Theorem proving with equality

master thesis in computer science

by

Alexander Maringele

submitted to the Faculty of Mathematics, Computer
Science and Physics of the University of Innsbruck

in partial fulfillment of the requirements
for the degree of Master of Science

supervisor: Assoc. Prof. Dr. Georg Moser,
Institute of Computer Science

Innsbruck, 15 February 2018

Master Thesis

Yet another instantiation based first order theorem prover with equality

Alexander Maringele (8517725)
alexander.maringele@uibk.ac.at

15 February 2018

Supervisor: Assoc. Prof. Dr. Georg Moser

Eidesstattliche Erklärung

Ich erkläre hiermit an Eides statt durch meine eigenhändige Unterschrift, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe. Alle Stellen, die wörtlich oder inhaltlich den angegebenen Quellen entnommen wurden, sind als solche kenntlich gemacht.
Die vorliegende Arbeit wurde bisher in gleicher oder ähnlicher Form noch nicht als Magister-/Master-/Diplomarbeit/Dissertation eingereicht.

Datum

Unterschrift

Abstract

Instantiation-based . . .

Acknowledgments

Thanks.!

Contents

1 Introduction

2 Preliminaries

In this thesis we assume the reader's familiarity with propositional and first order logic [10], term rewriting (TRW [2]), decision procedures (DP [11]), and satisfiability checking modulo theories (SMT [3]). Nevertheless — for clarity — we state basic notions and definitions of first order logic with equality in Section 2.1, introduce basic concepts of first order semantics in Section 2.2, and describe basic term rewriting terminology in Section 2.3. These notions and notations largely follow the lecture notes to term rewriting and automated reasoning [12, 14].

2.1 Syntax

In this section we introduce the syntax of arbitrary first order formulae (`FOF`), prenex normal form (`PNF`), and clausal normal form (`CNF`).

Definition 2.1. A first order *signature* with equality $\mathcal{F} = \mathcal{F}_{\mathsf{f}} \mathbin{\dot\cup} \mathcal{F}_{\mathsf{P}} \mathbin{\dot\cup} \{\approx\}$ is the disjoint union of a set of *function symbols* $\mathcal{F}_{\mathsf{f}}$, a set of *predicate symbols* $\mathcal{F}_{\mathsf{P}}$, and one distinct equality symbol. The *arity* of a symbol determines the number of its arguments in a first order expression. With $\mathcal{F}^{(n)} = \{f \in \mathcal{F} \mid \text{arity}(f) = n\}$ we denote symbols with arity n.

Remark. We use $\approx$ as equality symbol in our signatures to emphasize that at this point it is just a highlighted symbol without "meaning". On the other hand we use $=$ to express "identity" of objects like formulae or sets without actually defining how this identity can be determined.

Definition 2.2. We build the set of (first order) *terms* $\mathcal{T}_{\mathsf{f}} = \mathcal{T}(\mathcal{F}_{\mathsf{f}}, \mathcal{V})$ from function symbols and a countable set of *variables* $\mathcal{V}$ disjoint from $\mathcal{F}$. Every variable $x \in \mathcal{V}$ is a term, every *constant* $\mathsf{c} \in {\mathcal{F}_{\mathsf{f}}}^{(0)}$ is a term, and every expression $\mathsf{f}(t_1, \ldots, t_n)$ is a term for $n > 0$, function symbol $\mathsf{f} \in \mathcal{F}_{\mathsf{f}}^{(n)}$, and arbitrary terms $t_1, \ldots, t_n$.

Definition 2.3. We define the set of variables of a first order term t as follows:

$$\mathcal{V}ars(t) = \begin{cases} \{x\} & \text{if } t = x \in \mathcal{V} \\ \bigcup_{i=1}^{n} \mathcal{V}ars(t_i) & \text{if } t = \mathsf{f}(t_1, \ldots t_n) \end{cases}$$

A *ground* term t' does not contain any variables, i.e. $\mathcal{V}ars(t') = \emptyset$.

Definition 2.4. For an unary function symbol $\mathsf{g} \in \mathcal{F}_{\mathsf{f}}^{(1)}$, a natural number $i \in \mathbb{N}$, and an arbitrary term $t \in \mathcal{T}_{\mathsf{f}}$ we introduce the notation $\mathsf{g}^i(t)$ defined as follows:

$$\mathsf{g}^0(t) := t \qquad \mathsf{g}^{i+1}(t) := \mathsf{g}(\mathsf{g}^i(t))$$

Definition 2.5. We build the set of (first order) *predicates* $\mathcal{P}(\mathcal{F}_{\mathsf{P}}, \mathcal{T}_{\mathsf{f}})$ from predicate symbols and terms. Every proposition $\mathsf{p} \in \mathcal{F}_{\mathsf{P}}^{(0)}$ is a predicate, and every expression $\mathsf{P}(t_1, \ldots, t_n)$ is a predicate for $n > 0$, predicate symbol $\mathsf{P} \in \mathcal{F}_{\mathsf{P}}^{(n)}$ and arbitrary terms $t_1, \ldots, t_n$. We build the set of (first order) *equations* $\mathcal{E}(\approx, \mathcal{T}_{\mathsf{f}})$ from the equality symbol and terms. Every pair $s \approx t$ is an equation for arbitrary terms s and t. The set of atomic formulas (or *atoms* for short) is the (distinct) union of predicates and equations.

