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 Tableaux (Section 3.6) and Superposition (Section **??**) are presented only for its restriction without equality. Purely equational logic and first-order logic with equality are presented separately in Chapter **??** and Chapter **??**, respectively.

3.1 Syntax

Definition 3.1.1 (Many-Sorted Signature)**.** A *many-sorted signature* $\Sigma = (\mathcal{S}, \Omega, \Pi)$ is a pair 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 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 $\mathrm{sort}(t)$, i.e., if $t \in T_S(\Sigma, \mathcal{X})$ then $\mathrm{sort}(t) = S$. 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.

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 $P \in \Pi$, $\mathrm{arity}(P) = n$, $P \subseteq S_1 \times \ldots \times S_n$. An atom or its negation of an atom is called a *literal*.

The literal $s \stackrel{\cdot}{\approx} t$ 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 B to $\mathcal{S}$, (ii) introduce a fresh constant true of the sort B to Ω, (iii) for every predicate P, $P \subseteq S_1 \times \ldots \times S_n$ add a fresh function $f_P : S_1, \ldots, S_n \to B$ to Ω, and (iv) encode every atom $P(t_1, \ldots, t_n)$ as a function $f_P : S_1, \ldots, S_n \to B$. Thus, predicate atoms are turned into equations $f_P(t_1, \ldots, t_n) \approx \mathrm{true}$. are overloaded here.

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
$\bot$	falsum
$\top$	verum
$P(t_1, \ldots, t_n), s \approx t$	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 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 $\circ \in \{\wedge, \vee, \rightarrow, \leftrightarrow\}$ 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: $\mathrm{depth}(t) := max\{|p| \mid p \in \mathrm{pos}(t)\}$ ($\mathrm{depth}(\phi) := max\{|p| \mid p \in \mathrm{pos}(\phi)\}$). The set of *all* variables occurring in a term t (formula ϕ) is denoted by $\mathrm{vars}(t)$ ($\mathrm{vars}(phi)$) and formally defined as $\mathrm{vars}(t) := \{x \in \mathcal{X} \mid x = t|_p, p \in \mathrm{pos}(t)\}$ ($\mathrm{vars}(\phi) := \{x \in \mathcal{X} \mid x = \phi|_p, p \in \mathrm{pos}(\phi)\}$). A term t (formula ϕ) is *ground* if $vars(t) = \emptyset$ ($vars(\phi) = \emptyset$).

Note that $\mathrm{vars}(\forall x.a \approx b) = \emptyset$ where a, b are constants. This is justified by the fact that the formula does not depend on the quantifier, see semantics below. The set of *free* variables of a formula ϕ (term t) is given by $\mathrm{fvars}(\phi, \emptyset)$ ($\mathrm{fvars}(t, \emptyset)$) and recursively defined by $\mathrm{fvars}(\psi_1 \circ \psi_2, B) := \mathrm{fvars}(\psi_1, B) \cup \mathrm{fvars}(\psi_2, B)$ where $\circ \in \{\wedge, \vee, \rightarrow, \leftrightarrow\}$, $\mathrm{fvars}(\forall x.\psi, B) := \mathrm{fvars}(\psi, B \cup \{x\})$, $\mathrm{fvars}(\exists x.\psi, B) := \mathrm{fvars}(\psi, B \cup \{x\})$, $\mathrm{fvars}(\neg\psi, B) := \mathrm{fvars}(\psi, B)$, $\mathrm{fvars}(L, B) := \mathrm{vars}(\psi) \setminus B$ ($\mathrm{fvars}(t, B) := \mathrm{vars}(t) \setminus B$. For $\mathrm{fvars}(\phi, \emptyset)$ I also write $\mathrm{fvars}(\phi)$

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 $\forall x_1.\forall x_2 \ldots \forall x_n.\phi$, $\exists x_1.\forall x_2 \ldots \forall x_n.\phi$, respectively) are the *universal* and the *existential closure* of ϕ.

Example 3.1.6. For the literal $\neg P(f(x, g(a)))$ the atom $P(f(x, g(a)))$ is an immediate subformula of 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 $vars(\neg P(f(x, g(a)))) = \{x\}$.

