcal{A}$ of the signature Σ_{LIA} to satisfy a set of closed first-order formulas, called *axioms*, or the semantic way of fixing a set of algebras for Σ_{LIA}. In both cases, the set of algebras and axioms is a called a *theory* $\mathcal{T}$. For both cases I assume that the axioms are satisfiable and there is either at least on algebra in $\mathcal{T}$, respectively.

For the above example, the semantic way would be simply to fix the standard linear integer interpretation for $\mathcal{T} = \{\Sigma_{\text{LIA}}\}$ as the only algebra to be considered. The syntactic way would mean to add enough formulas such that any algebra satisfying the formulas is the desired algebra. More concretely, the formulas

$$\mathcal{T} = \{\{k \not\approx l \mid \text{for all } k, l \in \mathbb{Z},\ k \neq l\} \cup \{k < l \mid \text{for all } k, l \in \mathbb{Z},\ k < l\}\}$$

Note, that the right hand side $\neq$ and $<$ are the standard relations on the integers. For any algebra $\mathcal{A}$ satisfying the infinitely many axioms of $\mathcal{T}$, $\mathcal{A} \models 8 < 9$ and $\mathcal{A} \models 9 < 10$ and $\text{LIA}^{\mathcal{A}}$ will contain at least as many different elements as the integers. So $\text{LIA}^{\mathcal{A}} = \mathbb{Z}$ is a possible domain of an algebra for $\mathcal{T}$, but also $\text{LIA}^{\mathcal{A}} = \mathbb{Q}$ would satisfy the above axioms.

Fixing a set of algebras is actually the more general and powerful mechanism. However, it has also disadvantages. Given a finite set of axioms $\mathcal{T}$ proving with respect to $\mathcal{T}$ amounts to classical first-order theorem proving, e.g., validity is semi-decidable. Given a set $\mathcal{T}$ of algebras, proving with respect to the algebras is typically beyond first-order logic theorem proving, e.g., for $\mathcal{T} = \{\Sigma_{\text{LIA}}\}$ theorem proving means inductive theorem proving, in general, hence, validity is no longer semi-decidable, but undecidable.

Definition 3.17.1 (First-Order Logic Theory)**.** Given a first-order many-sorted signature Σ, a *theory* $\mathcal{T}$ is a non-empty set of Σ-algebras.

For some first-order formula ϕ over Σ we say that ϕ is $\mathcal{T}$-*satisfiable* if there is some $\mathcal{A} \in \mathcal{T}$ such that $\mathcal{A}(\beta) \models \phi$ for some β. We say that ϕ is $\mathcal{T}$-*valid* ($\mathcal{T}$-*unsatisfiable*) if for all $\mathcal{A} \in \mathcal{T}$ and all β it holds $\mathcal{A}(\beta) \models \phi$ ($\mathcal{A}(\beta) \not\models \phi$). In case of validity I also write $\models_{\mathcal{T}} \phi$.

Alternatively, $\mathcal{T}$ may contain a set of satisfiable axioms which then stand for all algebras satisfying the axioms.

The Σ-algebras can be restricted to term-generated models as long as there are "enough" constants (function) symbols in Σ, in general infinitely many are sufficient. Due to the Löwenheim-Skolem theorem different infinite cardinalities cannot be distinguished by first-order formulas.

Chapter 4

Equational Logic

From now on First-order Logic is considered with equality. In this chapter, I investigate properties of a set of unit equations. For a set of unit equations I write E. Full first-order clauses with equality are studied in Chapter 5. I recall certain definitions from Section 1.6 and Chapter 3.

The main reasoning problem considered in this chapter is given a set of unit equations E and an additional equation $s \approx t$, does $E \models s \approx t$ hold? As usual, all variables are implicitely universally quantified. The idea is to turn the equations E into a convergent term rewrite system (TRS) R such that the above problem can be solved by checking identity of the respective normal forms: $s \downarrow_R = t \downarrow_R$. Showing $E \models s \approx t$ is as difficult as proving validity of any first-order formula, see Section 3.15.

For example consider the euqational ground clauses $E = \{g(a) \approx b, a \approx b\}$ over a signature consisting of the constants a, b and unary function g, all defined over some unique sort. Then for all algebras $\mathcal{A}$ satifying E, all ground terms over a, b, and g, are mapped to the same domain element. In particular, it holds $E \models g(b) \approx b$. Now the idea is to turn E into a convergent term rewrite system R such that $g(b) \downarrow_R = b \downarrow_R$. To this end, the equations in E are oriented, e.g., a first guess might be the TRS $R_0 = \{g(a) \to b, a \to b\}$. For R_0 we get $g(b) \downarrow_{R_0} = g(b)$, $b \downarrow_{R_0} = b$, so not the desired result. The TRS R_0 is not confluent an all ground terms, because $g(a) \to_{R_0} b$ and $g(a) \to_{R_0} g(b)$, but b and $g(b)$ are R_0 normal forms. This problem can be repaired by adding the extra rule $g(b) \to b$ and this process is called *completion* and is studied in this chapter. Now the extended rewrite system $R_1 = \{g(a) \to b, a \to b, g(b) \to b\}$ is convergent and $g(b) \downarrow_{R_1} = b \downarrow_{R_1} = b$. Termination can be shown by using a KBO (or LPO) with precedence $g \succ a \succ b$. Then the left hand sides of the rules are strictly larger than the right hand sides. Actually, R_1 contains some redundancy, even removing the first rewrite rule $g(a) \to b$ from R_1 does not violate confluence. Detecting redundant rules is also discussed in this chapter.

Definition 4.0.1 (Equivalence Relation, Congruence Relation)**.** An *equivalence* relation $\sim$ on a term set $T(\Sigma, \mathcal{X})$ is a reflexive, transitive, symmetric binary

relation on $T(\Sigma, \mathcal{X})$ such that if $s \sim t$ then $\text{sort}(s) = \text{sort}(t)$.

Two terms s and t are called *equivalent*, if $s \sim t$.

