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 $\Omega$ of *operator symbols* (also called *function symbols*) over $\mathcal{S}$ and a set $\Pi$ 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 $\Sigma$, a variable set $\mathcal{X}$, disjoint from $\Omega$ 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 $t_i \in T_S(\Sigma, \mathcal{X})$ 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 $\Sigma$.

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 $\Sigma$. 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 $\Sigma$. Any equation is an *atom* (also called *atomic formula*) as well as every $P(t_1, \ldots, t_n)$ where $t_i \in T_S(\Sigma, \mathcal{X})$ 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 $s \approx t$ 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 $s \not\approx t$.

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 $\text{Bool}$ to $\mathcal{S}$, (ii) introduce a fresh constant true of the sort $\text{Bool}$ to $\Omega$, (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 $\Omega$, 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 \( FOL(\Sigma, X) \) of many-sorted first-order formulas with equality over the signature \( \Sigma \) is defined as follows for formulas \( \phi, \psi \in F_\Sigma(X) \) and a variable \( x \in X \):

\[
\begin{array}{ll}
\text{FOL}(\Sigma, X) & \text{Comment} \\
\bot & \text{false} \\
\top & \text{true} \\
P(t_1, \ldots, t_n), s \approx t & \text{atom} \\
(\neg \phi) & \text{negation} \\
(\phi \land \psi) & \text{conjunction} \\
(\phi \lor \psi) & \text{disjunction} \\
(\phi \rightarrow \psi) & \text{implication} \\
(\phi \leftrightarrow \psi) & \text{equivalence} \\
\forall x.\phi & \text{universal quantification} \\
\exists x.\phi & \text{existential quantification}
\end{array}
\]

A consequence of the above definition is that \( \text{PROP}(\Sigma) \subseteq \text{FOL}(\Sigma', X) \) if the propositional variables of \( \Sigma \) are contained in \( \Sigma' \) 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:

\[
\begin{align*}
\text{pos}(x) & := \{\epsilon\} \text{ if } x \in X \\
\text{pos}(\phi) & := \{\epsilon\} \text{ if } \phi \in \{\top, \bot\} \\
\text{pos}(\neg \phi) & := \{\epsilon\} \cup \{1p \mid p \in \text{pos}(\phi)\} \\
\text{pos}(\phi \land \psi) & := \{\epsilon\} \cup \{1p \mid p \in \text{pos}(\phi)\} \cup \{2p \mid p \in \text{pos}(\psi)\} \\
\text{pos}(s \approx t) & := \{\epsilon\} \cup \{1p \mid p \in \text{pos}(s)\} \cup \{2p \mid p \in \text{pos}(t)\} \\
\text{pos}(\forall x.\phi) & := \{\epsilon\} \cup \{1p \mid p \in \text{pos}(\phi)\} \\
\text{pos}(\exists x.\phi) & := \{\epsilon\} \cup \{1p \mid p \in \text{pos}(\phi)\}
\end{align*}
\]

where \( \circ \in \{\land, \lor, \rightarrow, \leftrightarrow\} \) and \( t_i \in T(\Sigma, X) \) 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 \( \phi \)) is said to contain another term \( s \) (formula \( \psi \)) if \( t|_p = s \) (\( \phi|_p = \psi \)). It is called a strict subexpression if \( p \neq \epsilon \). The term \( t \) (formula \( \phi \)) is called an immediate subexpression of \( s \) (formula \( \psi \)) if \( |p| = 1 \). For terms a subexpression is called a subterm and for formulas a subformula, respectively.

The size of a term \( t \) (formula \( \phi \)), written \( |t| \) (\(|\phi|\)), is the cardinality of pos\((t)\), i.e., \( |t| := |\text{pos}(t)| \) (\(|\phi| := |\text{pos}(\phi)|\)). The depth of a term, formula is the maximal
CHAPTER 3. FIRST-ORDER LOGIC

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 \( \phi \)) is denoted by \( \text{vars}(t) \) (\( \text{vars}(\phi) \)) and formally defined as
\( \text{vars}(t) := \{ x \in X \mid x = \phi|_p, p \in \text{pos}(t)\} \)
\( \text{vars}(\phi) := \{ x \in X \mid x = \phi|_p, p \in \text{pos}(\phi)\} \). A term \( t \) (formula \( \phi \)) is ground if \( \text{vars}(t) = \emptyset \) (\( \text{vars}(\phi) = \emptyset \)). Note that \( \text{vars}(\forall x.a \equiv b) = \emptyset \) 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 \( \phi \) (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 \( \circ \in \{\land, \lor, \to, \leftrightarrow\} \), \( \text{fvars}(\forall x.\psi, B) := \text{fvars}(\psi, B \cup \{x\}) \), \( \text{fvars}(\exists x.\psi, B) := \text{fvars}(\psi, B \cup \{x\}) \), \( \text{fvars}(\neg \psi, B) := \text{fvars}(\psi, B) \), \( \text{fvars}(L, B) := \text{vars}(L) \setminus B \) (\( \text{fvars}(t, B) := \text{vars}(t) \setminus B \).

For \( \text{fvars}(\phi, \emptyset) \) I also write \( \text{fvars}(\phi) \).

The function \( \text{top} \) 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 \( \phi \) is called the scope of the quantifier. An occurrence \( q \) of a variable \( x \) in a formula \( \phi (\phi|_q = x) \) 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 \( \phi \) then \( \forall x_1, \ldots, x_n.\phi \) and \( \exists x_1, \ldots, x_n.\phi \) (abbreviations for \( \forall x_1.\forall x_2.\cdots.\forall x_n.\phi, \exists x_1.\exists x_2.\cdots.\exists x_n.\phi \), respectively) are the universal and the existential closure of \( \phi \), 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 \( \neg P(f(x, g(a)))|_{b_{111}} = \neg P(f(h, g(a))) \) is obtained by replacing \( x \) with \( b \). \( \text{pos}(\neg P(f(x, g(a)))) = \{1, 1, 11, 111, 112, 1121\} \) meaning its size is 6, its depth 4 and \( \text{vars}(\neg P(f(x, g(a)))) = \{x\} \).

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

\[
\begin{align*}
\text{pol}(\phi, \epsilon) & := 1 \\
\text{pol}(\neg \phi, 1p) & := -\text{pol}(\phi, p) \\
\text{pol}(\phi \circ \phi_2, ip) & := \text{pol}(\phi_1, p) \text{ if } \circ \in \{\land, \lor\} \\
\text{pol}(\phi_1 \to \phi_2, 1p) & := -\text{pol}(\phi_1, p) \\
\text{pol}(\phi_1 \leftrightarrow \phi_2, ip) & := 0 \\
\text{pol}(P(t_1, \ldots, t_n), p) & := 1 \\
\text{pol}(t \approx s, p) & := 1 \\
\text{pol}(\forall x.\phi, 1p) & := \text{pol}(\phi, p) \\
\text{pol}(\exists x.\phi, 1p) & := \text{pol}(\phi, p)
\end{align*}
\]
3.2 Semantics

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

A (variable) assignment, also called a valuation for an algebra \( A \) is a function \( \beta : X \to U_A \) so that \( \beta(x) \in S_A \) for every variable \( x \in X \), where \( S = \text{sort}(x) \). A modification \( \beta[x \mapsto e] \) of an assignment \( \beta \) at a variable \( x \in X \), where \( e \in S_A \) 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 \( \beta \) for every variable except \( x \), which is mapped by \( \beta[x \mapsto e] \) to \( e \).

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

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

An algebra \( A \) is called term-generated, if every element \( e \) of the universe \( U_A \) of \( A \) is the image of some ground term \( t \), i.e., \( A(t) = e \).

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

\[
\begin{align*}
A(\beta)(\bot) & := 0 \\
A(\beta)(\top) & := 1 \\
A(\beta)(s \approx t) & := 1 \text{ if } A(\beta)(s) = A(\beta)(t) \text{ and } 0 \text{ otherwise} \\
A(\beta)(P(t_1, \ldots, t_n)) & := 1 \text{ if } (A(\beta)(t_1), \ldots, A(\beta)(t_n)) \in P^A \text{ and } 0 \text{ otherwise} \\
A(\beta)(\neg \phi) & := 1 - A(\beta)(\phi) \\
A(\beta)(\phi \land \psi) & := \min\{A(\beta)(\phi), A(\beta)(\psi)\} \\
A(\beta)(\phi \lor \psi) & := \max\{A(\beta)(\phi), A(\beta)(\psi)\} \\
A(\beta)(\phi \rightarrow \psi) & := \max\{1 - A(\beta)(\phi), A(\beta)(\psi)\} \\
A(\beta)(\phi \leftrightarrow \psi) & := \text{if } A(\beta)(\phi) = A(\beta)(\psi) \text{ then } 1 \text{ else } 0 \\
A(\beta)(\exists x_S \phi) & := 1 \text{ if } A(\beta[x \mapsto e])(\phi) = 1 \text{ for some } e \in S_A \text{ and } 0 \text{ otherwise} \\
A(\beta)(\forall x_S \phi) & := 1 \text{ if } A(\beta[x \mapsto e])(\phi) = 1 \text{ for all } e \in S_A \text{ and } 0 \text{ otherwise}
\end{align*}
\]
A formula $\phi$ is called satisfiable by $\mathcal{A}$ under $\beta$ (or valid in $\mathcal{A}$ under $\beta$) if $\mathcal{A}, \beta \models \phi$; in this case, $\phi$ is also called consistent; satisfiable by $\mathcal{A}$ if $\mathcal{A}, \beta \models \phi$ for some assignment $\beta$; satisfiable if $\mathcal{A}, \beta \models \phi$ for some algebra $\mathcal{A}$ and some assignment $\beta$; valid in $\mathcal{A}$, written $\mathcal{A} \models \phi$, if $\mathcal{A}, \beta \models \phi$ for any assignment $\beta$; in this case, $\mathcal{A}$ is called a model of $\phi$; valid, written $\models \phi$, if $\mathcal{A}, \beta \models \phi$ for any algebra $\mathcal{A}$ and any assignment $\beta$; in this case, $\phi$ is also called a tautology; unsatisfiable if $\mathcal{A}, \beta \not\models \phi$ for any algebra $\mathcal{A}$ and any assignment $\beta$; in this case $\phi$ is also called inconsistent.