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

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

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 $\phi \in \mathrm{FOL}(\Sigma, \mathcal{X})$ by

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

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 $\{\phi\} \models M$.

Since 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.

3.3 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 two fresh constants true and false.

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 3.3.1. 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 3.3.2. 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.

3.4 Substitution and Unifier

Definition 3.4.1 (Substitution)**.** A *substitution* is a mapping $\sigma : \mathcal{X} \rightarrow T(\Sigma, \mathcal{X})$ so that

1. $\sigma(x) \neq x$ for only finitely many variables x and
2. $sort(x) = sort(t)$ for every variable $x \in \mathcal{X}$ that is mapped to a term $t \in T_S(\Sigma, \mathcal{X})$.

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 of substitution 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 $\sigma\sigma = \sigma$. σ is idempotent iff $\mathrm{dom}(\sigma) \cap \mathrm{vars}(\mathrm{codom}(\sigma)) = \emptyset$.

Substitutions are often written as $\{x_1 \mapsto t_1, \ldots, x_n \mapsto t_n\}$ 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 expression and defined as following:

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 $e\sigma$ of applying a substitution σ to an expression e is called an *instance* of e. The substitution σ is called *ground* if it maps every domain variable to a ground term. If the application of a substitution σ to an expression e produces a ground expression $e\sigma$ then $e\sigma$ is called *ground instance* of e. A ground substitution σ is called *grounding for an expression* e if $e\sigma$ is ground. A substitution σ is called *variable renaming* if $im(\sigma) \subseteq \mathcal{X}$ and for any $x, y \in \mathcal{X}$, if $x \neq y$ then $x\sigma \neq y\sigma$.

Definition 3.4.2 (Unifier)**.** Two terms s and t are said to be *unifiable* if there exists a substitution σ so that $s\sigma = t\sigma$, the substitution σ is then called a *unifier* of s and t. The unifier σ is called *most general unifier*, written $\sigma = mgu(s, t)$, if any other unifier τ of s and t can be represented as $\tau = \sigma\tau'$, for some substitution τ'.

3.5 Unification Calculi

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 with $\text{sort}(s) = \text{sort}(t)$ (or atoms A, B) are unifiable is the set $E = \{s = t\}$. A variable x is solved in E if $E = \{x = t\} \uplus E'$, $x \notin \text{vars}(t)$ and $x \notin \text{vars}(E)$.

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

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

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

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

Occurs Check $E \uplus \{x = t\} \Rightarrow_{\mathrm{SU}} \bot$
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 3.5.1 (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 3.5.2 (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 $x \in vars(t)$

Orientation $E \uplus \{t = x\} \Rightarrow_{\mathrm{PU}} E \uplus \{x = t\}$
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 3.5.3 (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 3.5.4 (Unifier generated by $\Rightarrow_{\mathrm{PU}}$)**.** Let $\{s = t\} \Rightarrow^*_{\mathrm{PU}} \{x_1 = t_1, \ldots, x_n = t_n\}$. Then

γ	Descendant $\gamma(t)$
$\forall x_S.\psi$	$\psi\{x_S \mapsto t\}$
$\neg\exists x_S.\psi$	$\neg\psi\{x_S \mapsto t\}$
	for any ground term $t \in T_S(\Sigma)$

δ	Descendant $\delta(c)$
$\exists x_S.\psi$	$\psi\{x_S \mapsto c\}$
$\neg\forall x_S.\psi$	$\neg\psi\{x_S \mapsto c\}$
	for some fresh Skolem constant $c \in T_S(\Sigma)$

Figure 3.1: γ- and δ-Formulas

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. □

3.6 First-Order Tableaux

The different versions of tableaux for first-order logic differ in particular in the treatment of variables by the tableaux rules. The first variant is standard first-order tableaux where variables are instantiated by ground terms.

Definition 3.6.1 (γ-,δ-Formulas)**.** A formula ϕ is called a γ-formula if ϕ is a formula $\forall x_S.\psi$ or $\neg\exists x_S.\psi$. A formula ϕ is called a δ-formula if ϕ is a formula $\exists x_S.\psi$ or $\neg\forall x_S.\psi$.