An equivalence $\sim$ is called a *congruence* if $s \sim t$ implies $u[s] \sim u[t]$, for all terms $s, t, u \in T(\Sigma, \mathcal{X})$. Given a term $t \in T(\Sigma, \mathcal{X})$, the set of all terms equivalent to t is called the *equivalence class of t by $\sim$*, denoted by $[t]_\sim := \{t' \in T(\Sigma, \mathcal{X}) \mid t' \sim t\}$.

If the matter of discussion does not depend on a particular equivalence relation or it is unambiguously known from the context, $[t]$ is used instead of $[t]_\sim$. The above definition is equivalent to Definition 3.2.3.

The set of all equivalence classes in $T(\Sigma, \mathcal{X})$ defined by the equivalence relation is called a *quotient by* $\sim$, denoted by $T(\Sigma, \mathcal{X})|_\sim := \{[t] \mid t \in T(\Sigma, \mathcal{X})\}$. Let E be a set of equations then $\sim_E$ denotes the smallest congruence relation "containing" E, that is, $(l \approx r) \in E$ implies $l \sim_E r$. The equivalence class $[t]_{\sim_E}$ of a term t by the equivalence (congruence) $\sim_E$ is usually denoted, for short, by $[t]_E$. Likewise, $T(\Sigma, \mathcal{X})|_E$ is used for the quotient $T(\Sigma, \mathcal{X})|_{\sim_E}$ of $T(\Sigma, \mathcal{X})$ by the equivalence (congruence) $\sim_E$.

4.1 Term Rewrite System

I instantiate the abstract rewrite systems of Section 1.6 with first-order terms. The main difference is that rewriting takes not only place at the top position of a term, but also at inner positions.

Definition 4.1.1 (Rewrite Rule, Term Rewrite System)**.** A *rewrite rule* is an equation $l \approx r$ between two terms l and r so that l is not a variable and $vars(l) \supseteq vars(r)$. A *term rewrite system* R, or a TRS for short, is a set of rewrite rules.

Definition 4.1.2 (Rewrite Relation)**.** Let E be a set of (implicitly universally quantified) equations, i.e., unit clauses containing exactly one positive equation. The *rewrite relation* $\to_E \subseteq T(\Sigma, \mathcal{X}) \times T(\Sigma, \mathcal{X})$ is defined by

$$s \to_E t \quad \text{iff} \quad \begin{array}{l}\text{there exist } (l \approx r) \in E, p \in pos(s),\\ \text{and matcher } \sigma, \text{ so that } s|_p = l\sigma \text{ and } t = s[r\sigma]_p.\end{array}$$

Note that in particular for any equation $l \approx r \in E$ it holds $l \to_E r$, so the equation can also be written $l \to r \in E$.

Often $s = t \downarrow_R$ is written to denote that s is a normal form of t with respect to the rewrite relation $\to_R$. Notions $\to_R^0, \to_R^+, \to_R^*, \leftrightarrow_R^*$, etc. are defined accordingly, see Section 1.6. An instance of the left-hand side of an equation is called a *redex* (reducible expression). *Contracting* a redex means replacing it with the corresponding instance of the right-hand side of the rule. A term rewrite system R is called *convergent* if the rewrite relation $\to_R$ is confluent and terminating. A set of equations E or a TRS R is terminating if the rewrite relation $\to_E$ or $\to_R$ has this property. Furthermore, if E is terminating then it is a TRS. A rewrite system is called *right-reduced* if for all rewrite rules $l \to r$

in R, the term r is irreducible by R. A rewrite system R is called *left-reduced* if for all rewrite rules $l \to r$ in R, the term l is irreducible by $R \backslash \{l \to r\}$. A rewrite system is called *reduced* if it is left- and right-reduced.

Lemma 4.1.3 (Left-Reduced TRS)**.** Left-reduced terminating rewrite systems are convergent. Convergent rewrite systems define unique normal forms.

Lemma 4.1.4 (TRS Termination)**.** A rewrite system R terminates iff there exists a reduction ordering $\succ$ so that $l \succ r$, for each rule $l \to r$ in R.

4.1.1 E-Algebras

Let E be a set of universally quantified equations. A model $\mathcal{A}$ 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, this is also denoted with $s \approx_E t$. The goal is to 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 (well-sorted) equations over $T(\Sigma, \mathcal{X})$ where all variables are implicitly universally quantified. The following inference system allows to derive consequences of E:

Reflexivity $E \;\Rightarrow_{\mathrm{E}}\; E \cup \{t \approx t\}$

Symmetry $E \uplus \{t \approx t'\} \;\Rightarrow_{\mathrm{E}}\; E \cup \{t \approx t'\} \cup \{t' \approx t\}$

Transitivity $E \uplus \{t \approx t', t' \approx t''\} \;\Rightarrow_{\mathrm{E}}\; E \cup \{t \approx t', t' \approx t''\} \cup \{t \approx t''\}$

Congruence $E \uplus \{t_1 \approx t_1', \ldots, t_n \approx t_n'\} \;\Rightarrow_{\mathrm{E}}\; E \cup \{t_1 \approx t_1', \ldots, t_n \approx t_n'\} \cup \{f(t_1, \ldots, t_n) \approx f(t_1', \ldots, t_n')\}$
for any function $f : \mathrm{sort}(t_1) \times \ldots \times \mathrm{sort}(t_n) \to S$ for some S

Instance $E \uplus \{t \approx t'\} \;\Rightarrow_{\mathrm{E}}\; E \cup \{t \approx t'\} \cup \{t\sigma \approx t'\sigma\}$
for any well-sorted substitution σ

Lemma 4.1.5 (Equivalence of $\leftrightarrow_E^*$ and $\Rightarrow_E^*$)**.** The following properties are equivalent:

1. $s \leftrightarrow_E^* t$
2. $E \Rightarrow_E^* s \approx t$ is derivable.

where $E \Rightarrow_E^* s \approx t$ is an abbreviation for $E \Rightarrow_E^* E'$ and $s \approx t \in E'$.