Note that $\bot$ is inconsistent whereas $\top$ is valid. If $\phi$ 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 $\phi$ and $\psi$, $\phi \text{ entails } \psi$, or $\psi$ is a consequence of $\phi$, written $\phi \models \psi$, if for any algebra $\mathcal{A}$ and assignment $\beta$, if $\mathcal{A}, \beta \models \phi$ then $\mathcal{A}, \beta \models \psi$. The formulas $\phi$ and $\psi$ are called equivalent, written $\phi \equiv \psi$, if $\phi \models \psi$ and $\psi \models \phi$. Two formulas $\phi$ and $\psi$ are called equisatisfiable, if $\phi$ is satisfiable iff $\psi$ is satisfiable (not necessarily in the same models). Note that if $\phi$ and $\psi$ 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 $M_1 \models M_2$, written $M_1 \models M_2$, if for any algebra $\mathcal{A}$ and assignment $\beta$, 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 $M_1 \models M_2$, if $M_1 \models M_2$ and $M_2 \models M_1$. Given an arbitrary formula $\phi$ 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 $\beta$ 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 $\Sigma$-algebra. A *congruence* $\sim$ is an equivalence relation on $(S_1)^A \cup \ldots \cup (S_n)^A$ such that

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

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 $\Sigma$-algebra $\mathcal{A}$ the
interpretation of equality \( \approx^A \) 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 \( \Sigma \)-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 \( \beta \) 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 \( \neg P(x) \lor T(x) \) 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 \( \sigma : X \to T(\Sigma, X) \) so that

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

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

Substitutions are often written as sets of pairs \( \{ x_1 \mapsto t_1, \ldots, x_n \mapsto t_n \} \) if \( \text{dom}(\sigma) = \{ x_1, \ldots, x_n \} \) and \( x_i\sigma = t_i \) for every \( i \in \{1, \ldots, n\} \). The modification of a substitution \( \sigma \) 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 \( \sigma \) 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 \( \circ \in \{\lor, \land\} \),
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 \( \sigma \) to a term \( t \) (formula \( \phi \)) is called an instance of \( t (\phi) \). The substitution \( \sigma \) is called ground if it maps every domain variable to a ground term, i.e., the codomain of \( \sigma \) consists of ground terms only. If the application of a substitution \( \sigma \) to a term \( t \) (formula \( \phi \)) produces a ground term \( t\sigma \) (a variable-free formula, \( \text{vars}(\phi\sigma) = \emptyset \)), then \( t\sigma \) (\( \phi\sigma \)) is called ground instance of \( t (\phi) \) and \( \sigma \) is called grounding for \( t (\phi) \). The set of ground instances of a clause set \( N \) is given by \( \text{grd}(\Sigma, N) = \{ C\sigma \mid C \in N, \sigma \text{ is grounding for } C \} \) the set of ground instances of \( N \). A substitution \( \sigma \) is called a variable renaming if \( \text{codom}(\sigma) \subseteq \mathcal{X} \) 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 \( \beta \) be an assignment of some interpretation \( A \) of a term \( t \) and \( \sigma \) a substitution. Then

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

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

**Proof.** By structural induction on \( t \). If \( t = a \) is a constant, then \( \beta(a\sigma) = a^A = \beta[x_1 \mapsto \beta(x_1\sigma), \ldots, x_n \mapsto \beta(x_n\sigma)](a) \). The case \( t = x \) is a variable and \( x \notin \text{dom}(\sigma) \) is identical to the case that \( t \) is a constant. So \( t = x_i \) is a variable and \( x_i \in \text{dom}(\sigma) \), 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 \( x_i\sigma = s = f(s_1, \ldots, s_m) \). \( \beta(x_i\sigma) = \beta(f(s_1, \ldots, s_m)) = f^A(\beta(s_1), \ldots, \beta(s_m)) = \beta[x_1 \mapsto f(s_1, \ldots, s_m)](x_i) = \beta[x_1 \mapsto \beta(x_1\sigma), \ldots, x_n \mapsto \beta(x_n\sigma)](t) \).

For the inductive case let \( t = f(t_1, \ldots, t_m) \). Then \( \beta(t\sigma) = f^A(\beta(t_1\sigma), \ldots, \beta(t_m\sigma)) = f^A(\beta[x_1 \mapsto \beta(x_1\sigma), \ldots, x_n \mapsto \beta(x_n\sigma)](t_1), \ldots, \beta[x_1 \mapsto \beta(x_1\sigma), \ldots, x_n \mapsto \beta(x_n\sigma)](t_m)) = \beta[x_1 \mapsto \beta(x_1\sigma), \ldots, x_n \mapsto \beta(x_n\sigma)](t) \). \( \square \)
3.4 Equality

The equality predicate is built 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 \( \text{Bool} \) with a fresh constant \( \text{true} \).

\[
\text{InjEq} \quad \chi[R(t_1, \ldots, t_{1,n})]_{p_1} \ldots [R(t_m, \ldots, t_{m,n})]_{p_m} \Rightarrow_{\text{IE}} \chi[f_R(t_1, \ldots, t_{1,n}) \approx \text{true}]_{p_1} \ldots [f_R(t_m, \ldots, t_{m,n}) \approx \text{true}]_{p_m}
\]

provided \( R \) is a predicate occurring in \( \chi \), \( \{p_1, \ldots, p_m\} \) are all positions of atoms with predicate \( R \) in \( \chi \) and \( f_R \) are new with appropriate sorting.

**Proposition 3.4.1.** Let \( \chi \Rightarrow_{\text{IE}} \chi' \) then \( \chi \) is satisfiable (valid) iff \( \chi' \) is satisfiable (valid).

**Proof.** (Sketch) The basic proof idea is to establish the relation \((t_1^A, \ldots, t_{1,n}^A) \in R^A \) iff \( f_R(t_1^A, \ldots, t_{1,n}^A) = \text{true}^A \). Furthermore, the sort of \( \text{true} \) is fresh to \( \chi \) and the equations \( f_R(t_1, \ldots, t_{1,n}) \approx \text{true} \) do not interfere with any term \( t_i \) because the \( f_R \) are all fresh and only occur on top level of the equations. \( \square \)

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

\[
\text{RemEq} \quad \chi[l_1 \approx r_1]_{p_1} \ldots [l_m \approx r_m]_{p_m} \Rightarrow_{\text{RE}} \chi[E(l_1, r_1)]_{p_1} \ldots [E(l_m, r_m)]_{p_m} \land \text{def}(\chi, E)
\]

provided \( \{p_1, \ldots, p_m\} \) are all positions of equations \( l_i = r_i \) in \( \chi \) and \( E \) is a new binary predicate.

The formula \( \text{def}(\chi, E) \) is the axiomatization of equality for \( \chi \) and it consists of a conjunction of the equivalence relation axioms for \( E \):

\[
\forall x. E(x, x) \\
\forall x, y. (E(x, y) \rightarrow E(y, x)) \\
\forall x, y, z. ((E(x, y) \land E(x, z)) \rightarrow E(x, z))
\]

plus the congruence axioms for \( E \) for every \( n \)-ary function symbol \( f \).
\[\forall x_1, y_1, \ldots, x_n, y_n. (E(x_1, y_1) \land \ldots \land E(x_n, y_n)) \]
\[\rightarrow E(f(x_1, \ldots, x_n), f(y_1, \ldots, y_n))\]
plus the congruence axioms for \(E\) for every \(m\)-ary predicate symbol \(P\)
\[\forall x_1, y_1, \ldots, x_m, y_m. (E(x_1, y_1) \land \ldots \land E(x_m, y_m) \land P(x_1, \ldots, x_m)) \]
\[\rightarrow P(y_1, \ldots, y_m)\]

**Proposition 3.4.2.** Let \(\chi \Rightarrow_{RE} \chi'\) then \(\chi\) is satisfiable iff \(\chi'\) 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 \(\Rightarrow_{RE}\) is not validity preserving. Consider the simple example formula \(a \approx a\) which is valid for any constant \(a\). Its translation \(E(a, a) \land \text{def}(a \approx a, E)\) is not valid, e.g., consider an algebra with \(E^A = \emptyset\).

Now in case \(\chi\) 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 \(\Sigma\) 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 \(\Sigma\). 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 \(\Sigma\)) is a \(\Sigma\)-algebra \(H\) such that
3.5. HERBRAND’S THEOREM

1. \( S^H := T_S(\Sigma) \) for every sort \( S \in S \)

2. \( f^H : (s_1, \ldots, s_n) \mapsto f(s_1, \ldots, s_n) \) where \( f \in \Omega \), \( \text{arity}(f) = n \), \( s_i \in S^H_i \) and 
   \( f : S_1 \times \ldots \times S_n \rightarrow S \) is the sort declaration for \( f \)

3. \( P^H \subseteq (S^H_1 \times \ldots \times S^H_m) \) 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 \( \Sigma \)-algebra.