Definition 3.6.2 (Direct Standard Tableaux Descendant)**.** Given a γ- or δ-formula ϕ, Figure 3.1 shows its direct descendants.

For the standard first-order tableaux rules to make sense "enough" Skolem constants are needed in the signature, e.g., countably infinitely many for each sort. A δ formula ϕ occurring in some sequence is called *open* if no direct descendant of it is part of the sequence. In general, the number of γ descendants cannot be limited for a successful tableaux proof.

γ-Expansion $\quad N \uplus \{(\phi_1, \ldots, \psi, \ldots, \phi_n)\} \Rightarrow_{\mathrm{FT}} N \uplus \{(\phi_1, \ldots, \psi, \ldots, \phi_n, \psi')\}$
provided ψ is a γ-formula, ψ' a $\gamma(t)$ descendant where t is an arbitrary ground term in the signature of the sequence (branch) and the sequence is not closed.

δ-Expansion $\quad N \uplus \{(\phi_1, \ldots, \psi, \ldots, \phi_n)\} \Rightarrow_{\mathrm{FT}} N \uplus \{(\phi_1, \ldots, \psi, \ldots, \phi_n, \psi')\}$

provided ψ is an open δ-formula, ψ' a $\delta(c)$ descendant where c is fresh to the sequence and the sequence is not closed.

The standard first-order tableaux calculus consists of the rules α-, and β-expansion (see Section 2.4) and the above two rules γ-Expansion and δ-Expansion.

Theorem 3.6.3 (Standard First-Order Tableaux is Sound and Complete)**.** A formula ϕ (without equality) is valid iff standard tableaux computes a closed state out of $\{(\neg\phi)\}$.

Skolem *constants* are sufficient: In a δ-formula $\exists x\,\phi$, $\exists$ is the outermost quantifier and x is the only free variable in ϕ. The γ rule has to be applied several times to the same formula for tableaux to be complete. For instance, constructing a closed tableau for

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

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

The main disadvantage of standard first-order tableau is that the γ ground term instances need to be guessed. The whole complexity of the problem lies in this guessing as for otherwise tableaux terminates. A natural idea is to guess ground terms that can eventually be used to close a branch. This is the idea of free-variable first-order tableaux. Instead of guessing a ground term for a γ formula the variable remains, 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 Skolem terms in the formerly universally quantified variables that had the δ formula in their scope.

The new calculus suggests to keep track of scopes of variables, so I move from a state as a set of sequences of formulas to a set of sequences of pairs $l_i = (\phi_i, X_i)$ where X_i is a set of variables.

Definition 3.6.4 (Direct Free-Variable Tableaux Descendant)**.** Given a γ- or δ-formula ϕ, Figure 3.2 shows its direct descendants.

γ-Expansion $N \uplus \{(l_1, \ldots, (\psi, X), \ldots, l_n)\} \Rightarrow_{\mathrm{FT}} N \uplus \{(l_1, \ldots, (\psi, X), \ldots, l_n, (\psi', X \cup \{y\}))\}$
provided ψ is a γ-formula, ψ' a $\gamma(y)$ descendant where y is fresh to the sequence (branch) and the sequence is not closed.

δ-Expansion $N \uplus \{(l_1, \ldots, (\psi, X), \ldots, l_n)\} \Rightarrow_{\mathrm{FT}} N \uplus \{(l_1, \ldots, (\psi, X), \ldots, l_n, (\psi', X))\}$
provided ψ is an open δ-formula, ψ' a $\delta(f(y_1, \ldots, y_n))$ descendant where f is fresh to the sequence, $X = \{y_1, \ldots, y_n\}$ and the sequence is not closed.

Branch-Closing $N \uplus \{(l_1, \ldots, (L, X), \ldots, (K, X'), \ldots, l_n)\} \Rightarrow_{\mathrm{FT}} N\sigma \uplus \{(\phi_1, \ldots, (L, X), \ldots, (K, X'), \ldots, \phi_n, \}\sigma$

γ	Descendant $\gamma(y)$
$\forall x_S.\psi$	$\psi\{x_S \mapsto y\}$
$\neg\exists x_S.\psi$	$\neg\psi\{x_S \mapsto y\}$
	for a fresh variable $y, \mathrm{sort}(y) = S$

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
	where $f(y_1, \ldots, y_n) \in T_S(\Sigma)$

Figure 3.2: γ- and δ-Formulas

provided K and L are literals and there is an mgu σ such that $K\sigma = \neg L\sigma$ and the sequence is not closed.