Proof. (i)⇒(ii): $s \leftrightarrow_E t$ implies $E \Rightarrow_E^* s \approx t$ by induction on the depth of the position where the rewrite rule is applied; then $s \leftrightarrow_E^* t$ implies $E \Rightarrow_E^* s \approx t$ 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 $E \Rightarrow_E^* s \approx t$. □

Corollary 4.1.6 (Convergence of E)**.** If a set of equations E is convergent then $s \approx_E t$ if and only if $s \leftrightarrow^* t$ if and only if $s \downarrow_E = t \downarrow_E$.

Corollary 4.1.7 (Decidability of $\approx_E$)**.** If a set of equations E is finite and convergent then $\approx_E$ is decidable.

The above Lemma 4.1.5 shows equivalence of the syntactically defined relations $\leftrightarrow_E^*$ and $\Rightarrow_E^*$. What is missing, in analogy to Herbrand's theorem for first-order logic without equality Theorem 3.5.5, is a semantic characterization of the relations by a particular algebra.

Definition 4.1.8 (Quotient Algebra)**.** For sets of unit equations this is a *quotient algebra*: Let X be a set of variables. For $t \in T(\Sigma, \mathcal{X})$ let $[t] = \{t' \in T(\Sigma, \mathcal{X})) \mid E \Rightarrow_{\mathrm{E}}^* t \approx t'\}$ be the *congruence class* of t. Define a Σ-algebra $\mathcal{I}_E$, called the *quotient algebra*, technically $T(\Sigma, \mathcal{X})/E$, as follows: $S^{\mathcal{I}_E} = \{[t] \mid t \in T_S(\Sigma, \mathcal{X})\}$ for all sorts S and $f^{\mathcal{I}_E}([t_1], \ldots, [t_n]) = [f(t_1, \ldots, t_n)]$ for $f : \mathrm{sort}(t_1) \times \ldots \times \mathrm{sort}(t_n) \to T \in \Omega$ for some sort T.

Lemma 4.1.9 ($\mathcal{I}_E$ is an E-algebra)**.** $\mathcal{I}_E = T(\Sigma, \mathcal{X})/E$ is an E-algebra.

Proof. Firstly, all functions $f^{\mathcal{I}_E}$ are well-defined: if $[t_i] = [t_i']$, then $[f(t_1, \ldots, t_n)] = [f(t_1', \ldots, t_n')]$. This follows directly from the Congruence rule for $\Rightarrow^*$.

Secondly, let $\forall x_1 \ldots x_n (s \approx t)$ be an equation in E. Let β be an arbitrary assignment. It has to be shown that $\mathcal{I}_E(\beta)(\forall \vec{x}(s \approx t)) = 1$, or equivalently, that $\mathcal{I}_E(\gamma)(s) = \mathcal{I}_E(\gamma)(t)$ for all $\gamma = \beta[x_i \mapsto [t_i] \mid 1 \leq i \leq n]$ with $[t_i] \in \mathrm{sort}(x_i)^{\mathcal{I}_E}$. Let $\sigma = \{x_1 \mapsto t_1, \ldots, x_n \mapsto t_n\}$, with $t_i \in T_{\mathrm{sort}(x_i)}(\Sigma, \mathcal{X})$, then $s\sigma \in \mathcal{I}_E(\gamma)(s)$ and $t\sigma \in \mathcal{I}_E(\gamma)(t)$. By the Instance rule, $E \Rightarrow^* s\sigma \approx t\sigma$ is derivable, hence $\mathcal{I}_E(\gamma)(s) = [s\sigma] = [t\sigma] = \mathcal{I}_E(\gamma)(t)$. □

Lemma 4.1.10 ($\Rightarrow_E$ is complete)**.** Let $\mathcal{X}$ be a countably infinite set of variables; let $s, t \in T_S(\Sigma, \mathcal{X})$. If $\mathcal{I}_E \models \forall \vec{x}(s \approx t)$, then $E \Rightarrow_E^* s \approx t$ is derivable.

Proof. Assume that $\mathcal{I}_E \models \forall \vec{x}(s \approx t)$, i.e., $\mathcal{I}_E(\beta)(\forall \vec{x}(s \approx t)) = 1$. Consequently, $\mathcal{I}_E(\gamma)(s) = \mathcal{I}_E(\gamma)(t)$ for all $\gamma = \beta[\, x_i \mapsto [t_i] \mid 1 \leq i \leq n\,]$ with $[t_i] \in \mathrm{sort}(x_i)^{\mathcal{I}_E}$. Choose $t_i = x_i$, then $[s] = \mathcal{I}_E(\gamma)(s) = \mathcal{I}_E(\gamma)(t) = [t]$, so $E \Rightarrow^* s \approx t$ is derivable by definition of $\mathcal{I}_E$. □

Theorem 4.1.11 (Birkhoff's Theorem)**.** Let $\mathcal{X}$ be a countably infinite set of variables, let E be a set of (universally quantified) equations. Then the following properties are equivalent for all $s, t \in T_S(\Sigma, \mathcal{X})$:

1. $s \leftrightarrow_E^* t$.

2. $E \Rightarrow_E^* s \approx t$ is derivable.

3. $s \approx_E t$, i.e., $E \models \forall\vec{x}(s \approx t)$.

4. $\mathcal{I}_E \models \forall\vec{x}(s \approx t)$.

Proof. (1.)⇔(2.): Lemma 4.1.5.
(2.)⇒(3.): By induction on the size of the derivation for $E \Rightarrow^* s \approx t$.
(3.)⇒(4.): Obvious, since $\mathcal{I}_E = T(\Sigma, \mathcal{X})/E$ is an E-algebra.
(4.)⇒(2.): Lemma 4.1.10. □

Universal Algebra
$T(\Sigma, \mathcal{X})/E = T(\Sigma, \mathcal{X})/{\approx_E} = T(\Sigma, \mathcal{X})/{\leftrightarrow_E^*}$ is called the *free E-algebra* with generating set $\mathcal{X}/{\approx_E} = \{[x] \mid x \in \mathcal{X}\}$: Every mapping $\phi : \mathcal{X}/{\approx_E} \to \mathcal{B}$ for some E-algebra $\mathcal{B}$ can be extended to a homomorphism $\hat{\phi} : T(\Sigma, \mathcal{X})/E \to \mathcal{B}$.
$T(\Sigma, \emptyset)/E = T(\Sigma, \emptyset)/{\approx_E} = T(\Sigma, \emptyset)/{\leftrightarrow_E^*}$ is called the *initial E-algebra.*
${\approx_E} = \{(s, t) \mid E \models s \approx t\}$ is called the *equational theory* of E.
${\approx_E^I} = \{(s, t) \mid T(\Sigma, \emptyset)/E \models s \approx t\}$ is called the *inductive theory* of E.