Proof. (i) the carriers are non-empty because every signature contains a constant declaration for each sort. If \( S^H \cap T^H \neq \emptyset \), then there must be two declarations for the same function symbol in \( \Sigma \) 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 \( I \)

\[ (s_1, \ldots, s_n) \in P^A \iff P(s_1, \ldots, s_n) \in I \]

Thus Herbrand interpretations (over \( \Sigma \)) can be identified with sets of \( \Sigma \)-ground atoms. A Herbrand interpretation \( I \) is called a Herbrand model of \( \phi \), where I assume \( \phi \) 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 \( \Sigma \), where for all interpretations \( S_A = \{a, b\} \).

\[ I_1 := \emptyset \]

\[ I_2 := \{P(a), Q(a, a), Q(b, b)\} \]

\[ I_3 := \{P(a), P(b), Q(a, a), Q(b, b), Q(a, b), Q(b, a)\} \]

Now consider the extension \( \Sigma' \) of \( \Sigma \) by one unary function symbol \( g : S \rightarrow S \). Then the following are examples of Herbrand interpretations over \( \Sigma' \), where for all interpretations \( S_A = \{a, b, g(a), g(b), g(g(a)), \ldots\} \).

\[ I'_1 := \emptyset \]

\[ I'_2 := \{P(a), Q(a, g(a)), Q(b, b)\} \]

\[ I'_3 := \{P(a), P(g(a)), P(g(g(a))), \ldots, Q(a, a), Q(b, b), Q(b, g(b)), Q(b, g(g(b))), \ldots\} \]
Theorem 3.5.5 (Herbrand’s Theorem). Let $\Sigma$ be a finite set of $\Sigma$-clauses without equality. Then $\Sigma$ is satisfiable iff $\Sigma$ has a Herbrand model over $\Sigma$ if and only if $\Sigma$ has a Herbrand model over $\Sigma$.

Proof. Firstly, I prove that if $\Sigma$ has a model, then it has a Herbrand model over $\Sigma$. So let $\mathcal{A}$ be a model for $\Sigma$. Since $\Sigma$ is finite let’s consider exactly the subsignature of $\Sigma$. Then $P^H = \left\{ (t_1, \ldots, t_n) \mid (t_1^A, \ldots, t_n^A) \in P^A, t_i \in T(\Sigma) \right\}$.

Finally, I need to prove that $\mathcal{H}$ is a model for $\Sigma$. Assume not. Then there is a clause $C \in N$ and an assignment $\beta_H$ such that $\mathcal{H}(\beta_H)(C) = 0$ where $\beta_H(x_i) = t_i$ for all $x_i \in \vars(C)$ 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 $\beta_A$ where $\beta_A(x_i) = t_i^A$. Since $\mathcal{A} \models N$ also $\mathcal{A}(\beta_A) \models C$, in particular, there is a literal $L \in C$ with $\mathcal{A}(\beta_A)(L) = 1$. If it is an atom $P(t_1, \ldots, t_n)$ with $(\mathcal{A}(\beta_A)(t_1), \ldots, \mathcal{A}(\beta_A)(t_n)) \in P^A$, but then $(l_1, \ldots, t_i, \sigma) \in P^H$ by definition of $\mathcal{H}$ and Lemma 3.3.2. Hence $(\mathcal{H}(\beta_H)(l_1), \ldots, \mathcal{H}(\beta_H)(l_n)) \in P^H$, 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 $\Sigma$ has a Herbrand model over $\Sigma$ if and only if $\Sigma$ has a Herbrand model. First, assume $\Sigma$ has a Herbrand model $\mathcal{H}$ over $\Sigma$. Then $\mathcal{H}$ is also a model for $\Sigma$. Assume not. Then there is a clause $C\sigma \in \Sigma$, $C \in N$, such that $\mathcal{H} \not\models C\sigma$. But then $\mathcal{H}(\beta_H[x_1 \mapsto (x_1\sigma), \ldots, x_n \mapsto (x_n\sigma)])(C) = 0$, $\text{dom}(\sigma) = \{ x_1, \ldots, x_n \}$, contradicting $\mathcal{H}$ is a model for $\Sigma$. Secondly, assume $\mathcal{H}$ is a model for $\Sigma$. Then $\mathcal{H}$ is also a model for $\Sigma$. Assume not, then there is a clause $C \in N$ and an assignment $\beta_H[x_1 \mapsto (x_1\sigma), \ldots, x_n \mapsto (x_n\sigma)]$, $\vars(C) = \{ x_1, \ldots, x_n \}$, such that $\mathcal{H}(\beta_H[x_1 \mapsto (x_1\sigma), \ldots, x_n \mapsto (x_n\sigma)])(C) = 0$. But then $\mathcal{H} \not\models C\sigma$, contradicting $\mathcal{H}$ is a model for $\Sigma(\Sigma, N)$.

Example 3.5.6 (Example of a grd($\Sigma, N$)). Consider $\Sigma'$ from Example 3.5.4 and the clause set $N = \{ Q(x, x) \lor \neg P(x), \neg P(x) \lor P(g(x)) \}$. Then the set of ground instances $\text{grd}(\Sigma', N) = \{ Q(a, a) \lor \neg P(a), Q(b, b) \lor \neg P(b), Q(g(a), g(a)) \lor \neg P(g(a)) \ldots \}$ 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 $\Sigma$) is a $\Sigma$-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. \( S^H := T_S(\Sigma)/\sim \) for every sort \( S \in \mathcal{S} \)

3. \( f^H : ([s_1], \ldots, [s_n]) \mapsto [f(s_1, \ldots, s_n)] \) 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. \( P^H \subseteq (S_1^H \times \ldots \times S_m^H) \) 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 \( \Sigma \)-algebra.

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

(ii) functions are total by definition.

(iii) relations are assigned. \( \square \)

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 \( I \)

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

\[t \sim s \iff s \approx t \in I\]

Thus Herbrand interpretations (over \( \Sigma \)) can be identified with sets of \( \Sigma \)-ground atoms. A Herbrand interpretation \( I \) is called a Herbrand model of \( \phi \), 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 \( \Sigma \), 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 \( \Sigma' \) of \( \Sigma \) by one unary function symbol \( g : S \to S \). Then the following are examples of Herbrand interpretations over \( \Sigma' \), where for all interpretations \( S_\mathcal{A} = \{a, b, g(a), g(b), g(g(a)), \ldots\} \).
\[ I_1' := \emptyset \]
\[ I_2' := \{ P(a), Q(a, g(a)), Q(b, b) \} \]
\[ I_3' := \{ P(a), P(g(a)), P(g(a))), \ldots, Q(a, a), Q(b, b), Q(b, g(b)), Q(b, g(b))) \ldots \} \]

**Theorem 3.5.11** (Herbrand with Equality). Let \( N \) be a finite set of \( \Sigma \)-clauses. Then \( N \) is satisfiable iff \( N \) has a Herbrand model over \( \Sigma \) iff \( \text{grd}(\Sigma, N) \) has a Herbrand model over \( \Sigma \).

**Proof.** Firstly, I prove that if \( N \) has a model, then it has a Herbrand model over \( \Sigma \). So let \( A \) be a model for \( N \). Since \( N \) is finite let’s consider exactly the signature \( \Sigma \) of \( N \). The equivalence relation \( \sim \) is defined by \( t \sim s \) if \( t^A = s^A \) for all terms \( t, s \in T(\Sigma) \). Obviously, \( \sim \) is an equivalence relation. In addition, \( \sim \) is well-sorted. Assume not, so \( t \sim s \) but \( t \in T_T(\Sigma) \) and \( s \in T_S(\Sigma) \) with \( T \neq S \), but then \( S^A \cap T^A \neq \emptyset \), a contradiction. Then \( P^H = \{(\{t_1\}, \ldots, \{t_n\}) \mid (t_1^A, \ldots, t_n^A) \in P^A, t_i \in T(\Sigma)\} \). Finally, I need to prove that \( H \) is a model for \( N \). Assume not. Then there is a clause \( C \in N \) and an assignment \( \beta_H \) such that \( H(\beta_H)(C) = 0 \) where \( \beta_H(x_i) = [t_i] \) 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 \( \beta_A \) where \( \beta_A(x_i) = t_i^A \). Since \( A \models N \) also \( A(\beta_A) \models C \), in particular, there is a literal \( L \in C \) with \( A(\beta_A)(L) = 1 \). The literal \( L \) can have the following form: (i) it is an equation \( l \approx r \) with \( A(\beta_A)(l) = A(\beta_A)(r) \), but then \( l \sigma \sim r \sigma \) by definition of \( H \) and Lemma 3.3.2. Hence \( H(\beta_H)(l) = H(\beta_H)(r) \), a contradiction. (ii) it is an atom \( P(l_1, \ldots, l_n) \) with \( A(\beta_A)(l_1), \ldots, A(\beta_A)(l_n) \in P^A \), but then \( ([l_1 \sigma], \ldots, [l_n \sigma]) \in P^H \) by definition of \( H \) and Lemma 3.3.2. Hence \( H(\beta_H)(l_1), \ldots, H(\beta_H)(l_n) \in P^H \), a contradiction. The cases where \( L \) is negative are 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 \( \Sigma \) iff \( \text{grd}(\Sigma, N) \) has a Herbrand model. Firstly, assume \( N \) has a Herbrand model \( H \) over \( \Sigma \). Then \( H \) 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 \( H \not\models C \sigma \). But then \( H(\beta_H[x_1 \mapsto (x_1 \sigma), \ldots, x_n \mapsto (x_n \sigma)])(C) = 0 \), \( \text{dom}(\sigma) = \{x_1, \ldots, x_n\} \), contradicting \( H \) is a model for \( N \). Secondly, assume \( H \) is a model for \( \text{grd}(\Sigma, N) \). Then \( H \) is also a model for \( N \). Assume not, then there is a clause \( C \in N \) and ad assignment \( \beta_H[x_1 \mapsto (x_1 \sigma), \ldots, x_n \mapsto (x_n \sigma)] \), \( \text{vars}(C) = \{x_1, \ldots, x_n\} \), such that \( H(\beta_H[x_1 \mapsto (x_1 \sigma), \ldots, x_n \mapsto (x_n \sigma)])(C) = 0 \). But then \( H \not\models C \sigma \), contradicting \( H \) is a model for \( \text{grd}(\Sigma, N) \).