The standard first-order tableaux calculus consists of the rules α-, and β-expansion (see Section 2.4) which are adapted to pairs and the above three rules γ-Expansion, δ-Expansion and Branch-Closing.

Theorem 3.6.5 (Free-variable First-Order Tableaux is Sound and Complete)**.** A formula ϕ (without equality) is valid iff free-variable tableaux computes a closed state out of $\{(\neg\phi)\}$.

Example 3.6.6.

1.	$\neg[\exists w \forall x R(x, w, f(x, w)) \to \exists w \forall x \exists y R(x, w, y)]$	
2.	$\exists w \forall x\ R(x, w, f(x, w))$	1_1 $[\alpha]$
3.	$\neg\exists w \forall x \exists y\ R(x, w, y)$	1_2 $[\alpha]$
4.	$\forall x\ R(x, c, f(x, c))$	$2(c)$ $[\delta]$
5.	$\neg\forall x \exists y\ R(x, v_1, y)$	$3(v_1)$ $[\gamma]$
6.	$\neg\exists y\ R(g(v_1), v_1, y)$	$5(g(v_1))$ $[\delta]$
7.	$R(v_2, c, f(v_2, c))$	$4(v_2)$ $[\gamma]$
8.	$\neg R(g(v_1), v_1, v_3)$	$6(v_3)$ $[\gamma]$

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

$$v_2 = g(v_1),\ c = v_1,\ f(v_2, c) = v_3$$

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

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

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

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

Semantic 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.

3.7 First-Order CNF Transformation

Similar to the propositional case, first-order superposition operates 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 renaming this replacement only preserves satisfiability. Eventually, all variables in clauses are implicitly universally quantified.

As usual, the CNF transformation is done by a set of rules. All rules known from the propositional case apply. Further rules deal with the quantifies $\forall$, $\exists$ and some of the propositional rules need an extension in order to cope with first-order variables.

The first set of rules eliminates $\top$ and $\bot$ from a first-order formula.

ElimTB1 $\chi[(\phi \wedge \top)]_p \Rightarrow_{\mathrm{CNF}} \chi[\phi]_p$

ElimTB2 $\chi[(\phi \wedge \bot)]_p \Rightarrow_{\mathrm{CNF}} \chi[\bot]_p$

ElimTB3 $\chi[(\phi \vee \top)]_p \Rightarrow_{\mathrm{CNF}} \chi[\top]_p$

ElimTB4 $\chi[(\phi \vee \bot)]_p \Rightarrow_{\mathrm{CNF}} \chi[\phi]_p$

ElimTB5 $\chi[\neg\bot]_p \Rightarrow_{\mathrm{CNF}} \chi[\top]_p$

ElimTB6 $\chi[\neg\top]_p \Rightarrow_{\mathrm{CNF}} \chi[\bot]_p$

ElimTB7 $\chi[\phi \leftrightarrow \bot]_p \Rightarrow_{\text{CNF}} \chi[\neg\phi]_p$

ElimTB8 $\chi[\phi \leftrightarrow \top]_p \Rightarrow_{\text{CNF}} \chi[\phi]_p$

ElimTB9 $\chi[\phi \rightarrow \bot]_p \Rightarrow_{\text{CNF}} \chi[\neg\phi]_p$