Example 4.1.12. Let $E = \{\forall x(x + 0 \approx x),\ \forall x\forall y(x + s(y) \approx s(x + y))\}$. Then $x + y \approx_E^I y + x$, but $x + y \not\approx_E y + x$.

4.2 Critical Pairs

By Theorem 4.1.11 the semantics of E and ${\leftrightarrow_E^*}$ conincide. In order to decide ${\leftrightarrow_E^*}$ we need to turn ${\rightarrow_E^*}$ in a confluent and terminating relation. If ${\leftrightarrow_E^*}$ is terminating then confluence is equivalent to local confluence, see Newman's Lemma, Lemma 1.6.6. Local confluence is the following problem for TRS: if $t_1\ {}_E\!\leftarrow t_0 \rightarrow_E t_2$, does there exist a term s so that $t_1 \rightarrow_E^* s\ {}_E^*\!\leftarrow t_2$? If the two rewrite steps happen in different subtrees (disjoint redexes) then a repitition of the respective other step yields the common term s. If the two rewrite steps happen below each other (overlap at or below a variable position) again a repetition of the respective other step yields the common term s. If the left-hand sides of the two rules overlap at a non-variable position there is no ovious way to generate s.

More technically two rewrite rules $l_1 \to r_1$ and $l_2 \to r_2$ overlap if there exist some non-variable subterm $l_1|_p$ such that l_2 and $l_1|_p$ have a common instance $(l_1|_p)\sigma_1 = l_2\sigma_2$. If the two rewrite rules do not have common variables, then only a single substitution is necessary, the mgu σ of $(l_1|_p)$ and l_2.

Definition 4.2.1 (Critical Pair)**.** Let $l_i \to r_i$ $(i = 1, 2)$ be two rewrite rules in a TRS R whithout common variables, i.e., $\mathrm{vars}(l_1) \cap \mathrm{vars}(l_2) = \emptyset$. Let $p \in \mathrm{pos}(l_1)$ be a position so 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 $r_1\sigma \downarrow_R (l_1\sigma)[r_2\sigma]_p$.

Recall that $\mathrm{vars}(l_i) \supseteq \mathrm{vars}(r_i)$ for the two rewrite rules by Definition 4.1.1.

Theorem 4.2.2 ("Critical Pair Theorem")**.** A TRS R is locally confluent iff all its critical pairs are joinable.

Proof. ($\Rightarrow$) Obvious, since joinability of a critical pair is a special case of local confluence.

($\Leftarrow$) Suppose s rewrites to t_1 and t_2 using rewrite rules $l_i \to r_i \in R$ at positions $p_i \in \mathrm{pos}(s)$, where $i = 1, 2$. The two rules are variable disjoint, hence $s|_{p_i} = l_i\sigma$ and $t_i = s[r_i\sigma]_{p_i}$. There are two cases to be considered:

1. Either p_1 and p_2 are in disjoint subtrees ($p_1 \parallel p_2$) or
2. one is a prefix of the other (w.l.o.g., $p_1 \leq p_2$).

Case 1: $p_1 \parallel p_2$. Then $s = s[l_1\sigma]_{p_1}[l_2\sigma]_{p_2}$, and therefore $t_1 = s[r_1\sigma]_{p_1}[l_2\sigma]_{p_2}$ and $t_2 = s[l_1\sigma]_{p_1}[r_2\sigma]_{p_2}$. Let $t_0 = s[r_1\sigma]_{p_1}[r_2\sigma]_{p_2}$. 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 σ', where $\sigma' = \sigma[x \mapsto (x\sigma)[r_2\sigma]_{q_2}]$.
Case 2.2: $p_2 = p_1p$, where p is a non-variable position of l_1. Then $s|_{p_2} = l_2\sigma$ and $s|_{p_2} = (s|_{p_1})|_p = (l_1\sigma)|_p = (l_1|_p)\sigma$, so σ is a unifier of l_2 and $l_1|_p$.Let σ' be the mgu of l_2 and $l_1|_p$, then $\sigma = \tau \circ \sigma'$ 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, $t_1 = s[r_1\sigma]_{p_1} = s[r_1\sigma'\tau]_{p_1} \to_R^* s[v\tau]_{p_1}$ and $t_2 = s[r_2\sigma]_{p_2} = s[(l_1\sigma)[r_2\sigma]_p]_{p_1} = s[(l_1\sigma'\tau)[r_2\sigma'\tau]_p]_{p_1} = s[((l_1\sigma')[r_2\sigma']_p)\tau]_{p_1} \to_R^* s[v\tau]_{p_1}$. □

Please note that critical pairs between a rule and (a renamed variant of) itself must be considered, except if the overlap is at the root, i.e., $p = \epsilon$, because this critical pair always joins.

Corollary 4.2.3. A terminating TRS R is confluent if and only if all its critical pairs are joinable.

Proof. By the Theorem 4.2.2 and because every locally confluent and terminating relation $\to$ is confluent, Newman's Lemma, Lemma 1.6.6. □

Corollary 4.2.4. For a finite terminating TRS, confluence is decidable.

Proof. For every pair of rules and every non-variable position in the first rule there is at most one critical pair $\langle u_1, u_2\rangle$. Reduce every u_i to some normal form u_i'. If $u_1' = u_2'$ for every critical pair, then R is confluent, otherwise there is some non-confluent situation $u_1' \ {}_R^*\!\leftarrow u_1 \leftarrow_R s \to_R u_2 \to_R^* u_2'$. □

4.3 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 4.3.1. Both termination problems for TRSs are undecidable in general.