2.1.1 Formulae and Normal forms

Definition 2.6 (`FOF`)**.** The atoms in Definition 2.5 are *first order formulae*. The universal quantification $(\forall x F)$ and the existential quantification $(\exists x F)$ of a first order formula are (quantified) first order formulae with *bound* variable $x \in \mathcal{V}$. The negation $(\neg F)$ of a first order formula is a (composite) first order formula. Further, the disjunction $(F \vee F')$, the conjunction $(F \wedge F')$, and the implication $(F \rightarrow F')$ of two first order formulae are (composite) first order formulae.

Remark. The Symbol → for implication is not to be confused with the equational symbol for rewrite rules in Section 2.3.

Definition 2.7. We define the set of *free* variables and the set of *bound* variables of a first order formula F as follows:

$$\mathcal{F}vars(F) = \begin{cases} \bigcup_{i=1}^{n} \mathcal{V}ars(t_i) & \text{if } F = \mathsf{P}(t_1, \ldots, t_n) \text{ or } t_1 \approx t_{n=2} \\ \mathcal{F}vars(G) & \text{if } F = \neg G \\ \mathcal{F}vars(G) \cup \mathcal{F}vars(H) & \text{if } F \in \{\, G \wedge H, G \vee H, G \rightarrow H \,\} \\ \mathcal{F}vars(G) \setminus \{x\} & \text{if } F \in \{\, \forall x\, G, \exists x\, G \,\} \end{cases}$$

$$\mathcal{B}vars(F) = \begin{cases} \emptyset & \text{if } F = \mathsf{P}(t_1, \ldots, t_n) \text{ or } t_1 \approx t_{n=2} \\ \mathcal{B}vars(G) & \text{if } F = \neg G \\ \mathcal{B}vars(G) \cup \mathcal{B}vars(H) & \text{if } F \in \{\, G \wedge H, G \vee H, G \rightarrow H \,\} \\ \{x\} \cup \mathcal{B}vars(G) & \text{if } F \in \{\, \forall x\, G, \exists x\, G \,\} \end{cases}$$

Example 2.8. With formula $F = (\forall x (x \approx y)) \vee (\exists y \mathsf{P}(x, y))$ we have $\mathcal{F}vars(F) = \mathcal{B}vars(F)$, which we would like to avoid: Formulae $F' = (\forall x (x \approx y')) \vee (\exists y \mathsf{P}(x', y))$ is equivalent to F (see Section 2.2 on page 8) and we get $\mathcal{F}vars(F') \cap \mathcal{B}vars(F') = \emptyset$.

We often will write formulae or sentences without stating the signature. The reader can easily deduce the underlying *implicit* signature with arities and the set of variables by applying the definitions of the syntax for first order formulae. We follow the convention to use x, y, z for variables and $\mathsf{a}, \mathsf{b}, \mathsf{c}$ for constant function symbols (which avoids ambiguity in the presence of free variables). For easier readability we will use uppercase predicate symbols and lowercase function symbols. We may denote $\mathcal{F}(F)$ for the implicit signature of an formula F.

Definition 2.9. A first order formula is closed, i.e. a first order *sentence*, if it does not contain free variables — all occurring variables are bound. Additionally we assume for any sentence that each variable is bound exactly once. Additionally the variable occurs as free variable in the subformula of the quantified subformula where it was bound:

$$
\begin{aligned}
\mathcal{B}\mathit{vars}(G * H) &= \mathcal{B}\mathit{vars}(G) \mathbin{\dot\cup} \mathcal{B}\mathit{vars}(H) && * \in \{\wedge, \vee, \rightarrow\} \\
\mathcal{B}\mathit{vars}(\exists\!\!\!\forall x F) &= \{x\} \mathbin{\dot\cup} \mathcal{B}\mathit{vars}(F),\ x \in \mathcal{F}\mathit{vars}(F) && \exists\!\!\!\forall \in \{\forall, \exists\}
\end{aligned}
$$

Example 2.10.

$$
\begin{array}{rcrc}
\forall x(\mathsf{P}(x) \vee \forall x \mathsf{Q}(x)) & ✗ & \forall x(\mathsf{P}(x) \vee \forall y \mathsf{Q}(y)) & ✓ \\
(\forall x \mathsf{P}(x)) \vee (\forall x \mathsf{Q}(x)) & ✗ & (\forall x \mathsf{P}(x)) \vee (\forall y \mathsf{Q}(y)) & ✓
\end{array}
$$

Definition 2.11 (`PNF`)**.** A first order sentence $F = \exists\!\!\!\forall_1 x_1 \ldots \exists\!\!\!\forall_n x_n\, G$ with n quantifiers $\exists\!\!\!\forall_i \in \{\exists, \forall\}$, n bound and distinct variables x_i, and quantifier free subformula G with a matching set of free variables, i.e. $\mathcal{F}\mathit{vars}(G) = \mathcal{B}\mathit{vars}(F)$, is in *prenex normal form*.