Example 3.5.12 (Example of a \( \text{grd}(\Sigma, N) \)). Consider \( \Sigma' \) from Example 3.5.4 and the clause set \( N = \{ Q(x, x) \lor \neg P(x), \neg P(x) \lor P(g(x)) \} \). Then the set of ground instances \( \text{grd}(\Sigma', N) = \{ \)
Chapter 6

Decidable Logics

This chapter is about decidable logics. There are many decidable fragments
of first-order logic, some of them are discussed in Chapter 3 and Chapter 5.
Here I discuss logics that are typically not representable in first-order logic,
e.g., linear integer arithmetic, Section 6.2, or logics where specialized decision
procedures exist, beyond the general procedures discussed in previous chapters,
e.g., equational reasoning on ground terms by congruence closure, Section 6.1,
that can also be solved by Knuth-Bendix completion, Chapter 4.

6.1 Congruence Closure

In general, satisfiability of first-order formulas with respect to equality is un-
decidable. Even the word problem for conjunctions of equations is undecidable.
However, I will show that satisfiability is decidable for ground first-order formu-
las.

It suffices to consider conjunctions of literals. Arbitrary ground formulas can
be converted into DNF, potentially at the price of an exponential blow up. A
formula in DNF is satisfiable if and only if one of its conjunctions is satisfiable.
So it is sufficient to consider a conjunction of ground literals, e.g., a conjunction
of ground equations.

Note that the problem can be written in several ways. An equational clause

\[ \forall \bar{x} (t_1 \approx s_1 \lor \ldots \lor t_n \approx s_n \lor t_1 \not\approx r_1 \lor \ldots \lor t_k \not\approx r_k) \]

is valid iff

\[ \exists \bar{x} (t_1 \not\approx s_1 \land \ldots \land t_n \not\approx s_n \land t_1 \approx r_1 \land \ldots \land t_k \approx r_k) \]

is unsatisfiable iff the Skolemized (ground!) formula

\[ (t_1 \not\approx s_1 \land \ldots \land t_n \not\approx s_n \land t_1 \approx r_1 \land \ldots \land t_k \approx r_k)\{ \bar{x} \mapsto \bar{c} \} \]

is unsatisfiable iff the formula

\[ (t_1 \not\approx s_1 \land \ldots \land t_n \not\approx s_n \land t_1 \approx r_1 \land \ldots \land t_k \approx r_k)\{ \bar{x} \mapsto \bar{c} \} \]
The goal of the procedure is to check (un-)satisfiability of a ground conjunction

$$(t_1 \approx s_1 \lor \ldots \lor t_n \approx s_n \lor l_1 \neq r_1 \lor \ldots \lor l_k \neq r_k) \{\vec{x} \mapsto \vec{c}\}$$

is valid.

Please note validity of these transformations do depends on the shape of the (starting) formula. Validity is no preserved in case of a quantifier alternation or an existentially quantified formula, in general, or the eventual formula must not be ground. There is no way to transform a first-order (equational) formula into a ground formula preserving validity, in general.

The theory is also known as EUF (equality with uninterpreted function symbols) and one of the standard theories considered in SMT (Satisfiability Modulo Theories). The decision procedure discussed here is based on congruence closure.

The goal of the procedure is to check (un-)satisfiability of a ground conjunction

$$s_1 \not\approx t_1 \land \ldots \land s_k \not\approx t_k \land l_1 \approx r_1 \land \ldots \land l_n \approx r_n$$

The main idea is to transform the equations $E = \{l_1 \approx r_1, \ldots, l_n \approx r_n\}$ into an equivalent convergent TRS $R$ and check whether $s_i \downarrow_R = t_i \downarrow_R$. If $s_i \downarrow_R = t_i \downarrow_R$ for some $i$ then because $s_i \downarrow_R = t_i \downarrow_R$ iff $s_i \leftrightarrow_E t_i$ iff $E \models s_i \approx t_i$ (see Chapter 4) the overall conjunction is unsatisfiable. If $s_i \downarrow_R = t_i \downarrow_R$ for no $i$, i.e., $s_i \not\approx t_i \downarrow_R$ for all $i$ then $E_E$ is a model of both the equations $l_i \approx r_i$, and the inequations $s_j \not\approx t_j$. Hence the overall conjunction is satisfiable.

Knuth-Bendix completion, Chapter 4, can be used to convert $E$ into an equivalent convergent TRS $R$. If done properly, Knuth-Bendix completion always terminates for ground inputs. However, for the ground case, the procedure can be further optimized.

The first step is to introduce additional “names”, i.e, extra constants for all non-constant subterms. This implements implicitly sharing among subterms.

Let $E = l_1 \approx r_1 \land \ldots \land l_n \approx r_n$.

**Flattening** $E[f(t_1, \ldots, t_n)]_{p_1, \ldots, p_k} \Rightarrow_{CCF} E[c/p_1, \ldots, p_k] \land f(t_1, \ldots, t_n) \approx c$

provided all $t_i$ are constants, the $p_j$ are all positions in $E$ of $f(t_1, \ldots, t_n)$, $|p_k| > 2$ for some $k$, or, $p_k = m.2$ and $E|_{m.1}$ is not a constant for some $m$, and $c$ is fresh.

Here I consider $E$ to be a conjunction of equations in order for the positions to make sense. Note that after applying flattening to some term $f(t_1, \ldots, t_n)$ it cannot be applied a second time, because the position $p$ pointing to $f(t_1, \ldots, t_n)$ in $E[c/p_1, \ldots, p_k] \land f(t_1, \ldots, t_n) \approx c$ has size 2, i.e., $|p| = 2$.

For example, the system $E = [g(a, h(h(b))) \approx h(a)]$ is eventually replaced by $E = [h(b) \approx c_3 \land h(c_3) \approx c_4 \land h(a) \approx c_4 \land g(a, c_4) \approx c_3]$.

As a result: only two kinds of equations left. Term equations: $f(c_1, \ldots, c_n) \approx c_{i_0}$ and constant equations: $c_1 \approx c_j$. This can be further explored in an implementation by specific data structures. In particular, a union-find data structure.
6.1. CONGRUENCE CLOSURE

efficiently represents the equivalence classes encoded by the constant equations (rules).

The congruence closure algorithm is presented as a set of abstract rewrite rules operating on a pair of equations \( E \) and a set of rules \( R \), similar to Knuth-Bendix completion, Section 4.4.

\[
(E_0; R_0) \Rightarrow_{CC} (E_1; R_1) \Rightarrow_{CC} (E_2; R_2) \Rightarrow_{CC} \ldots
\]

At the beginning, \( E = E_0 \) is a set of constant equations and \( R = R_0 \) is the set of term equations oriented from left-to-right. At termination, \( E \) is empty and \( R \) contains the result. By exhaustive application of Flattening any conjunction of equations can be transformed into this form, preserving satisfiability. Recall that the atom \( s \approx t \) denotes either \( s \approx t \) or \( t \approx s \).

\[
\text{Simplify} \quad (E \cup \{c \approx c'\}; R \cup \{c \rightarrow c''\}) \Rightarrow_{CC} (E \cup \{c' \approx c''\}; R \cup \{c \rightarrow c''\})
\]

\[
\text{Delete} \quad (E \cup \{c \approx c\}; R) \Rightarrow_{CC} (E; R)
\]

\[
\text{Orient} \quad (E \cup \{c \approx c'\}; R) \Rightarrow_{CC} (E; R \cup \{c \rightarrow c'\}) \\
\text{if } c \triangleright c'
\]

\[
\text{Deduce} \quad (E; R \cup \{t \rightarrow c, t \rightarrow c'\}) \Rightarrow_{CC} (E \cup \{c \approx c'\}; R \cup \{t \rightarrow c\})
\]

\[
\text{Collapse} \quad (E; R \cup \{t[c]' \rightarrow c', c \rightarrow c''\}) \Rightarrow_{CC} (E; R \cup \{t[c]' \rightarrow c', c \rightarrow c''\}) \\
p \neq \epsilon
\]

For rule Deduce, \( t \) is either a term of the form \( f(c_1, \ldots, c_n) \) or a constant \( c_i \). For rule Collapse, \( t \) is always of the form \( f(c_1, \ldots, c_n) \) For ground rewrite rules, critical pair computation does not involve substitution. Therefore, every critical pair computation can be replaced by a simplification, either using Deduce or Collapse.

The inference rules are usually applied according to the following strategy: Simplify, Delete and Orient are preferred over Deduce and Collapse. Then if Collapse becomes applicable, it is exhaustively applied followed by an application of Deduce.

Instead of fixing the ordering \( \triangleright \) in advance, it is preferable to define it on the fly during the algorithm: if an equation \( c \approx c' \) between two constants is oriented, a good heuristic is to make that constant symbol larger that occurs less often in \( R \), hence producing afterwards fewer Collapse steps.

The average runtime of the algorithm is \( O(m \log m) \), where \( m \) is the number of edges in the graph representation of the initial constant and term equations.