Proof. Encode Turing machines (TM) 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 Different Scenarios

Depending on the application, the TRS whose termination has to be shown can be

1. fixed and known in advance, or
2. evolving (e.g., generated by some saturation process).

Methods for case 2. are also usable for case 1.. Many methods for case 1. are not usable for case 2..

First consider case 2., additional techniques for case 1. will be considered later.

Reduction Orderings

Goal: Given a finite TRS R, show termination of R by looking at finitely many rules $l \to r \in R$, rather than at infinitely many possible replacement steps $s \to_R s'$.

A binary relation $\sqsupset$ over $T(\Sigma, \mathcal{X})$ is called *compatible with Σ-operations,* if $s \sqsupset s'$ implies $f(t_1, \ldots, s, \ldots, t_n) \sqsupset f(t_1, \ldots, s', \ldots, t_n)$ for all $f \in \Omega$ and $s, s', t_i \in T(\Sigma, \mathcal{X})$.

Lemma 4.3.2. 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 $s, s', t \in T(\Sigma, \mathcal{X})$ and $p \in pos(t)$.

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

A binary relation $\sqsupset$ over $T(\Sigma, \mathcal{X})$ is called *stable under substitutions,* if $s \sqsupset s'$ implies $s\sigma \sqsupset s'\sigma$ for all $s, s' \in T(\Sigma, \mathcal{X})$ and substitutions σ. A binary relation $\sqsupset$ is called a *rewrite relation,* if it is compatible with Σ-operations and stable under substitutions.

Example 4.3.3. If R is a TRS, then $\to_R$ is a rewrite relation.

A strict partial ordering over $T(\Sigma, \mathcal{X})$ that is a rewrite relation is called *rewrite ordering.* A well-founded rewrite ordering is called *reduction ordering.*

Theorem 4.3.4. A TRS R terminates if and only if there exists a reduction ordering $\succ$ so that $l \succ r$ for every rule $l \to r \in R$.

Proof. ($\Rightarrow$): $s \to_R s'$ 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.

($\Leftarrow$): 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 $T(\Sigma, \mathcal{X})$ by $s \succ_\mathcal{A} t$ iff $\mathcal{A}(\beta)(s) \succ \mathcal{A}(\beta)(t)$ for all assignments $\beta : \mathcal{X} \to U_\mathcal{A}$. Is $\succ_\mathcal{A}$ a reduction ordering?

Lemma 4.3.5. $\succ_\mathcal{A}$ 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 $\beta : \mathcal{X} \to U_\mathcal{A}$. Let σ be a substitution. It has to be shown that $\mathcal{A}(\gamma)(s\sigma) \succ \mathcal{A}(\gamma)(s'\sigma)$ for all assignments $\gamma : \mathcal{X} \to U_\mathcal{A}$. Choose $\beta = \gamma \circ \sigma$, then by the substitution lemma, $\mathcal{A}(\gamma)(s\sigma) = \mathcal{A}(\beta)(s) \succ \mathcal{A}(\beta)(s') = \mathcal{A}(\gamma)(s'\sigma)$. Therefore $s\sigma \succ_\mathcal{A} s'\sigma$. □

A function $f : U_\mathcal{A}^n \to U_\mathcal{A}$ is called *monotone* w.r.t. $\succ$), if $a \succ a'$ implies $f(b_1, \ldots, a, \ldots, b_n) \succ f(b_1, \ldots, a', \ldots, b_n)$ for all $a, a', b_i \in U_\mathcal{A}$.

Lemma 4.3.6. If the interpretation $f_\mathcal{A}$ of every function symbol f is monotone w.r.t. $\succ$, then $\succ_\mathcal{A}$ is compatible with Σ-operations.

Proof. Let $s \succ s'$, that is, $\mathcal{A}(\beta)(s) \succ \mathcal{A}(\beta)(s')$ for all $\beta : \mathcal{X} \to U_\mathcal{A}$. Let $\beta : \mathcal{X} \to U_\mathcal{A}$ 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 4.3.7. 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 $\mathcal{A}(\beta)(s_1) \succ \mathcal{A}(\beta)(s_2) \succ \ldots$ (with β chosen arbitrarily). Thus $\succ_\mathcal{A}$ is well-founded. Irreflexivity and transitivity are proved similarly. □

Polynomial Orderings

Polynomial orderings:

1. Instance of the interpretation method:
2. The carrier set $U_\mathcal{A}$ is $\mathbb{N}$ or some subset of $\mathbb{N}$.
3. To every function symbol f with arity n a polynomial $P_f(X_1, \ldots, X_n) \in \mathbb{N}[X_1, \ldots, X_n]$ with coefficients in $\mathbb{N}$ is associated and indeterminates $X_1, \ldots, X_n$. Then 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:

1. $U_{\mathcal{A}} = \{n \in \mathbb{N} \mid n \geq 1\}$.
2. If $arity(f) = 0$, then P_f is a constant ≥ 1.
3. If $arity(f) = n \geq 1$, then P_f is a polynomial $P(X_1, \ldots, X_n)$, so that every X_i occurs in some monomial with exponent at least 1 and non-zero coefficient. $\Rightarrow$ Requirements 1 and 2 are satisfied.

The mapping from function symbols to polynomials can be extended to terms: A term t containing the variables $x_1, \ldots, x_n$ yields a polynomial P_t with indeterminates $X_1, \ldots, X_n$ (where X_i corresponds to $\beta(x_i)$).

Example 4.3.8. Let $\Omega = \{b/0, f/1, g/3\}, P_b = 3, P_f(X_1) = X_1^2, P_g(X_1, X_2, X_3) = X_1 + X_2X_3$ and $t = g(f(b), f(x), y)$, then $P_t(X, Y) = 9 + X^2Y$.