Definition 2.12 (`CNF`)**.** A (first order) *literal* L is either an atom A or the negation $(\neg A)$ of an atom. We usually abbreviate $\neg(s \approx t)$ with $s \not\approx t$. The *complement* L^c of an atom (positive literal) is the negation of the atom. The complement of a negated atom (negative literal) is the atom itself. A (first order) *clause* $\mathcal{C} = L_1 \vee \ldots \vee L_n$ is a possible empty multiset of literals. A finite *set of clauses* $S = \{\mathcal{C}_1, \ldots, \mathcal{C}_n\}$ is in *clausal normal form*.

Remark. First order terms, atoms, literals, clauses, and formulae are first order expressions. We call a first order expression without variables *ground*, i.e. we build our ground expressions over an empty set of variables. Implicitly ground first order formulae do not contain quantifiers.

2.1.2 Substitution

Definition 2.13. A *substitution* σ is a mapping from variables $x \in \mathcal{V}$ to terms in $\mathcal{T}(\mathcal{F}_\mathsf{f}, \mathcal{V})$ where *domain* $\mathrm{dom}(\sigma) = \{x \in \mathcal{V} \mid \sigma(x) \neq x\}$ and image $\mathrm{img}(\sigma) = \{\sigma(x) \mid x \in V, \sigma(x) \neq x\}$ are finite. We write substitutions as bindings, e.g. $\sigma = \{x_1 \mapsto s_1, \ldots, x_n \mapsto s_n\}$ where $\mathrm{dom}(\sigma) = \{x_1, \ldots, x_n\}$ and $\sigma(x_i) = s_i$. A *variable substitution* is a mapping from $\mathcal{V}$ to $\mathcal{V} \subseteq \mathcal{T}(\mathcal{F}_\mathsf{f}, \mathcal{V})$. A *renaming* is a bijective variable substitution. A *proper instantiator* is a substitution that is not a variable substitution (at least one variable is mapped to a non-variable term).

Definition 2.14. We define the instance $t\sigma$ respectively the application of a substitution σ to a literal or term t as follows

$$
t\sigma = \begin{cases}
s_i & \text{if } t = x_i \in \mathrm{dom}(\sigma), \sigma(x_i) = s_i \\
y & \text{if } t = y \in \mathcal{V} \setminus \mathrm{dom}(\sigma) \\
f(t_1\sigma, \ldots, t_n\sigma) & \text{if } t = f(t_1, \ldots, t_n) \text{ where } f \in \mathcal{F}^{(n)} \\
\neg(A\sigma) & \text{if } t = \neg A, \text{ where } A \text{ is an atom}
\end{cases}
$$

Further we define the instance of a clause as the multiset of the instances of its literals.

Definition 2.15. We can easily extend our definition to composite first order formulae, but the cases of quantified formulae need more consideration. So we only partially define $F\sigma$ for first order formulae F and substitution σ as follows (if $G\sigma$ and $H\sigma$ are defined in the respective cases).

$$F\sigma = \begin{cases} \neg(G\sigma) & \text{if } F = \neg G \\ (G\sigma) * (H\sigma) & \text{if } F = G * H, * \in \{\wedge, \vee, \rightarrow\} \\ \exists\!\!\!\forall x(G\sigma) & \text{if } F = \exists\!\!\!\forall xG, \exists\!\!\!\forall \in \{\forall, \exists\}, x \notin \operatorname{dom}(\sigma) \end{cases}$$

Definition 2.16. A clause $\mathcal{C}$ *strictly subsumes* a clause $\mathcal{D}$ if their exists a substitution θ such that $\mathcal{C}\theta \subsetneq \mathcal{D}$, e.g. when clause $\mathcal{D} = \mathcal{C}\theta \vee \mathcal{D}'$ is a weakened instance of clause $\mathcal{C}$.

Definition 2.17. We define the *composition* of two substitutions σ and τ as follows

$$\sigma\tau = \{x_i \mapsto s_i\tau \mid x_i \in \operatorname{dom}(\sigma)\} \cup \{y_i \mapsto t_i \mid y_i \in \operatorname{dom}(\tau) \setminus \operatorname{dom}(\sigma)\}.$$

Lemma 2.18. *With the definitions in 2.13 and 2.17 the equation* $(t\sigma)\tau = t(\sigma\tau)$ *holds for term, atoms, and literals.*

Proof. Assume σ and τ are substitutions. Then we use induction on the structure of the expression t that the equation $(t\sigma)\tau = t(\tau\sigma)$ holds in all possible cases.