ElimTB10 $\chi[\phi \rightarrow \top]_p \Rightarrow_{\text{CNF}} \chi[\top]_p$

ElimTB11 $\chi[\bot \rightarrow \phi]_p \Rightarrow_{\text{CNF}} \chi[\top]_p$

ElimTB12 $\chi[\top \rightarrow \phi]_p \Rightarrow_{\text{CNF}} \chi[\phi]_p$

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

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

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

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

Once the variable renaming is done, renaming of beneficial subformulas is the next step. The 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 $\phi \Rightarrow_{\text{CNF}} \chi[A_1]_{p_1}[A_2]_{p_2}\ldots[A_n]_{p_n} \wedge \text{def}(\phi,p_1,A_1) \wedge \ldots \wedge \text{def}(\chi[A_1]_{p_1}[A_2]_{p_2}\ldots[A_{n-1}]_{p_{n-1}},p_n,A_n)$
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 the $A_i = P_i(x_{i,1},\ldots,x_{i,k_i})$ where $\text{fvars}(\phi|_{p_i}) = \{x_{i,1},\ldots,x_{i,k_i}\}$ and all P_i are different and new to ϕ

Negation normal form is again done as in the propositional case with additional rules for the quantifiers.

ElimEquiv1 $\chi[(\phi \leftrightarrow \psi)]_p \Rightarrow_{\text{CNF}} \chi[(\phi \rightarrow \psi) \wedge (\psi \rightarrow \phi)]_p$
provided $\text{pol}(\chi, p) \in \{0, 1\}$

ElimEquiv2 $\chi[(\phi \leftrightarrow \psi)]_p \Rightarrow_{\text{CNF}} \chi[(\phi \wedge \psi) \vee (\neg\phi \wedge \neg\psi)]_p$
provided $\text{pol}(\chi, p) = -1$

ElimImp $\chi[(\phi \rightarrow \psi)]_p \Rightarrow_{\text{CNF}} \chi[(\neg\phi \vee \psi)]_p$

PushNeg1 $\chi[\neg(\phi \vee \psi)]_p \Rightarrow_{\text{CNF}} \chi[(\neg\phi \wedge \neg\psi)]_p$

PushNeg2 $\chi[\neg(\phi \wedge \psi)]_p \Rightarrow_{\text{CNF}} \chi[(\neg\phi \vee \neg\psi)]_p$

PushNeg3 $\chi[\neg\neg\phi]_p \Rightarrow_{\text{CNF}} \chi[\phi]_p$

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

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

In propositional logic after NNF, 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 $\chi[\forall x.(\psi_1 \circ \psi_2)]_p \Rightarrow_{\text{CNF}} \chi[(\forall x.\psi_1) \circ \psi_2]_p$
provided $\circ \in \{\wedge, \vee\}$, $x \notin \text{fvars}(\psi_2)$

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

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

MiniScope4 $\chi[\exists x.(\psi_1 \vee \psi_2)]_p \Rightarrow_{\text{CNF}} \chi[(\exists x.\psi_1) \vee (\exists x.\psi_2)\sigma]_p$
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 $\chi[\exists x, \psi]_p \Rightarrow_{\text{CNF}} \chi[\psi\{x \mapsto f(y_1, \ldots, y_n)\}]_p$
provided there is no q, $q < p$ with $\phi|_q = \exists x'.\psi'$, $\text{fvars}(\exists x.\psi) = \{y_1, \ldots, y_n\}$, $\text{arity}(f) = n$ is a new function symbol to ϕ matching the respective sorts of the y_i with range sort $\text{sort}(x)$

Example 3.7.1 (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

$$\forall x.\exists y.(R(x,x) \wedge P(y)) \Rightarrow_{\text{CNF,Skolemization}} \forall x.(R(x,x) \wedge P(g(x))$$

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

$$\begin{array}{ll} \forall x.\exists y.(R(x,x) \wedge P(y)) & \Rightarrow^*_{\text{CNF,MiniScope2,1}} \quad \forall x.R(x,x) \wedge \exists y.P(y) \\ & \Rightarrow_{\text{CNF,Skolemization}} \quad \forall x.R(x,x) \wedge P(a) \end{array}$$

for some Skolem constant a 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 3.14).

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 $\chi[\forall x.\psi]_p \Rightarrow_{\text{CNF}} \chi[\psi]_p$

The actual CNF is then done by distributivity.