If P, Q are polynomials in $\mathbb{N}[X_1, \ldots, X_n]$, $P > Q$ is written if $P(a_1, \ldots, a_n) > Q(a_1, \ldots, a_n)$ for all $a_1, \ldots, a_n \in U_{\mathcal{A}}$. Clearly, $l \succ_{\mathcal{A}} r$ iff $P_l > P_r$ iff $P_l - P_r > 0$. The question is whether $P_l - P_r > 0$ can be checked 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 4.3.9. Hilbert's 10th Problem is undecidable.

Proposition 4.3.10. Given a polynomial interpretation and two terms l, r, it is undecidable whether $P_l > P_r$.

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

One easy case: If restricted to linear polynomials, deciding whether $P_l - P_r > 0$ is trivial: $\sum k_i a_i + k > 0$ for all $a_1, \ldots, a_n \geq 1$ if and only if $k_i \geq 0$ for all $i \in \{1, \ldots, n\}$ and $\sum k_i + k > 0$.

Another possible solution: Test whether $P_l(a_1, \ldots, a_n) > P_r(a_1, \ldots, a_n)$ for all $a_1, \ldots, a_n \in \{x \in \mathbb{R} \mid x \geq 1\}$. This is decidable (but hard). Since $U_{\mathcal{A}} \subseteq \{x \in \mathbb{R} \mid x \geq 1\}$ this implies $P_l > P_r$.

Alternatively: Use fast overapproximations.

Simplification Orderings

The *proper subterm ordering* $\rhd$ is defined by $s \rhd t$ if and only if $s|_p = t$ for some position $p \neq \epsilon$ of s.

A rewrite ordering $\succ$ over $T(\Sigma, \mathcal{X})$ is called *simplification ordering* if it has the *subterm property:* $s \rhd t$ implies $s \succ t$ for all $s, t \in T(\Sigma, \mathcal{X})$.

Example 4.3.11. Let R_{emb} be the rewrite system $R_{emb} = \{ f(x_1, \ldots, x_n) \to x_i \mid f \in \Omega, 1 \leq i \leq n = f/n\}$. Define $\rhd_{emb} = \to^+_{R_{emb}}$ and $\trianglerighteq = \to^*_{R_{emb}}$ *("homeomorphic embedding relation")* and $\rhd_{emb}$ is a simplification ordering.

Lemma 4.3.12. If $\succ$ is a simplification ordering then $s \rhd_{emb} t$ implies $s \succ t$ and $s \trianglerighteq t$ implies $s \succeq t$.

Proof. Since $\succ$ is transitive and $\succeq$ is transitive and reflexive, it suffices to show that $s \to_{R_{emb}} t$ implies $s \succ t$. By definition, $s \to_{R_{emb}} t$ if and only if $s = s[l\sigma]$ and $t = s[r\sigma]$ for some rule $l \to r \in R_{emb}$. 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 4.3.13 ("Kruskal's Theorem")**.** Let Σ be a finite signature, let $\mathcal{X}$ be a finite set of variables. Then for every infinite sequence $t_1, t_2, t_3, \ldots$ there are indexes $j > i$ so that $t_j \trianglerighteq_{emb} t_i$. ($\trianglerighteq_{emb}$ is called a *well-partial-ordering (wpo).*)

Proof. The proof can be found in the book of Baader and Nipkow [**?**] pages 113–115. □

Theorem 4.3.14 (Dershowitz)**.** If Σ is a finite signature, then every simplification ordering $\succ$ on $T(\Sigma, \mathcal{X})$ is well-founded (and therefore a reduction ordering).

Proof. Suppose that $t_1 \succ t_2 \succ t_3 \succ \ldots$ is an infinite descending chain. First assume that there is an $x \in vars(t_{i+1}) \setminus vars(t_i)$. Let $\sigma = \{x \mapsto t_i\}$, then $t_{i+1}\sigma \trianglerighteq x\sigma = t_i$ and therefore $t_i = t_i\sigma \succ t_{i+1}\sigma \succeq t_i$, contradicting reflexivity.

Consequently, $vars(t_i) \supseteq vars(t_{i+1})$ and $t_i \in T(\Sigma, \mathcal{V})$ for all i, where $\mathcal{V}$ is the finite set $vars(t_1)$. By Kruskal's Theorem, there are $i < j$ with $t_i \trianglelefteq_{emb} t_j$. 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 4.3.15.

Let $R = \{f(f(x)) \to f(g(f(x)))\}$. R terminates and $\to_R^+$ is therefore a reduction ordering. Assume that $\to_R$ was 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 $f(g(f(x))) \trianglerighteq_{emb} f(f(x))$ implies $f(g(f(x))) \succeq f(f(x))$, hence $f(f(x)) \succ f(f(x))$.

4.4 Knuth-Bendix Completion (KBC)

Given a set E of equations, the goal of Knuth-Bendix completion is to transform E into an equivalent convergent set R of rewrite rules. If R is finite this yields a decision procedure for E. For ensuring termination the calculus fixes a reduction ordering $\succ$ and constructs R in such a way that $\to_R \subseteq \succ$, i.e., $l \succ r$ for every

$l \to r \in R$. For ensuring confluence the calculus checks whether all critical pairs are joinable.

The completion procedure itself is presented as a set of abstract rewrite rules working on a pair of equations E and rules R: $(E_0;R_0) \Rightarrow_{\mathrm{KBC}} (E_1;R_1) \Rightarrow_{\mathrm{KBC}} (E_1;R_2) \Rightarrow_{\mathrm{KBC}} \ldots$. The initial state is $(E_0, \emptyset)$ where $E = E_0$ contains the input equations. If $\Rightarrow_{\mathrm{KBC}}$ successfully terminates then E is empty and R is the convergent rewrite system for E_0. For each step $(E; R) \Rightarrow_{\mathrm{KBC}} (E'; R')$ the equational theories of $E \cup R$ and $E' \cup R'$ agree: $\approx_{E \cup R} = \approx_{E' \cup R'}$. By $\mathrm{cp}(R)$ I denote the set of critical pairs between rules in R.