- (base case) Let $t = x_i \in \operatorname{dom}(\sigma)$ then $((x_i)\sigma)\tau \overset{\text{def}}{=} s_i\tau \overset{\text{def}}{=} x_i(\sigma\tau)$ holds.
- (base case) Let $t = y \notin \operatorname{dom}(\sigma)$ then $(y\sigma)\tau \overset{\text{def}}{=} y\tau \overset{\text{def}}{=} y(\sigma\tau)$ holds.
- (step case) Let $t = f(t_1, \ldots, t_n)$ then $((f(t_1, \ldots, t_n))\sigma)\tau \overset{\text{def}}{=} (f(t_1\sigma, \ldots, t_n\sigma))\tau \overset{\text{def}}{=} f((t_1\sigma)\tau, \ldots, (t_n\sigma)\tau) \overset{\text{IH}}{=} f(t_1(\sigma\tau), \ldots, t_n(\sigma\tau)) \overset{\text{def}}{=} (f(t_1, \ldots, t_n))(\sigma\tau)$ holds.
- (step case) Let $t = \neg A$ then $((\neg A)\sigma)\tau \overset{\text{def}}{=} (\neg(A\sigma))\tau \overset{\text{def}}{=} \neg((A\sigma)\tau) \overset{\text{IH}}{=} \neg(A(\sigma\tau)) \overset{\text{def}}{=} (\neg A)(\sigma\tau)$ holds.

□

Definition 2.19. Two first order expressions t, u are *unifiable* if there exists a *unifier*, i.e. a substitution σ such that $t\sigma = u\sigma$. The *most general unifier* $\operatorname{mgu}(t, u)$ is a unifier such that for every other unifier σ' there exists a substitution τ where $\sigma' = \sigma\tau$. Two literals are variants if their most general unifier is a renaming. Two literals are *clashing* when the first literal and the complement of the second literal are unifiable, i.e. literals L' and L are clashing if $\operatorname{mgu}(L', L^c)$ exists.

Remark. The unification of quantified formulae remains undefined in this thesis.

Example 2.20. Literals $\mathsf{P}(x)$ and $\neg\mathsf{P}(\mathsf{f}(\mathsf{a}, y))$ are clashing by unifier $\{x \mapsto \mathsf{f}(\mathsf{a}, y)\}$.

Definition 2.21. A *position* is a finite sequence of positive integers. The root position is the empty sequence ϵ. The position pq is obtained by concatenation of positions p and q. A position p is *above* a position q if p is a prefix of q, i.e. there exists a unique position r such that $pr = q$, we write $p \leq q$ and $q \backslash r = p$. We write $p < q$ if p is a proper prefix of q, i.e. $p \leq q$ but $p \neq q$. We define $\mathrm{head}(iq) = i$ and $\mathrm{tail}(iq) = q$ for $i \in \mathbb{N}$, $q \in \mathbb{N}^*$, further $\mathrm{length}(\epsilon) = 0$, $\mathrm{length}(iq) = 1 + \mathrm{length}(q)$. Two positions $p \parallel q$ are parallel if none is above the other, i.e. for any common prefix r both remaining tails $p \backslash r$ and $q \backslash r$ are different and not root positions. A position p is left of position q if $\mathrm{head}(p \backslash r) < \mathrm{head}(p \backslash r)$ for maximal common prefix r.

Definition 2.22. We define the set of *positions* in an atom or a term recursively,

$$\mathcal{P}os(t) = \begin{cases} \{\epsilon\} & \text{if } t = x \in \mathcal{V} \\ \{\epsilon\} \cup \bigcup_{i=1}^{n} \{iq \mid q \in \mathcal{P}os(t_i)\} & \text{if } t = \mathsf{f}(t_1, \ldots, t_n), \mathsf{f} \in \mathcal{F}_{\mathsf{f}}^{(n)} \\ \{\epsilon\} \cup \bigcup_{i=1}^{n} \{iq \mid q \in \mathcal{P}os(t_i)\} & \text{if } t = \mathsf{P}(t_1, \ldots, t_n), \mathsf{P} \in \mathcal{F}_{\mathsf{P}}^{(n)} \text{ or } t = t_1 \approx t_2 \end{cases}$$

the set of *term positions* in an atom or a term,

$$t\text{-}\mathcal{P}os(t) = \begin{cases} \mathcal{P}os(t) & \text{if } t \text{ is a term} \\ \mathcal{P}os(t) \setminus \{\epsilon\} & \text{if } t \text{ is an atom} \end{cases}$$

the extraction of a subterm at a term position $p \in t\text{-}\mathcal{P}os(t)$ from an atom or a term,

$$t|_p = \begin{cases} t & \text{if } p = \epsilon, (t \text{ is a term}) \\ t_i|_q & \text{if } t = f(t_1, \ldots, t_n), p = iq, f \in \mathcal{F}^{(n)} \end{cases}$$

and the insertion of a term s at a term position $p \in t\text{-}\mathcal{P}os(t)$ into an atom or a term by replacing the subterm at that term position.