The inference rules are sound in the usual sense. The conclusions are entailed by the premises, so every model of the premises is a model of the conclusions.
For the initial flattening rule, however, only a weaker result holds. The models of the original equations have to be extended by interpretations for the freshly introduced constants to obtain models of the flattened equations. The result is a new algebra with the same universe as the old one, with the same interpretations for old functions and predicate symbols, but with appropriately chosen interpretations for the new constants.

Consequently, the relations $\approx_E$ and $\approx_R$ for the original $E$ and the final $R$ are not the same. On the other hand, the model extension preserves the universe and the interpretations for old symbols. Therefore, if $s$ and $t$ are terms over the old symbols, we have $s \approx_E t$ iff $s \approx_R t$. This is sufficient for our purposes: The terms $s_i$ and $t_i$ that we want to normalize using $R$ do not contain new symbols.

6.1.1 History

Congruence closure algorithms have been published, among others, by Shostak (1978), by Nelson and Oppen (1980), and by Downey, Sethi and Tarjan (1980). Kapur (1997) showed that Shostak’s algorithm can be described as a completion procedure.

Bachmair and Tiwari (2000) did this also for the Nelson/Oppen and the Downey/Sethi/Tarjan algorithm.

The algorithm presented here is the Downey/Sethi/Tarjan algorithm in the presentation of Bachmair and Tiwari.

6.2 Linear Arithmetic

There are several ways of introducing linear arithmetic and in particular its syntax. I start with a syntax that already contains $-$, $\leq$, $<$, $\geq$, $\neq$ and $\mathbb{Q}$. All these functions and relations are indeed expressible by first-order formulas over 0, 1, $\approx$, and $\succ$. For the semantics there are two approaches. Either providing axioms, i.e., closed formulas, for the above symbols and then considering all algebras satisfying the axioms, or fixing one particular algebra or a class of algebras. For this chapter I start with a rich syntax and a semantics based on a fixed algebra.

Definition 6.2.1 (LA Syntax). The syntax of LA is

$$\Sigma_{LA} = (\{LA\}, \{0, 1, +, -\} \cup \mathbb{Q}, \{\leq, <, \neq, >, \geq\})$$

where $-$ is unitary and all other symbols have the usual arities.

Terms and formulas over $\Sigma_{LA}$ are built in the classical free first-order way, see Section 3.1. All first-order notions, i.e., terms, atoms, equations, literals, clauses, etc. carry over to LA formulas. The atoms and terms built over the LA signature are written in their standard infix notation, i.e., I write $3 + 5$ instead of $+(3, 5)$. Note that the signature does not contain multiplication. A term $3x$ is just an abbreviation for a term $x + x + x$. 
6.2. LINEAR ARITHMETIC

For the semantics I start with considering as the domain the rationals, \( \mathbb{Q} \). As long as coefficients are from the integers, with respect to the satisfiability, validity of a formula the rationals cannot be distinguished from the reals. Restricting the domain from the rationals to the integers, however, results in a difference in satisfiability, validity of a formula, in general. In this case the signature is restricted to integer constants as well.

**Definition 6.2.2** (Linear Rational Arithmetic Standard Semantics). The \( \Sigma_{LA} \) algebra \( A_{LRA} \) is defined by \( LA^{A_{LRA}} = \mathbb{Q} \) and all other signature symbols are assigned the standard interpretations over the rationals.

Due to the expressive LA language there is no need for negative literals, because \((\neg <)^{A_{LRA}} = (\geq)^{A_{LRA}}, (\neg >)^{A_{LRA}} = (\leq)^{A_{LRA}}, \) and \((\neg \approx)^{A_{LRA}} = (\neq)^{A_{LRA}}.\]

Note the difference between the above standard semantics over \( \Sigma_{LA} \) and the free first-order semantics over \( \Sigma_{LA} \), Definition 3.2.1. The equation \( 3 + 4 \approx 5 \) has a model in the free first-order semantics, hence it is satisfiable, whereas in the standard model of linear rational arithmetic, Definition 6.2.2, the equation \( 3 + 4 \approx 5 \) is false. In addition, with respect to the standard LRA semantics the definitions of validity, satisfiability coincide with truth and the definition of unsatisfiability coincides with falsehood. This is the result of a single algebra semantics.

6.2.1 Fourier-Motzkin Quantifier Elimination

It is decidable whether a first-order formula over \( \Sigma_{LA} \) is true or false in the standard LRA semantics. This was first discovered in 1826 by J. Fourier and re-discovered by T. Motzkin in 1936 and is called FM for short. Note that validity of a \( \Sigma_{LA} \) formula with respect to the standard semantics is undecidable, Exercise ??

Similar to Congruence Closure, Section 6.1, the starting point of the procedure is a conjunction of atoms without atoms of the form \( \neq \). These will eventually be replaced by a disjunction, i.e., an atom \( t \neq s \) is replaced by \( t < s \lor t > s \).

Every atom over the variables \( x, y_1, \ldots, y_n \) can be converted into an equivalent atom \( x \circ t[y] \) or \( 0 \circ t[y] \), where \( \circ \in \{<,>,\leq,\geq,\approx\} \) and \( t[y] \) has the form \( \sum q_i y_i + q_0 \) where \( q_i \in \mathbb{Q} \). In other words, a variable \( x \) can be either isolated on one side of the atom or eliminated completely. This is the starting point of the FM calculus deciding a conjunction of LA atoms without \( \neq \) modulo the isolation of variables and the reduction of ground formulas to \( \top, \bot \).

The calculus operates on a set of atoms \( N \). The normal forms are conjunctions of atoms \( sot \) where \( s, t \) do not contain any variables. These can be obviously eventually reduced to \( \top \) or \( \bot \). The FM calculus consists of two rules:

**Substitute** \[ N \uplus \{ x \approx t \} \Rightarrow_{FM} N \{ x \mapsto t \} \]

provided \( x \) does not occur in \( t \)

**Eliminate** \[ N \uplus L_i \{ x \circ_i^1 t_i \} \uplus \bigcup_j \{ x \circ_j^2 s_j \} \Rightarrow_{FM} N \cup L_{i,j} \{ t_i \circ_{i,j} s_j \} \]
provided \( x \) does not occur in \( N \) nor in the \( t_i, s_j, c_1^i \in \{<,\leq\}, c_2^j \in \{>,\geq\} \), and
\( \alpha_{i,j} = > \) if \( c_1^i = < \) or \( c_2^j = > \), and \( \alpha_{i,j} = \geq \) otherwise.

If all variables in \( N \) are implicitly existentially quantified, i.e., \( N \) stands for \( \exists x. N \), then the above two rules constitute a sound and complete decision procedure for conjunctions of LA atoms without \( \not\approx \).

**Lemma 6.2.3** (FM Termination on a Conjunction of Atoms). FM terminates on a conjunction of atoms.

**Proof.** Any rule applications strictly reduces the number of variables. \( \square \)

**Lemma 6.2.4** (FM Soundness and Completeness on a Conjunction of Atoms). \( N \Rightarrow^*_{FM} \top \) iff \( A_{LRA} \models \exists x. N \). \( N \Rightarrow^*_{FM} \bot \) iff \( A_{LRA} \not\models \exists x. N \).

**Proof.** \( \Rightarrow : \) Assume that \( A_{LRA}(\beta) \models N \) for some \( \beta \). Proof by case analysis on the two rules. For rule Substitute obviously \( A_{LRA}(\beta)(x) = A_{LRA}(\beta)(t) \) hence \( A_{LRA}(\beta) \models N \{x \mapsto t\} \). For rule Eliminate obviously \( A_{LRA}(\beta)(x) c_1^i A_{LRA}(\beta)(t_i) \) and \( A_{LRA}(\beta)(x) c_2^j A_{LRA}(\beta)(s_j) \). A case analysis on \( c_1^i, c_2^j \) yields \( A_{LRA}(\beta) \models t_i \alpha_{i,j} s_j \) for all \( i,j \).

\( \Leftarrow : \) Again by a case analysis on the rules. For rule Substitute if \( A_{LRA}(\beta) \models N \{x \mapsto t\} \) then \( A_{LRA}(\beta[x \mapsto A_{LRA}(\beta)(t)]) \models N \cup \{x \approx t\} \). For rule Eliminate if \( A_{LRA}(\beta) \models N \cup \bigcup_{i,j} \{t_i \alpha_{i,j} s_j\} \) then \( A_{LRA}(\beta[x \mapsto \frac{1}{2}(\min(\cup_i t_i) + \max(\cup_j s_j))] \models N \cup \bigcup_i \{x c_1^i t_i\} \cup \bigcup_j \{x c_2^j s_j\} \). \( \square \)

The FM calculus on conjunctions of atoms can be extended to arbitrary closed LRA first-order formulas \( \phi \). I always assume that different quantifier occurrences in \( \phi \) bind different variables. This can always be obtained by renaming one variable. The first step is to eliminate \( \top, \bot \) from \( \phi \) and to transform \( \phi \) in negation normal form, see Section 3.9. The resulting formula only contains the operators \( \forall, \exists, \land, \lor, \neg \), where all negation symbols occur in front of atoms. Then the following rule can be used to remove the negation symbols as well:

\[ \text{ElimNeg} \quad \chi[\neg s \circ^1 t]_p \Rightarrow_{FM} \chi[s \circ^2 t]_p \]

where the pairs \( (c_1, c_2) \) are given by pairs \( (<,>), (\leq,\geq), (\approx,\not\approx) \) and their symmetric variants.