Orient $(E \uplus \{s \stackrel{\cdot}{\approx} t\}; R) \Rightarrow_{\mathrm{KBC}} (E; R \cup \{s \to t\})$
if $s \succ t$

Delete $(E \uplus \{s \approx s\}; R) \Rightarrow_{\mathrm{KBC}} (E; R)$

Deduce $(E; R) \Rightarrow_{\mathrm{KBC}} (E \cup \{s \approx t\}; R)$
if $\langle s, t\rangle \in \mathrm{cp}(R)$

Simplify-Eq $(E \uplus \{s \stackrel{\cdot}{\approx} t\}; R) \Rightarrow_{\mathrm{KBC}} (E \cup \{u \approx t\}; R)$
if $s \to_R u$

R-Simplify-Rule $(E; R \uplus \{s \to t\}) \Rightarrow_{\mathrm{KBC}} (E; R \cup \{s \to u\})$
if $t \to_R u$

L-Simplify-Rule $(E; R \uplus \{s \to t\}) \Rightarrow_{\mathrm{KBC}} (E \cup \{u \approx t\}; R)$
if $s \to_R u$ using a rule $l \to r \in R$ so that $s \sqsupset l$, see below.

Trivial equations cannot be oriented and since they are not needed they can be deleted by the Delete rule. The rule Deduce turns critical pairs between rules in R into additional equations. Note that if $\langle s, t\rangle \in \mathrm{cp}(R)$ then $s_R \leftarrow u \to_R t$ and hence $R \models s \approx t$. The simplification rules are not needed but serve as reduction rules, removing redundancy from the state. Simplification of the left-hand side may influence orientability and orientation of the result. Therefore, it yields an equation. For technical reasons, the left-hand side of $s \to t$ may only be simplified using a rule $l \to r$, if $l \to r$ cannot be simplified using $s \to t$, that is, if $s \sqsupset l$, where the *encompassment quasi-ordering* $\sqsupseteq$ is defined by $s \sqsupseteq l$ if $s|_p = l\sigma$ for some p and σ and $\sqsupset = \sqsupseteq \setminus \sqsubseteq$ is the strict part of $\sqsupseteq$.

Lemma 4.4.1. $\sqsupset$ is a well-founded strict partial ordering.

Lemma 4.4.2. If $(E; R) \Rightarrow_{KBC} (E'; R')$, then $\approx_{E \cup R} = \approx_{E' \cup R'}$.

Lemma 4.4.3. If $(E; R) \Rightarrow_{KBC} (E'; R')$ and $\to_R \subseteq \succ$, then $\to_{R'} \subseteq \succ$.

Proposition 4.4.4 (Knuth-Bendix Completion Correctness)**.** If the completion procedure on a set of equations E is run, different things can happen:

1. A state where no more inference rules are applicable is reached and E is not empty. $\Rightarrow$ Failure (try again with another ordering?)
2. A state where E is empty is reached and all critical pairs between the rules in the current R have been checked.
3. The procedure runs forever.

In order to treat these cases simultaneously some definitions are needed:

Definition 4.4.5 (Run)**.** A (finite or infinite) sequence $(E_0; R_0) \Rightarrow_{KBC} (E_1; R_1) \Rightarrow_{KBC} (E_2; R_2) \Rightarrow_{KBC} \ldots$ with $R_0 = \emptyset$ is called a *run* of the completion procedure with input E_0 and $\succ$. For a run, $E_\infty = \bigcup_{i\geq 0} E_i$ and $R_\infty = \bigcup_{i\geq 0} R_i$.

Definition 4.4.6 (Persistent Equations)**.** The sets of *persistent equations of rules* of the run are $E_* = \bigcup_{i\geq 0} \bigcap_{j\geq i} E_j$ and $R_* = \bigcup_{i\geq 0} \bigcap_{j\geq i} R_j$.

Note: If the run is finite and ends with E_n, R_n then $E_* = E_n$ and $R_* = R_n$.

Definition 4.4.7 (Fair Run)**.** A run is called *fair* if $\mathrm{CP}(R_*) \subseteq E_\infty$ (i.e., if every critical pair between persisting rules is computed at some step of the derivation).

Goal: Show: If a run is fair and E_* is empty then R_* is convergent and equivalent to E_0. In particular: If a run is fair and E_* is empty then $\approx_{E_0} = \approx_{E_\infty \cup R_\infty} = \leftrightarrow^*_{E_\infty \cup R_\infty} = \downarrow_{R_*}$.

From now on, $(E_0; R_0) \Rightarrow_{KBC} (E_1; R_1) \Rightarrow_{KBC} (E_2; R_2) \Rightarrow_{KBC} \ldots$ is a fair run and R_0 and E_* are empty.

A *proof* of $s \approx t$ in $E_\infty \cup R_\infty$ is a finite sequence $(s_0, \ldots, s_n)$ so that $s = s_0, t = s_n$ and for all $i \in \{1, \ldots, n\}$ it holds:

1. $s_{i-1} \leftrightarrow_{E_\infty} s_i$ or
2. $s_{i-1} \rightarrow_{R_\infty} s_i$ or
3. $s_{i-1}\ {}_{R_\infty}\!\leftarrow s_i$.

The pairs (s_{i-1}, s_i) are called *proof steps*. A proof is called a *rewrite proof* in R_* if there is a $k \in \{0, \ldots, n\}$ so that $s_{i-1} \rightarrow_{R_*} s_i$ for $1 \leq i \leq k$ and $s_{i-1}\ {}_{R_*}\!\leftarrow s_i$ for $k + 1 \leq i \leq n$.

Idea (Bachmair, Derschowitz, Hsiang): Define a well-founded ordering on proofs so 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_*. A *cost* $c(s_{i-1}, s_i)$ is associated with every proof step as follows:

1. If $s_{i-1} \leftrightarrow_{E_\infty} s_i$ then $c(s_{i-1}, s_i) = (\{s_{i-1}, s_i\}, -, -)$ where the first component is a multiset of terms and $-$ denotes an arbitrary (irrelevant) term.