$$t[s]_p = \begin{cases} s & \text{if } p = \epsilon, (t \text{ is a term}) \\ f(t_1, \ldots, t_i[s]_q, \ldots, t_n) & \text{if } t = f(t_1, \ldots, t_n), p = iq, f \in \mathcal{F}^{(n)}, 0 < i \leq n \end{cases}$$

We may write $t[s]$ if s is a subterm of t (at some term position $p \in t\text{-}\mathcal{P}os(t)$, such that $t|_p = s$). With a follow up statement $t[s']$ in the same scope we express the replacement of subterm s with term s' in t, i.e. the application of $t[s']_p$.

2.1.3 Provability

In general a proof may be a finite sequence of proof steps from none or some premises via intermediate statements to a final, the then proven statement. A formal proof system or logical calculus describes admissible basic proof steps in the underlying logic of the statements, in our case first order logic. A formal proof comprises only proof steps confirmed by rules of the applied logical calculus.

Definition 2.23 ([10])**.** We recall the rules of *natural deduction* for connectives in Table 2.1, for equality in Table 2.2, and for quantifiers in Table 2.3. Natural deduction provides a logical calculus, i.e. a formal proof system for first order logic. The formulae F and G in these rules are sentences, the bound variable in $\forall x F'$ occurs free in F', and terms s and t are ground.

$$\frac{F \quad G}{F \wedge G}\,(\wedge i) \qquad \frac{F \wedge G}{G}\,(\wedge e_1) \qquad \frac{F \wedge G}{F}\,(\wedge e_2) \qquad \frac{F}{\neg\neg F}\,(\neg\neg i) \qquad \frac{\neg\neg F}{F}\,(\neg\neg e)$$

$$\frac{\bot}{F}\,(\bot e) \qquad \frac{F \quad \neg F}{\bot}\,(\neg e) \qquad \frac{}{F \vee \neg F}\,\text{LEM} \qquad \frac{F}{F \vee G}\,(\vee i_1) \qquad \frac{G}{F \vee G}\,(\vee i_2)$$

$$\frac{\boxed{\begin{array}{c} F \\ \vdots \\ \bot \end{array}}}{\neg F}\,(\neg i) \qquad \frac{\boxed{\begin{array}{c} \neg F \\ \vdots \\ \bot \end{array}}}{F}\,\text{PBC} \qquad \frac{\boxed{\begin{array}{c} F \\ \vdots \\ G \end{array}}}{F \to G}\,(\to i) \qquad \frac{F \vee G \quad \boxed{\begin{array}{c} F \\ \vdots \\ H \end{array}} \quad \boxed{\begin{array}{c} G \\ \vdots \\ H \end{array}}}{H}\,(\vee e)$$

$$\frac{F \quad F \to G}{G}\,\begin{array}{l}\text{modus}\\ \text{ponens}\end{array} \qquad \frac{F \to G \quad \neg G}{\neg F}\,\begin{array}{l}\text{modus}\\ \text{tollens}\end{array}$$

Table 2.1: Natural Deduction Rules for Connectives

$$\frac{s = t \quad F'\{x \mapsto s\}}{F'\{x \mapsto t\}}\,(=e) \qquad \frac{}{t = t}\,(=i)$$

Table 2.2: Natural Deduction Rules for Equality

Definition 2.24. A sentence in first order logic is provable if their exists a proof in a formal proof system for first order logic, e.g. natural deduction. We write $F_1, \ldots, F_n \vdash G$ when we can prove G from premises $F_1, \ldots, F_n$.

A natural deduction proof starts with a (possible empty) set of sentences — the premises — and infer other sentences — the conclusions — by applying the syntactic proof inference rules. A box must be opened for each assumption, e.g. a term or a sentence. Closing the box discards the assumption and all its conclusions within the box(es), but may introduce a derived sentence outside the box(es). Then $F_1, \ldots, F_n \vdash H$ claims that H is in the transitive closure of inferable formulae from $\{F_1, \ldots, F_n\}$ outside of any box.