PushDisj $\chi[(\phi_1 \wedge \phi_2) \vee \psi]_p \Rightarrow_{\text{CNF}} \chi[(\phi_1 \vee \psi) \wedge (\phi_2 \vee \psi)]_p$

Theorem 3.7.2 (Properties of the CNF Transformation)**.** Let ϕ be a first-order sentence, then

1. $\text{cnf}(\phi)$ terminates
2. ϕ is satisfiable iff $\text{cnf}(\phi)$ 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 6.
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. □

Algorithm 6: cnf(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;
```

C 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.

3.8 Herbrand Interpretations

For propositional logic the existence of a canonical model is straightforward because the definition of the semantics leads to an effective representation. A propositional variable can be either true or false. For first-order logic this is no longer straightforward because an interpretation can assign any non-empty set to a sort, any function to a function symbol and any relation to a predicate symbol. A giant step forward towards the mechanization of first-order logic was the discovery of a canonical model construction by Herbrand. A first-order formula has a model iff it has such a canonical model which is build out of the syntax.

For this and the following section I restrict the focus to first-order logic without equality. Equality is then considered and added to the concepts of this chapter in Chapters ??, ??.

Definition 3.8.1 (Herbrand Interpretation)**.** A *Herbrand Interpretation* (over Σ) is a Σ-algebra $\mathcal{A}$ so that

1. $S^{\mathcal{A}} = T_S(\Sigma)$ for every sort $S \in \mathcal{S}$

2. $f^{\mathcal{A}} : (s_1, \ldots, s_n) \mapsto f(s_1, \ldots, s_n)$ 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
3. $P^{\mathcal{A}} \subseteq (T_{S_1}(\Sigma) \times \ldots \times T_{S_m}(\Sigma))$ where $P \in \Pi$, $\text{arity}(P) = m$ and $P \subseteq S_1 \times \ldots \times S_m$ is the sort declaration for P

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

Proposition 3.8.2. Every set of ground atoms I uniquely determines a Herbrand interpretation $\mathcal{A}$ via

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

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, if $I \models \phi$.

Example 3.8.3. 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 3.8.4 (Herbrand)**.** Let N be a set of Σ-clauses. Then N is satisfiable iff N has a Herbrand model over Σ iff $\text{ground}(\Sigma, N)$ has a Herbrand model over Σ, where $\text{ground}(\Sigma, N) = \{C\sigma \mid C \in N, \text{dom}(\sigma) = \text{vars}(C)$, and $x\sigma \in T_{\text{sort}(x)}(\Sigma)$ for all $x \in \text{dom}(\sigma)\}$ is the set of *ground instances* of N.

Example 3.8.5 (Example of a $\text{ground}(\Sigma, N)$)**.** Consider Σ' from Example 3.8.3 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 $\text{ground}(\Sigma', N) = \{$

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

is satisfiable. For example by the Herbrand models
$I_1 := \emptyset$
$I_2 := \{P(b), Q(b,b), P(g(b)), Q(g(b), g(b)), \ldots\}$

3.9 Orderings

Definition 3.9.1 (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 3.9.2. 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 3.9.3 (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.

Definition 3.9.4 (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 3.9.5.
1. The LPO is a rewrite ordering.
2. If the precedence $\succ$ is total on Ω then $\succ_{lpo}$ is total on the set of ground terms $T(\Sigma)$.
3. If Ω is finite then $\succ_{lpo}$ is well-founded.

Example 3.9.6. 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 3.9.7 (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 $w : \Omega \cup \mathcal{X} \to \mathbb{R}_0^+$ be a *weight function*, so that the following admissibility conditions are satisfied:

1. $w(x) = w_0 \in \mathbb{R}^+$ for all variables $x \in \mathcal{X}$; $w(c) \geq w_0$ for all constants $c \in \Omega$.
2. If $w(f) = 0$ for some $f \in \Omega$ with arity$(f) = 1$, then $f \succeq g$ for all $g \in \Omega$.

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) $t = x$, $s = f^n(x)$ for some $n \geq 1$, or
   (b) $s = f(s_1, \ldots, s_m)$, $t = g(t_1, \ldots, t_n)$, and $f \succ g$, or
   (c) $s = f(s_1, \ldots, s_m)$, $t = f(t_1, \ldots, t_m)$, and $(s_1, \ldots, s_m)(\succ_{kbo})_{lex}(t_1, \ldots, t_m)$.

Theorem 3.9.8. 1. The KBO is a rewrite ordering.

2. If the precedence $\succ$ is total on Ω then $\succ_{kbo}$ is total on the set of ground terms $T(\Sigma)$.
3. If Ω is finite then $\succ_{kbo}$ is well-founded.

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 3.9.4-2. 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.