The above two FM rules on conjunctions cannot cope with atoms \( s \not\approx t \), so they are eliminated as well:

\[ \text{Elim\not\approx} \quad \chi[s \not\approx t]_p \Rightarrow_{FM} \chi[s < t \lor s > t]_p \]

The next step is to compute a **Prenex Normal Form**, a formula \( \{\exists, \forall\} x_1 \ldots \{\exists, \forall\} x_n. \phi \) where \( \phi \) does not contain any quantifiers. This can be done by simply applying the mini-scoping rules, see Section 3.9, in the opposite direction:
6.2. LINEAR ARITHMETIC

\textbf{Prenex1} \quad \chi([\forall x.\psi_1] \circ \psi_2)_p \Rightarrow_{\text{FM}} \chi[\forall x.(\psi_1 \circ \psi_2)]_p \\
provided \circ \in \{\land, \lor\}, x \notin \text{fvars}(\psi_2)

\textbf{Prenex2} \quad \chi([\exists x.\psi_1] \circ \psi_2)_p \Rightarrow_{\text{FM}} \chi[\exists x.(\psi_1 \circ \psi_2)]_p \\
provided \circ \in \{\land, \lor\}, x \notin \text{fvars}(\psi_2)

\textbf{Prenex3} \quad \chi([\forall x.\psi_1] \land (\forall y.\psi_2)]_p \Rightarrow_{\text{FM}} \chi[\forall x.(\psi_1 \land \psi_2\{y \mapsto x\})]_p

\textbf{Prenex4} \quad \chi([\exists x.\psi_1] \lor (\exists y.\psi_2)_p \Rightarrow_{\text{FM}} \chi[\exists x.(\psi_1 \lor \psi_2\{y \mapsto x\})]_p

where Prenex3 and Prenex4 are preferred over Prenex1 and Prenex2. Finally, for the resulting formula \{\exists, \forall\}x_1 \ldots \{\exists, \forall\}x_n, \phi in prenex normal form the FM algorithm computes a DNF of \phi by exhaustively applying the rule PushConj, Section 2.5.2. The result is a formula \{\exists, \forall\}x_1 \ldots \{\exists, \forall\}x_n, \phi where \phi is a DNF of atoms without containing an atom of the form \(s \neq t\). All the above formulas transformations are equivalence preserving. Therefore, to each conjunct of \phi the above two FM rules decide the conjunct, if all variables are existentially quantified. This is the final obstacle in order to obtain the FM decision procedure for arbitrary formulas.

It is solved by considering the quantifiers iteratively in an innermost way. For the formula \{\exists, \forall\}x_1 \ldots \{\exists, \forall\}x_n, \phi always the innermost quantifier \{\exists, \forall\}x_n is considered. If it is an existential quantifier, \exists x_n, then the FM rules Substitute, Eliminate are applied to the variable \(x_n\) for each conjunct \(C_i\) of \(\phi = C_1 \lor \ldots \lor C_n\). The result is a formula \{\exists, \forall\}x_1 \ldots \{\exists, \forall\}x_{n-1}, (C'_1 \lor \ldots \lor C'_n) which is again in prenex DNF. Furthermore, by Lemma 6.2.4 it is equivalent to \{\exists, \forall\}x_1 \ldots \{\exists, \forall\}x_n. \phi. If the innermost quantifier is a universal quantifier \forall x_n, then the formula is replaced by \{\exists, \forall\}x_1 \ldots \{\exists, \forall\}x_{n-1}, \exists x_n, \neg \phi and the above steps for negation normal form and DNF are repeated for \neg \phi resulting in an equivalent formula \{\exists, \forall\}x_1 \ldots \{\exists, \forall\}x_{n-1}, \exists x_n, \phi' where \phi' is in DNF and does not contain negation symbols nor atoms \(s \neq t\). Then the FM rules Substitute, Eliminate are applied to the variable \(x_n\) for each conjunct \(C_i\) of \(\phi' = C_1 \lor \ldots \lor C_n\). The result is an equivalent formula \{\exists, \forall\}x_1 \ldots \{\exists, \forall\}x_{n-1}, \neg (C'_1 \lor \ldots \lor C'_n). Finally, the above steps for negation normal form and DNF are repeated for \neg(C'_1 \lor \ldots \lor C'_n) resulting in an equivalent formula \{\exists, \forall\}x_1 \ldots \{\exists, \forall\}x_{n-1}, \phi'' where \phi'' is in DNF and does not contain negation symbols nor atoms \(s \neq t\). This completes for FM decision procedure for LRA formulas.

Every LRA formula can be reduced to \(\top\) or \(\bot\) via the FM decision procedure. Therefore LRA is called a complete theory, i.e., every closed formula over the signature of LRA is either true or false.

LA formulas over the rationals and over the reals are indistinguishable by first-order formulas over the signature of LRA. These properties do not hold for extended signatures, e.g., then additional free symbols are introduced. Furthermore, FM is no decision procedures over the integers, even if the LA syntax is restricted to integer constants.
The complexity of the FM calculus depends mostly on the quantifier alternations in \( \exists \forall x_1 \ldots \exists \forall x_n \phi \). In case an existential quantifier \( \exists \) is eliminated, the formula size grows worst-case quadratically, therefore \( O(n^2) \) runtime. For \( m \) quantifiers \( \exists \ldots \exists \) a naive implementation needs worst-case \( O(n^{2m}) \) runtime. It is not known whether an optimized implementation with simply exponential runtime is possible. If there are \( m \) quantifier alternations \( \exists \forall \exists \forall \ldots \exists \forall \), a CNF to DNF conversion is required after each step. Each conversion has a worst-case exponential run time, see Section 2.5. Therefore, the overall procedure has a worst-case non-elementary runtime.

There are meanwhile more efficient decision procedures for the theory LRA known, e.g., see Section 6.2.3. There are problems occurring in practice where the elimination of a variable via FM results in an only linear increase in size. In such cases FM is still valuable. Many state-of-the-art LRA procedures actually calculate the size of the formula after eliminating a variable via FM and redundancy elimination and decide on this basis whether FM is applied or not.

### 6.2.2 Simplex

The Simplex algorithm is the prime algorithm for solving optimization problems of systems of linear inequations \([46]\) over the rationals. For automated reasoning optimization at the level of conjunctions of inequations is not in focus. Rather, solvability of a set of linear inequations as a subproblem of some theory combination is the typical application. In this context the simplex algorithm is useful as well, due to its incremental nature. If an inequation \( t \circ c, \circ \in \{\leq, \geq, <, >\} \), \( t = \sum a_i x_i, a_i, c \in \mathbb{Q} \), is added to a set \( N \) of inequations where the simplex algorithm has already found a solution for \( N \), the algorithm needs not to start from scratch. Instead it continues with the solution found for \( N \). In practice, it turns out that then typically only few steps are needed to derive a solution for \( N \cup \{t \circ d\} \) if it exists.

The simplex algorithm introduced in this section is a simplified version of the classical dual simplex used for solving optimization problems.

First, I show the case for non-strict inequations. Starting point is a set \( N \) (conjunction) of (non-strict) inequations of the form \( \sum_{x_j \in X} a_{i,j} x_j \circ_i c_i \) where \( \circ_i \in \{\geq, \leq\} \) for all \( i \). Note that an equation \( \sum a_i x_i = c \) can be encoded by two inequations \( \{ \sum a_i x_i \leq c, \sum a_i x_i \geq c \} \).

The variables occurring in \( N \) are assumed to be totally ordered by some ordering \( \prec \). The ordering \( \prec \) will eventually guarantee termination of the simplex algorithm, see Definition 6.2.10 and Theorem 6.2.11 below. I assume the \( x_j \) to be all different, without loss of generality \( x_j \prec x_{j+1} \), and I assume that all coefficients are normalized by the gcd of the \( a_{i,j} \) for all \( j \): if the gcd is different from 1 for one inequation, it is used for division of all coefficients of the inequation.
6.2. LINEAR ARITHMETIC

The goal is to decide whether there exists an assignment \( \beta \) from the \( x_j \) into \( \mathbb{Q} \) such that \( \text{LRA}(\beta) \models \bigwedge_i \left[ (\sum_{x_j \in X} a_{i,j} x_j) \circ_i c_i \right] \), or equivalently, \( \text{LRA}(\beta) \models N \). So the \( x_j \) are free variables, i.e., placeholders for concrete values, i.e., existentially quantified.

The first step is to transform the set \( N \) of inequations into two disjoint sets \( E, B \) of equations and simple bounds, respectively. The set \( E \) contains equations of the form \( y_i \approx \sum_{x_j \in X} a_{i,j} x_j \), where the \( y_i \) are fresh and the set \( B \) contains the respective simple bounds \( y_i \circ_i c_i \). In case the original inequation from \( N \) was already a simple bound, i.e., of the form \( x_j \circ_j c_j \) it is simply moved to \( B \). If in \( N \) left hand sides of inequations \( (\sum_{x_j \in X} a_{i,j} x_j) \circ_i c_i \) are shared, it is sufficient to introduce one equation for the respective left hand side. The \( y_i \) are also part of the total ordering \( \prec \) on all variables. Clearly, for any assignment \( \beta \) and its respective extension on the \( y_i \), the two representations are equivalent:

\[
\text{LRA}(\beta) \models N
\]

iff

\[
\text{LRA}(\beta[y_i \mapsto \beta(\sum_{x_j \in X} a_{i,j} x_j)]) \models E
\]

and

\[
\text{LRA}(\beta[y_i \mapsto \beta(\sum_{x_j \in X} a_{i,j} x_j)]) \models B.
\]

Given \( E \) and \( B \) a variable \( z \) is called dependent if it occurs on the left hand side of an equation in \( E \), i.e., there is an equation \( z \approx \sum_{x_j \in X} a_{i,j} x_j \in E \), and in case such a defining equation for \( z \) does not exist in \( E \) the variable \( z \) is called independent. Note that by construction the initial \( y_i \) are all dependent and do not occur on the right hand side of an equation.