Example 2.25. We show $\forall x(\mathsf{P}(x) \wedge \neg\mathsf{Q}(x)) \vdash \forall x(\neg\mathsf{Q}(x) \wedge \mathsf{P}(x))$ with natural deduction. We note our premise (1), we open a box and assume an arbitrary constant (2), we create a ground instance of our premise with quantifier elimination and the constant (3), we extract the literals with both variants of conjunction elimination (4, 5), we introduce a conjunction of the ground literals (6), and close the box to introduce the universal

$$\frac{\forall x F'}{F'\{x \to t\}}\ (\forall e) \qquad\qquad \frac{F'\{x \mapsto t\}}{\exists x F'}\ (\exists i)$$

$$\frac{\boxed{\begin{array}{l} t \\ \qquad\vdots \\ \quad F'\{x \mapsto t\} \end{array}}}{\forall x F'}\ (\forall i) \qquad\qquad \frac{\exists x F' \quad \boxed{\begin{array}{c} t \quad F'\{x \mapsto t\} \\ \vdots \\ H \end{array}}}{H}\ (\exists e)$$

Table 2.3: Natural Deduction Rules for Quantifiers

quantified conjunction (7).

1	$\forall x(\mathsf{P}(x) \land \neg\mathsf{Q}(x))$	`premise`
2	c	
3	$\mathsf{P}(\mathsf{c}) \land \neg\mathsf{Q}(\mathsf{c})$	$1 : \forall e$
4	$\neg\mathsf{Q}(\mathsf{c})$	$3 : \land e_1$
5	$\mathsf{P}(\mathsf{c})$	$3 : \land e_2$
6	$\neg\mathsf{Q}(\mathsf{c}) \land \mathsf{P}(\mathsf{c})$	$4 + 5 : \land i$
7	$\forall x(\neg\mathsf{Q}(x) \land \mathsf{P}(x))$	$2 - 6 : \forall i$

2.2 Semantics

In this section we recall some basic aspects and definitions of semantics in first order logic. We state satisfiability and validity of arbitrary first order formulae or sets of clauses.

2.2.1 Models

Definition 2.26. An *interpretation* $\mathcal{I}$ over a signature $\mathcal{F}$ consists of a non-empty set A — the *universe* or *domain*, definitions of mappings $\mathsf{f}_\mathcal{I} : A^n \to A$ for every function symbol $\mathsf{f} \in \mathcal{F}_\mathsf{f}$, and definitions of (possibly empty) n-ary relations $\mathsf{P}_\mathcal{I} \subseteq A^n$ for every predicate symbol $\mathsf{P} \in \mathcal{F}_\mathsf{P}$ and the definition of a binary relation $\approx_\mathcal{I} \subseteq A^2$ for the equality symbol. A (variable) *assignment* is a mapping from variables to elements of the domain. We define the *evaluation* $\alpha_\mathcal{I}$ of a term t for assignment α and interpretation $\mathcal{I}$:

$$\alpha_\mathcal{I}(t) = \begin{cases} \alpha_\mathcal{I}(x) & \text{if } t = x \in \mathcal{V} \\ \mathsf{c}_\mathcal{I} & \text{if } t = \mathsf{c} \in \mathcal{F}_\mathsf{f}^{(0)} \\ \mathsf{f}_\mathcal{I}(\alpha_\mathcal{I}(t_1), \ldots, \alpha_\mathcal{I}(t_n)) & \text{if } t = \mathsf{f}(t_1, \ldots, t_n), \mathsf{f} \in \mathcal{F}_\mathsf{f}^{(n>0)}, t_i \in \mathcal{T}_\mathsf{f} \end{cases}$$

Remark. The evaluation of ground terms does not depend on variable assignments.

Definition 2.27. A predicate $\mathsf{P}(t_1, \ldots, t_n)$ *holds* for an assignment $\alpha_\mathcal{I}$ if and only if the evaluation of its n-tuple $(\alpha_\mathcal{I}(t_1), \ldots, \alpha_\mathcal{I}(t_n))$ is an element of the relation $\mathsf{P}_\mathcal{I} \subseteq A^n$. Similar an equation $s \approx t$ holds if $\alpha_\mathcal{I}(s) \approx_\mathcal{I} \alpha_\mathcal{I}(t)$.

Definition 2.28 (Semantics of `FOF`)**.** A universally quantified sentence $\forall x F$ holds in an interpretation if its subformula F holds for all assignments for x. An existential quantified sentence $\exists x F$ holds if its subformula F holds for at least one assignment for x. For a given interpretation and predefined assignments for all occurring free variables a negation $\neg F$ holds if its subformula F does not hold, a disjunction $F \vee G$ holds if one or both of its subformulae F or G hold, a conjunction $F \wedge G$ holds, if both of its subformulae F and G hold, an implication $F \rightarrow G$ holds if its first subformula F does not hold or its second subformula F' holds (or both).

Remark. Usually we use precedences on connectives to omit parentheses and some heuristics to structure the formulae for readability without introducing semantic ambiguity.

Definition 2.29 (Semantics of `CNF`)**.** An atom holds in an interpretation if and only if it holds with all possible assignments. A literal holds if and only if its complement does not hold. A clause holds if at least one of its literals holds, hence the empty clause $\square$ does not hold in any interpretation. A set of clauses holds if and only if every clause in the set holds.