3.10 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 2.6 for first-order ground clauses. Let N be a set of ground clauses.

Definition 3.10.1 (Clause Ordering)**.** Let $\prec$ be a total strict rewrite ordering on terms and 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 3.10.2 (Properties of the Clause Ordering)**.** (i) The orderings on literals and clauses are total and well-founded.
(ii) Let C and D be clauses with $P(t_1, \ldots, t_n) = |\max(C)|$, $Q(s_1, \ldots, s_m) = |\max(D)|$, where $\max(C)$ denotes the maximal literal in C.

1. If $Q(s_1, \ldots, s_m) \prec_L P(t_1, \ldots, t_n)$ then $D \prec_C C$.
2. 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 3.10.3 (Abstract Redundancy)**.** A ground clause C is *redundant* with respect to a ground clause set 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 3.10.4 (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 on a clause, any superposition inference must be on the selected literal.

Definition 3.10.5 (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:

$$
\begin{array}{lcl}
N_C & := & \bigcup_{D \prec C} \delta_D \\
\\
\delta_D & := & \begin{cases} \{P(t_1,\ldots,t_n)\} & \text{if } D = D' \vee P(t_1,\ldots,t_n), P(t_1,\ldots,t_n) \text{ strictly maximal, no literal} \\ & & \text{selected in } D \text{ and } N_D \not\models D \\ \emptyset & \text{otherwise} \end{cases} \\
N_\mathcal{I} & := & \bigcup_{C \in N} \delta_C
\end{array}
$$

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

Proposition 3.10.6. 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 $\delta_C = \{P\}$.

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 3.8.2). $N_\mathcal{I}$ 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 3.10.7 (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.

Examples for specific redundancy rules that can be efficiently decided are

Subsumption provided $C_1 \subset C_2$	$(N \uplus \{C_1, C_2\}) \Rightarrow_{\text{SUP}} (N \cup \{C_1\})$
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 3.10.8. All clauses removed by Subsumption, Tautology Deletion, Condensation and Subsumption Resolution are redundant with respect to the kept or added clauses.

Theorem 3.10.9. 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 3.10.10 (Compactness of First-Order Logic)**.** Let N be a, possibly 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 $N \Rightarrow^i_{\text{SUP}} N'$ 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 3.10.11 (Compactness of First-Order Logic: Classical)**.** A set N of clauses is satisfiable iff all finite subsets of N are satisfiable

Theorem 3.10.12 (Soundness and Completeness of Ground Superposition)**.** A first-order Σ-sentence ϕ is valid iff there exists a ground superposition refutation for $\text{ground}(\Sigma, \text{cnf}(\neg\phi))$.

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

Theorem 3.10.13 (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 $\text{ground}(\Sigma, \text{cnf}(\neg\phi))$ and perform superposition inference steps. The enumeration can, e.g., be done by considering Herbrand terms of increasing size. □

Example 3.10.14 (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 3.10.15 (Craig Theorem [14])**.** Let ϕ and ψ be two propositional 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 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. □

3.11 First-Order Superposition with Selection

The completeness proof of ground superposition (Section 3.10) 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 3.10.1 where clauses with variables are projected to their ground instances for ordering computations.

Definition 3.11.1 (Maximal Literal)**.** A literal L is called *[strictly] maximal* in a clause C if and only if there exists a grounding substitution σ so that $L\sigma$ is [strictly] maximal in $C\sigma$ (i.e., if for no other L' in C: $L\sigma \prec L'\sigma$ [$L\sigma \preceq L'\sigma$]).

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

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

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

where (i) $P(t_1,\ldots,t_n)\sigma$ is maximal in $(C \vee P(t_1,\ldots,t_n) \vee P(t_1,\ldots,t_n))\sigma$ (ii) no literal is selected in $C \vee P(t_1,\ldots,t_n) \vee P(t_1,\ldots,t_n)$ (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 Superpositions Left and Factoring are generalizations of their respective counterparts from Section 3.10. On ground clauses they coincide. Therefore, we can safely overload them in the sequel.