Given a dependant variable \( x \), an independent variable \( y \), and a set of equations \( E \), the pivot operation exchanges the roles of \( x, y \) in \( E \) where \( y \) occurs with non-zero coefficient in the defining equation of \( x \). Let \((x \approx ay + t) \in E \) be the defining equation of \( x \) in \( E \). When writing \((x \approx ay + t) \) for some equation, I always assume that \( y \notin \text{vars}(t) \). Let \( E' \) be \( E \) without the defining equation of \( x \). Then

\[
\text{piv}(E, x, y) := \{ y \approx \frac{1}{a} x + \frac{1}{-a} t \} \cup E' \{ y \mapsto \left( \frac{1}{a} x + \frac{1}{-a} t \right) \}.
\]

Given an assignment \( \beta \), an independent variable \( y \), a rational value \( c \), and a set of equations \( E \) then the update of \( \beta \) with respect to \( y, c \) and \( E \) is

\[
\text{upd}(\beta, y, c, E) := \beta[y \mapsto c, \{ x \mapsto \beta[y \mapsto c](t) \mid x \approx t \in E \}].
\]

A Simplex problem state is a quintuple \((E; B; \beta; S; s)\) where \( E \) is a set of equations; \( B \) a set of simple bounds; \( \beta \) an assignment to all variables in \( E, B \); \( S \) a set of derived bounds, and \( s \) the status of the problem with \( s \in \{ \top, \mathrm{IV, DV, \bot} \} \). The state \( s = \top \) indicates that \( \text{LRA}(\beta) \models S \); the state \( s = \mathrm{IV} \) that potentially
LRA(β) \nmodels x \circ c \text{ for some independent variable } x, \ x \circ c \in S; \text{ the state } s = DV \text{ that LRA(β) \models x \circ c \text{ for all independent variables } x, \ x \circ c \in S}; \text{ but potentially LRA(β) \nmodels x' \circ c' \text{ for some dependent variable } x', \ x' \circ c' \in S; \text{ and the state } s = \bot \text{ that the problem is unsatisfiable. In particular, the following states can be distinguished: }

\begin{align*}
(E; B; \beta_0; \emptyset; \top) & \text{ is the start state for } N \text{ and its transformation into } E, \text{ B, and assignment } \beta_0(x) := 0 \text{ for all } x \in \text{vars}(E \cup B) \\
(E; \emptyset; \beta; S; \top) & \text{ is a final state, where LRA(β) \models E \cup S \text{ and hence the problem is solvable.} } \\
(E; B; \beta; S; \bot) & \text{ is a final state, where } E \cup B \cup S \text{ has no model.}
\end{align*}

Important invariants of the simplex rules are: (i) for every dependent variable there is exactly one equation in E defining the variable and (ii) dependent variables do not occur on the right hand side of an equation, (iii) LRA(β) \models E. These invariants are maintained by a pivot (piv) or an update (upd) operation. Here are the rules:

\begin{align*}
\text{EstablishBound} & \quad \Rightarrow_{\text{SIMP}} (E; B; \beta; S \cup \{x \circ c\}; IV) \\
\text{AckBounds} & \quad (E; B; \beta; S; s) \Rightarrow_{\text{SIMP}} (E; B; \beta; S; \top) \\
\text{FixIndepVar} & \quad (E; B; \beta; S; \emptyset) \Rightarrow_{\text{SIMP}} (E; B; \beta; S; \emptyset) \\
\text{AckIndepBound} & \quad (E; B; \beta; S; \emptyset) \Rightarrow_{\text{SIMP}} (E; B; \beta; S; \emptyset) \\
\text{FixDepVar} & \quad (E; B; \beta; S; x \circ c) \Rightarrow_{\text{SIMP}} (E; B; \beta; S; x \circ c) \\
\text{FailBounds} & \quad (E; B; \beta; S; \emptyset) \Rightarrow_{\text{SIMP}} (E; B; \beta; S; \bot)
\end{align*}
if \((x \leq c) \in S\), \(x\) dependent, LRA(\(\beta\)) \(\not\models x \leq c\) and there is no independent variable \(y\) and equation \((x \approx ay + t) \in E\) where \((a < 0\) and \(\beta(y) < c'\) for all \((y \leq c') \in S\) or \((a > 0\) and \(\beta(y) > c'\) for all \((y \geq c') \in S\)

\[
\text{FixDepVar} \geq (E; B; \beta; S; \text{DV}) \Rightarrow \text{SIMP} \ (E; B; \beta; S; \bot)
\]

if \((x \geq c) \in S\), \(x\) dependent, \(\beta \not\models_{\text{LA}} x \geq c\) and there is no independent variable \(y\) and equation \((x \approx ay + t) \in E\) where \((a > 0\) and \(\beta(y) < c'\) for all \((y \leq c') \in S\) or \((a < 0\) and \(\beta(y) > c'\) for all \((y \geq c') \in S\)

The simplex rules satisfy a number of invariants that eventually lead to proofs for soundness, completeness and termination. A state \((E; B; \beta; \emptyset; \top)\) is called an \textit{start state} if \(E\) is a finite set of equations \(x_i \approx \sum a_{i,j,y_j}\) such that the \(x_i\) occur only on left hand sides and only once in \(E\), and \(B\) is a finite set of simple bounds \(z_i \circ c\) where \(z_i\) occurs in \(E\) and \(c \in \{\text{leq, ge}\}\), and \(\beta\) maps all variables to 0.

Example 6.2.5 (Simplex Detecting Satisfiability). Consider the equational system

\[E = \{2y + x \geq 1, y - x \leq -2, x \geq 0\}\]

which results after preprocessing in the sets \(E_0 = \{z_1 \approx 2y + x, z_2 \approx y - x\}\) and \(B_0 = \{z_1 \geq 1, z_2 \leq -2, x \geq 0\}\).

Starting with an initial assignment \(\beta_0\) that maps all variables to 0 and hence satisfies \(E_0\), a Simplex run is as follows. Each line gets a number and I make references to the components of the simplex state of previous lines with respect to the line number.

\[
(E_0, B_0, \beta_0, \emptyset, \top)
\]

(1) \Rightarrow \text{SIMP} \ (E_0, B_0 \setminus \{x \geq 0\}, \beta_0, \{x \geq 0\}, \text{IV})

(2) \Rightarrow \text{SIMP} \ (E_0, B_1, \beta_0, \{x \geq 0\}, \top)

(3) \Rightarrow \text{SIMP} \ (E_0, \{z_2 \leq -2\}, \beta_0, \{x \geq 0, z_1 \geq 1\}, \text{IV})

(4) \Rightarrow \text{SIMP} \ (E_0, \{z_2 \leq -2\}, \beta_0, \{x \geq 0, z_1 \geq 1\}, \text{DV})

Now the bound \(z_1 \geq 1\) is clearly not satisfied by \(\beta_0\), so in order to fix it rule \(\text{FixDepVar} \geq\) is applied. In order to increase \(z_1\) with respect to \(z_1 \approx 2y + x\) either \(y\) or \(x\) need to be increased. Variable \(y\), is not contained in \(S_4\) and \(x\) is only bound from below, so both variables can be selected for pivoting. Here I select \(x\), resulting in the new equational system \(E_5 = \{x \approx -2y + z_1, z_2 \approx 3y - z_1\}\) and assignment \(\beta_5 = \{z_1 \mapsto 1, y \mapsto 0, x \mapsto 1, z_2 \mapsto -1\}\)

(5) \Rightarrow \text{SIMP} \ (E_5, \{z_2 \leq -2\}, \beta_5, \{x \geq 0, z_1 \geq 1\}, \text{DV})

(6) \Rightarrow \text{SIMP} \ (E_5, \{z_2 \leq -2\}, \beta_5, S_5, \top)

(7) \Rightarrow \text{SIMP} \ (E_5, \emptyset, \beta_5, S_5, \{z_2 \leq -2\}, IV)

(8) \Rightarrow \text{SIMP} \ (E_5, \emptyset, \beta_5, S_7, \text{DV})

Now the bound \(z_2 \leq -2\) is not satisfied by \(\beta_5\), because \(\beta_5(z_2) = -1\). Pivoting on \(z_2 \approx 3y - z_1\) on \(y\) yields \(E_9 = \{x \approx -\frac{2}{3}z_2 + \frac{1}{3}z_1, y \approx \frac{1}{3}(z_2 + z_1)\}\) and assignment \(\beta_9 = \{z_2 \mapsto -2, z_1 \mapsto 1, x \mapsto \frac{2}{3}, y \mapsto -\frac{1}{3}\}\).

(9) \Rightarrow \text{SIMP} \ (E_9, \emptyset, \beta_9, \{z_1 \geq 1, z_2 \leq -2, x \geq 0\}, \text{DV})

(10) \Rightarrow \text{SIMP} \ (E_9, \emptyset, \beta_9, S_9, \top)

Now \(B_{10}\) is empty and \(\beta_{10}\) satisfies all bounds and hence constitutes a solution to the initial problem.
The equational system and the respective bounds of Example 6.2.5 can be interpreted geometrically. Then a FixDepVar rule application corresponds to testing the intersection points between two of the three initial straights for a solution.

**Example 6.2.6** (Simplex Detecting Unsatisfiability). Consider the equational system $E = \{ x + 2y \geq 1, x - y \leq 3, x \geq 0, y \leq -1 \}$ which results after preprocessing in the sets $E_0 = \{ z_1 \approx x + 2y, z_2 \approx x - y \}$ and $B_0 = \{ z_1 \geq 1, z_2 \leq 3, x \geq 0, y \leq -1 \}$. Starting with an initial assignment $\beta_0$ that maps all variables to 0 and hence satisfies $E_0$, a Simplex run is as follows. Again, each line gets a number and I make references to the components of the simplex state of previous lines with respect to the line number.