Definition 2.30. A *model* $\mathcal{M}$ for a set of clauses S (for a sentence F) is an interpretation that *satisfies* the set of clauses (the sentence), i.e. the set of clauses (the sentence) holds in that interpretation $\mathcal{M}$. We write $\mathcal{M} \models S$ or $\mathcal{M} \models F$.

A set of clauses (a sentence) is *satisfiable* if there exists at least one model for it. A set of clauses (a sentence) is *valid* if and only if every interpretation is a model.

Definition 2.31. The *Herbrand universe* for a first order signature $\mathcal{F}$ is the smallest set of terms that contains all $H_{i \geq 0}$ defined inductively as

$$
H_0 = \begin{cases} \{\, \mathsf{c} \mid \mathsf{c} \in \mathcal{F}_{\mathsf{f}}^{(0)} \,\} & \text{if } \mathcal{F}_{\mathsf{f}}^{(0)} \neq \emptyset \\ \{\, \mathsf{c}_0 \,\} & \text{if } \mathcal{F}_{\mathsf{f}}^{(0)} = \emptyset, \mathsf{c}_0 \notin \mathcal{F} \end{cases} \qquad H'_0 = H_0
$$

$$
H_{k+1} = \bigcup_{n>0} \{\, \mathsf{f}(t_1, \dots, t_n) \mid \mathsf{f} \in \mathcal{F}_{\mathsf{f}}^{(n)}, t_1, \dots, t_n \in H'_k \,\} \qquad H'_{k+1} = H_k \cup H'_{k+1}
$$

Definition 2.32. An *Herbrand interpretation* $\mathcal{H}$ is an interpretation where the domain is an Herbrand universe and the interpretation of each ground term $t_{\mathcal{H}} := t$ is the term itself.

2.2.2 Equivalence and Equisatisfiability

Definition 2.33. A (first order) sentence G is a semantic consequence of a set of sentences $\Gamma = \{F_1, \dots, F_n\}$ if G holds in all models for $F_1, \dots, F_n$. We write $\Gamma \models G$ and also say that Γ entails G. Two sentences $F \equiv G$ are equivalent if an only if the first sentence entails the second and vice versa.

Lemma 2.34. *We can easily see that not satisfiable $F \wedge \neg F$ entails every formula, that* *valid $F \lor \neg F$ is entailed by every sequence, further that*

$$
\begin{array}{ccc}
\Delta, F \models G & \textit{if and only if} & \Delta \models \neg F \lor G \\
F_1, \ldots, F_n \models G & \textit{if and only if} & F_1 \land \ldots \land F_n \models G \\
F \land G \equiv G \land F & F \lor G \equiv G \lor F & F \rightarrow G \equiv \neg F \lor \neg G
\end{array}
$$

by the definitions for semantics, entailment, and equivalence. □

Example 2.35. It is easy to see that $\mathsf{P}(\mathsf{a}) \not\equiv \mathsf{P}(\mathsf{b})$ with an interpretation where $\mathsf{P} = \{\mathsf{a}\}$. but there the same in "satisfiability".

Definition 2.36. A equivalence transformation transforms a sentence F to an equivalent sentence F'.

but $\mathsf{P}(\mathsf{f}(\mathsf{a})) \approxeq \mathsf{Q}(\mathsf{g}(\mathsf{b}))$

Example 2.37. We can relate sets of clauses to sentences

$$\{\, \mathsf{P}(x, \mathsf{f}(x)),\ \mathsf{Q}(y, \mathsf{a}) \,\} \approxeq \forall x \exists y (\mathsf{P}(x, y)) \land \exists a \forall y (\mathsf{Q}(y, a)).$$

2.2.3 Equality

In Definition 2.27 we have interpreted the equality symbol as binary relation without restrictions. This of course allows undesirable models as in the following Example 2.38. Hence we state useful definitions to deal with this situation and demonstrate their usage in an example.

Example 2.38. Any interpretation $\mathcal{I}$ with $\approx_{\mathcal{I}} = \emptyset$ satisfies $\mathsf{a} \not\approx \mathsf{a}$.

Definition 2.39. An *normal* interpretation defines $\approx_{\mathcal{I}}$ as identity on its domain, e.g. the equation of terms $s \approx_{\mathcal{I}} t$ holds if and only if any evaluation of its terms are equal $\alpha_{\mathcal{I}}(s) = \alpha_{\mathcal{I}}(t)$ for all assignments α. In other words a normal interpretation yields different elements for ground terms s' and t' if and only if $s' \not\approx_{\mathcal{I}} t'$.

Definition 2.40. A *term interpretation* $\mathcal{I}_t$ is an interpretation where the elements of its domain $A = \mathcal{T}(\mathcal{F}_{\mathsf{f}}, \emptyset)/_{\sim}$ are equivalence classes of ground terms and the interpretation of each ground term $t^{\mathcal{I}_t} := [t]_{\sim}$ is its equivalence class. A ground predicate $\mathsf{P}(t_1, \ldots, t_n)$ holds if $([t_1]_{\sim}, \ldots, [t_n]_{\sim}) \in \mathsf{P}^{\mathcal{I}_t} \subseteq A^n$.