2. If $s_{i-1} \to_{R_\infty} s_i$ using $l \to r$ then $c(s_{i-1}, s_i) = (\{s_{i-1}\}, l, s_i)$.
3. If $s_{i-1} \ {}_{R_\infty}\!\leftarrow s_i$ using $l \to r$ then $c(s_{i-1}, s_i) = (\{s_i\}, l, s_{i-1})$.

Proof steps are compared using the lexicographical 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 cost of its proof steps. The *proof ordering* $\succ_C$ compares the cost of proofs using the multiset extension of the proof step ordering.

Lemma 4.4.8. $\succ_C$ is well-founded ordering.

Lemma 4.4.9. Let P be a proof in $E_\infty \cup R_\infty$. If P is not a rewrite proof in R_* then there exists an equivalent proof P' in $E_\infty \cup R_\infty$ so that $P \succ_C P'$.

Proof. If P is not a rewrite proof in R_* then it contains

1. a proof step that is in E_∞ or
2. a proof step that is in $R_\infty \backslash R_*$ or
3. a subproof $s_{i-1} \ {}_{R_*}\!\leftarrow s_i \to s_{i+1}$ (peak).

It is shown that in all three cases the proof step or subproof can be replaced by a smaller subproof:
Case 1.: A proof step using an equation $s \dot{\approx} t$ is in E_∞. This equation must be deleted during the run.

If $s \dot{\approx} t$ is deleted using *Orient*:

$$\ldots s_{i-1} \leftrightarrow_{E_\infty} s_i \ldots \quad \Longrightarrow \quad \ldots s_{i-1} \to_{R_\infty} s_i \ldots$$

If $s \dot{\approx} t$ is deleted using *Delete*:

$$\ldots s_{i-1} \leftrightarrow_{E_\infty} s_{i-1} \ldots \quad \Longrightarrow \quad \ldots s_{i-1} \ldots$$

If $s \dot{\approx} t$ is deleted using *Simplify-Eq*:

$$\ldots s_{i-1} \leftrightarrow_{E_\infty} s_i \ldots \quad \Longrightarrow \quad \ldots s_{i-1} \to_{R_\infty} s' \leftrightarrow_{E_\infty} s_i \ldots$$

Case 2.: A proof step using a rule $s \to t$ is in $R_\infty \backslash R_*$. This rule must be deleted during the run.

If $s \to t$ is deleted using *R-Simplify-Rule*:

$$\ldots s_{i-1} \to_{R_\infty} s_i \ldots \quad \Longrightarrow \quad \ldots s_{i-1} \to_{R_\infty} s' \ {}_{R_\infty}\!\leftarrow s_i \ldots$$

If $s \to t$ is deleted using *L-Simplify-Rule*:

$$\ldots s_{i-1} \to_{R_\infty} s_i \ldots \quad \Longrightarrow \quad \ldots s_{i-1} \to_{R_\infty} s' \leftrightarrow_{E_\infty} s_i \ldots$$

Case 3.: A subproof has the form $s_{i-1} \ {}_{R_*}\!\leftarrow s_i \to_{R_*} s_{i+1}$.

If there is no overlap or a non-critical overlap:

$$\ldots s_{i-1} \; {}_{R_*}\!\!\leftarrow s_i \rightarrow_{R_*} s_{i+1} \ldots \quad \Longrightarrow \quad \ldots s_{i-1} \rightarrow^*_{R_*} s' \; {}^*_{R_*}\!\!\leftarrow s_{i+1} \ldots$$

If there is a critical pair that has been added using *Deduce*:

$$\ldots s_{i-1} \; {}_{R_*}\!\!\leftarrow s_i \rightarrow_{R_*} s_{i+1} \ldots \quad \Longrightarrow \quad \ldots s_{i-1} \leftrightarrow_{E_\infty} s_{i+1} \ldots$$

In all cases, checking that the replacement subproof is smaller than the replaced subproof is routine. □

Theorem 4.4.10 (KBC Soundness)**.** Let $(E_0; R_0) \Rightarrow_{KBC} (E_1; R_1) \Rightarrow_{KBC} (E_2; R_2) \Rightarrow_{KBC} \ldots$ be a fair run and let R_0 and E_* be empty. Then

1. every proof in $E_\infty \cup R_\infty$ is equivalent to a rewrite proof in R_*,
2. R_* is equivalent to E_0 and
3. R_* is convergent.

Proof. 1. By well-founded induction on $\succ_C$ using the previous lemma.

2. Clearly, $\approx_{E_\infty \cup R_\infty} = \approx_{E_0}$. Since $R_* \subseteq R_\infty$ this yields $\approx_{R_*} \subseteq \approx_{E_\infty \cup R_\infty}$. On the other hand, by 1. it holds that $\approx_{E_\infty \cup R_\infty} \subseteq \approx_{R_*}$.

3. Since $\rightarrow_{R_*} \subseteq \succ$, R_* is terminating. By 1. it holds that R_* is confluent.

□

Now using the proof of Theorem 3.15.2 termination of $\Rightarrow_{KBC}$ is undecidable.

Corollary 4.4.11 (KBC Termination)**.** Termination of $\Rightarrow_{KBC}$ is undecidable for some given finite set of equations E.

Proof. Using exactly the construction of Theorem 3.15.2 it remains to be shown that all computed critical pairs can be oriented. Critical pairs corresponding to the search for a PCP solution result in equations $f_R(u(x), v(y)) \approx f_R(u'(x), v'(y))$ or $f_R(u'(x), v'(x)) \approx c$. By chosing an appropriate ordering, all these equations can be oriented. Thus $\Rightarrow_{KBC}$ does not produce any unorientable equations. The rest follows from Theorem 3.15.2. □

4.4.1 Unfailing Completion

Classical completion: Try to transform a set E of equations into an equivalent convergent TRS. Fail, if an equation can neither be oriented nor deleted.

*Unfailing completion (from Bachmair, Derschowitz and Plaisted [**?**]):* If an equation cannot be oriented, *orientable instances* can still be used for rewriting. Note: If $\succ$ is total on ground terms, then every *ground instance* of an equation is trivial or can be oriented. The goal is to derive a *ground convergent* set of equations.