Definition 3.11.2 (Abstract Redundancy)**.** A clause C is *redundant* with respect to a clause set N if for all ground instances $C\sigma$ where 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 3.11.3 (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.

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 redundancy notions from Section 3.10 carry over to the non-ground case. Let dup be a function from clauses to clauses that removes duplicate literals, i.e., $\mathrm{dup}(C) = C'$ 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 $(N \uplus \{C_1 \vee L \vee L'\}) \Rightarrow_{\mathrm{SUP}} (N \cup \{\mathrm{dup}((C_1 \vee L \vee L')\sigma)\})$
provided $L\sigma = L'$ and $\mathrm{dup}((C_1 \vee L \vee L')\sigma)$ 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 3.11.4. All reduction rules are instances of the abstract redundancy criterion

Lemma 3.11.5 (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 bound 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 3.11.6 (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

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

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

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

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

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

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

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

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 3.11.7 (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 3.11.8 (Soundness and Completeness)**.** Superposition is sound and complete.

Proof. Soundness is obvious. For completeness, Theorem 3.10.12 proves the ground case. Now by applying Lemma 3.11.6 to this proof it can be lifted to the first-order level. □

There are questions left open by Lemma 3.11.8. It just says that a ground refutation can be lifted to a first-order refutation. But what about abstract redundancy, Definition 3.11.2? 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 $\mathrm{ground}(\Sigma, N)_{\mathcal{I}}$ a model? The next two lemmas answer these questions positively.

Lemma 3.11.9 (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 $\mathrm{ground}(\Sigma, N)$ such that no ground clause in the refutation is redundant. Now according to Lemma 3.11.8 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 3.11.2 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 3.11.10 (Model Property)**.** If N is a saturated clause set and $\bot \notin N$ then $\text{ground}(\Sigma, N)_{\mathcal{I}} \models N$.

Proof. As usual we assume that selection on the ground and respective non-ground clauses is identical. Assume $\text{ground}(\Sigma, N)_{\mathcal{I}} \not\models N$. Then there is a minimal ground clause $C\sigma$, $C \neq \bot$, $C \in N$ such that $\text{ground}(\Sigma, N)_{\mathcal{I}} \not\models C\sigma$. Note that $C\sigma$ is not redundant as for otherwise $\text{ground}(\Sigma, N)_{\mathcal{I}} \models C\sigma$. So $\text{ground}(\Sigma, N)$ 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 $L\sigma \in \text{ground}(\Sigma, N)_{\mathcal{I}}$ and hence $\text{ground}(\Sigma, N)_{\mathcal{I}} \models C\sigma$ contradicting $\text{ground}(\Sigma, N)_{\mathcal{I}} \not\models C\sigma$.

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, $\text{ground}(\Sigma, N)_{\mathcal{I}} \not\models (C' \vee L)\tau$, 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 $(D' \vee L')\rho$, $L'\rho \in \text{ground}(\Sigma, N)_{\mathcal{I}}$ 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 $\text{ground}(\Sigma, N)_{\mathcal{I}} \not\models (C' \vee D')\tau\sigma\rho$ contradicting the minimal choice of $C\sigma$. □

Definition 3.11.11 (Persistent Clause)**.** Let $N_0 \Rightarrow_{\text{SUP}} N_1 \Rightarrow_{\text{SUP}} \ldots$ 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 3.11.12 (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 3.11.13 (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 3.11.8 there is a derivation of $\bot$ by superposition. Furthermore, no clause contributing to the derivation of $\bot$ is redundant (Lemma 3.11.9). 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. □

Lemma 3.11.14. Let $\text{red}(N)$ 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 $\text{red}(N) \subseteq \text{red}(M)$
2. if $M \subseteq \text{red}(N)$ then $\text{red}(N) \subseteq \text{red}(N \setminus M)$

It follows that redundancy is preserved when, during a theorem proving process, new clauses are added (or derived) or redundant clauses are deleted. Furthermore, $\text{red}(N)$ may include clauses that are not in N.

Algorithm 7: 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;
```