1. $(E_0, B_0, \emptyset, \top)$
2. $(E_0, B_0 \setminus \{ x \geq 0 \}, \beta_0, \{ x \geq 0 \}, IV)$
3. $(E_0, B_1, \beta_0, \{ x \geq 0 \}, \top)$
4. $(E_0, B_1 \setminus \{ y \leq -1 \}, \beta_0, \{ x \geq 0, y \leq -1 \}, IV)$
5. $(E_0, B_3, \{ x \mapsto 0, y \mapsto -1, z_1 \mapsto -2, z_2 \mapsto 1 \}, S_3, IV)$
6. $(E_0, B_3 \setminus \{ z_1 \geq 1 \}, \beta_4, S_3, IV)$
7. $(E_0, B_3, \beta_4, S_3, \top)$
8. $(E_0, B_6, \beta_8, S_6, DV)$
9. $(E_0, B_6, \emptyset, \beta_8, S_6, \top)$
10. $(E_8, B_6, \beta_8, S_8, \top)$
11. $(E_8, \emptyset, \beta_8, S_{10}, DV)$
12. $(E_8, \emptyset, \emptyset, \emptyset, II)$

The bound $z_1 \geq 1$ is not satisfied by $\beta_7$ because $\beta_7(z_1) = -2$. Pivoting on $x$ in $z_1 \approx x + 2y$ yields $E_8 = \{ x \approx z_1 - 2y, z_2 \approx z_1 - 3y \}$ and $\beta_8 = \{ z_1 \mapsto 1, y \mapsto -1, x \mapsto 3, z_2 \mapsto 4 \}$.

**Lemma 6.2.7** (Simplex State Invariants). The following invariants hold for any state $(E_i; B_i; \beta_i; S_i; s_i)$ derived by $\Rightarrow_{SIMP}$ on a start state $(E_0; B_0; \emptyset; \top)$:

1. For every dependent variable there is exactly one equation in $E$ defining the variable.
2. Dependent variables do not occur on the right hand side of an equation.
3. $LRA(\beta) \models E_i$. 
4. for all in dependent variables \( x \) either \( \beta_i(x) = 0 \) or \( \beta_i(x) = c \) for some bound \( x \circ c \in S_i \),

5. for all assignments \( \alpha \) it holds \( \text{LRA}(\alpha) \models E_0 \) iff \( \text{LRA}(\alpha) \models E_i \)

**Proof.** 1. By induction on the length of a \( \Rightarrow_{\text{SIMP}} \) derivation. A consequence of the definition of \( \text{pivot} \).

3. By induction on the length of a \( \Rightarrow_{\text{SIMP}} \) derivation. A consequence of the definition of \( \text{upd} \).

4. By induction on the length of a \( \Rightarrow_{\text{SIMP}} \) derivation and a case analysis for all rules changing \( \beta_i \). Recall that initially \( \beta_0 \) maps all variables to 0.

5. The pivot operation is equivalence preserving, i.e., an assignment \( \alpha \) satisfies \( E \) iff it satisfies \( \text{pivot}(E, x, y) \) for a dependent variable \( x \) and an independent variable \( y \).

**Lemma 6.2.8** (Simplex Run Invariants). For any run of \( \Rightarrow_{\text{SIMP}} \) from start state \((E_0; B_0; \beta_0; \emptyset; \top) \Rightarrow_{\text{SIMP}} (E_1; B_1; \beta_1; S_1; s_1) \Rightarrow_{\text{SIMP}} \ldots:\)

1. the set \( \{\beta_0, \beta_1, \ldots\} \) is finite

2. if the sets of dependent and independent variables for two equational systems \( E_i, E_j \) coincide, then \( E_i = E_j \)

3. the set \( \{E_0, E_1, \ldots\} \) is finite

4. let \( S_i \) not contain contradictory bounds, then \((E_i; B_i; \beta_i; S_i; s_i) \Rightarrow_{\text{SIMP}}^{\text{FixIndepVar},*} \) is finite

**Proof.** 1. By induction on the length of a \( \Rightarrow_{\text{SIMP}} \) derivation. Variables are bound by the \( \beta_i \) to constants occurring \( B_0 \). This set is finite. Furthermore, the domain of each \( \beta_i \) is constant. Hence the set \( \{\beta_0, \beta_1, \ldots\} \) is finite.

2. By Lemma 6.2.7.1 and 2, for any dependent variable \( z \) there is exactly one equation \( z \approx a_1 x_1 + \ldots + a_n x_n \) in every \( E \). Now assume that dependent and independent variables for two equational systems \( E_i, E_j \) coincide but actually \( E_i \) and \( E_j \) differ in one equation \((z \approx a_1 x_1 + \ldots + a_n x_n) \in E_i \) and \((z \approx b_1 y_1 + \ldots + b_m y_m) \in E_j \). By Lemma 6.2.7.5 it must hold \( x_i = y_i \) and \( n = m \). It remains to show that the coefficients are identical. For \( n = 1 \) this is obvious. For \( n \geq 2 \) this follows again from Lemma 6.2.7.5 by the following two assignments \( \gamma, \gamma' \), assuming \( a_1 \neq b_1 \). The first assignment is defined by \( \gamma(z) = n \), and \( \gamma(x_k) = \frac{1}{a_k} \) for \( 1 \leq k \leq n \) and the second by \( \gamma'(z) = n - 2 \), \( \gamma'(x_k) = -\frac{1}{a_k} \) and \( \gamma'(x_k) = \frac{1}{a_k} \) for \( 2 \leq k \leq n \). Both assignments satisfy the defining equations for \( z \) and can be extended to satisfy \( E_i \) and \( E_j \). Then from \( \gamma \) we can conclude

\[
\frac{a_1}{a_1} > b_1 \frac{1}{a_1} \quad \text{iff} \quad a_2 \frac{1}{a_2} + \ldots + a_n \frac{1}{a_n} < b_2 \frac{1}{a_2} + \ldots + b_n \frac{1}{a_n}
\]

and from \( \gamma' \) accordingly

\[
\frac{a_1}{a_1} > b_1 \frac{1}{a_1} \quad \text{iff} \quad a_2 \frac{1}{a_2} + \ldots + a_n \frac{1}{a_n} > b_2 \frac{1}{a_2} + \ldots + b_n \frac{1}{a_n}
\]
a contradiction.
3. A consequence of 2.
4. The independent variables are in fact independent from each other. Thus any bound on an independent can be eventually satisfied by rule FixIndepVar.

**Corollary 6.2.9** (Infinite Runs Contain a Cycle). Let \((E_0; B_0; \beta_0; \emptyset; \top) \Rightarrow \text{SIMP} (E_1; B_1; \beta_1; S_1; s_1) \Rightarrow \text{SIMP} \ldots\) be an infinite run. Then there are two states \((E_i; B_i; \beta_i; S_i; s_i), (E_k; B_k; \beta_k; S_k; s_k)\) such that \(i \neq k\) and \((E_i; B_i; \beta_i; S_i; s_i) = (E_k; B_k; \beta_k; S_k; s_k)\).

**Proof.** The initial sets are all finite. No rule adds a simple bound to any \(B_i\), they can only be moved to some \(S_i\) and stay there. So there are only finitely many such configurations \(B_i, S_i\) during a run. By Lemma 6.2.8.1 there are only finitely many different \(\beta_i\). By Lemma 6.2.8.3 there are only finitely many different \(E_i\). In sum, any infinite run must contain two identical states, a cycle.

**Definition 6.2.10** (Reasonable Strategy). A reasonable strategy prefers Fail-Bounds over EstablishBounds and the FixDepVar rules select minimal variables \(x, y\) in the ordering \(\prec\).

**Theorem 6.2.11** (Simplex Soundness, Completeness & Termination). Given a reasonable strategy and initial set \(N\) of inequations and its separation into \(E\) and \(B:\)

1. \(\Rightarrow \text{SIMP}\) terminates on \((E_0; B_0; \beta_0; \emptyset; \top)\)
2. if \((E; B; \beta_0; \emptyset; \top) \Rightarrow^* \text{SIMP} (E'; B'; \beta; S; \bot)\) then \(N\) has no solution
3. if \((E; B; \beta_0; \emptyset; \top) \Rightarrow^* \text{SIMP} (E'; \emptyset; \beta; B; \top)\) and \((E; \emptyset; \beta; B; \top)\) is a normal form, then \(\text{LRA}(\beta) \models N\)
4. all final states \((E; B; \beta; S; s)\) match either 2. or 3.

**Proof.**
1. (Idea) An infinite run must contain a cycle due to Corollary 6.2.9. Runs always selecting minimal variables for the FixDepVar rules cannot contain cycles.
2. (Sketch) The fail rules are correct, given Lemma 6.2.7.5.
3. By Lemma 6.2.7.5 and all initial bounds are satisfied by \(\beta\), because Ack-Bounds is the only rule generating \(\top\).
4. A state \((E; B; \beta; S; \text{IV})\) can always be rewritten to a state \((E; B; \beta'; S; \top)\) or \((E; B; \beta'; S; \text{DV})\). Any state \((E; B; \beta; S; \text{DV})\) is either rewritten to a final state \((E; B; \beta; S; \bot)\) or again a state \((E'; B; \beta'; S; \text{DV})\). The rest follows from termination.

In case of strict bounds the idea is to introduce an infinitesimal small constant \(\delta > 0\) and to replace the strict bound by a non-strict one. So, for example, a bound \(x < 5\) is replaced by \(x \leq 5 - \delta\). Now \(\delta\) is treated symbolically through the