Example 2.41. Consider the satisfiable set of clauses $S = \{\mathsf{f}(x) \approx x\}$. We easily find a Herbrand model $\mathcal{H}$ with predicate definition $\approx_{\mathcal{H}} = \{(\mathsf{f}^{i+1}(\mathsf{a})), \mathsf{f}^i(\mathsf{a}) \mid i \geq 0\}$. However $\mathcal{H}$ is not a normal model because obviously $\mathsf{f}(\mathsf{a}) \neq \mathsf{a}$ in its domain. Further on we easily find an normal model $\mathcal{M}$ with domain $\{\mathsf{c}\}$, function definition $\mathsf{f}_{\mathcal{M}}(\mathsf{c}) \mapsto \mathsf{c}$, and the relation $\approx_{\mathcal{M}} = \{(\mathsf{c}, \mathsf{c})\}$ coincides with identity in its domain. Certainly this model $\mathcal{M}$ is not an Herbrand model because the interpretation of ground term $\mathsf{f}(\mathsf{c})_{\mathcal{M}} = \mathsf{c}$ is not the ground term $\mathsf{f}(\mathsf{c})$ itself. On the other hand we easily construct a normal term model $\mathcal{M}_t$ with domain $\{[\mathsf{a}]_{\sim}\}$, a plain function definition $\mathsf{f}_{\mathcal{M}_t}([\mathsf{a}]_{\sim}) \mapsto [\mathsf{a}]_{\sim}$ with equivalence relation $\mathsf{a} \sim \mathsf{f}(\mathsf{a})$. Hence $\approx_{\mathcal{M}_t}$ agrees to equality in its domain of equivalence classes of ground terms.

2.2.4 Completeness of First Order Logic

Theorem 2.42 (Gödels Vollständigkeitssatz)**.** *Es gibt einen Kalkül der Prädikatenlogik erster Stufe derart, dass für jede Formelmenge Γ und für jede Formel φ gilt: φ folgt genau dann aus Γ, wenn φ im Kalkül aus Γ hergeleitet werden kann.*

Theorem 2.43. *Natural deduction is a complete calculus, i.e. a proof $\Gamma \vdash G$ exists, whenever $\Gamma \models G$.*

2.3 Term Rewriting and Orderings

Definition 2.44. A term rewrite signature $\mathcal{F}_\mathsf{f}$ is a set of function symbols with associated arities as in Definition 2.1. Terms, term variables, ground terms and unary function symbol notations are defined as in Definitions 2.2 to 2.4.

Definition 2.45. A *rewrite rule* is an equation of terms where the left-hand side is not a variable and the variables occuring in the right-hand side occur also in the left-hand side. A rewrite rule $\ell' \to r'$ is a *variant* of $\ell \to r$ if there is a variable renaming ϱ such that $(\ell \to r)\varrho := \ell\varrho \to r\varrho = l' \to r'$. A *term rewrite system* is a set of rewrite rules without variants. In a *ground* term rewrite system every term on every side in every rule is a ground term.

Although we use the the same symbol for implications $F \to G$ between first order formulae and rewrite rules $s \to t$ or rewrite steps $s' \to_\mathcal{R} t$ between first order terms, there will not arise any ambiguity for the reader about the role of the symbol.

Definition 2.46. We say $s \to_\mathcal{R} t$ is a *rewrite step* with respect to TRS $\mathcal{R}$ when there is a position $p \in \mathcal{P}os(s)$, a rewrite rule $\ell \to r \in \mathcal{R}$, and a substitution σ such that $s|_p = \ell\sigma$ and $s[r\sigma]_p = t$. The subterm $\ell\sigma$ is called *redex* and s rewrites to t by *contracting* $\ell\sigma$ to *contractum* $r\sigma$. We say a term s is *irreducible* or in *normal form* with respect to TRS $\mathcal{R}$ if there is no rewrite step $s \to_\mathcal{R} t$ for any term t. The set of normal forms $\mathsf{NF}(\mathcal{R})$ contains all irreducible terms of the TRS $\mathcal{R}$.

Definition 2.47. A term s can be rewritten to term t with notion $s \to_\mathcal{R}^* t$ if there exists at least one *rewrite sequence* $(a_1, \ldots, a_n)$ such that $s = a_1$, $a_n = t$, and $a_i \to_\mathcal{R} a_{i+1}$ are rewrite steps for $1 \leq i < n$. A TRS is *terminating* if there is no infinite rewrite sequence of terms.

Definition 2.48. A *rewrite relation* is a binary relation $\circledast$ on arbitrary terms s and t, which additionally is *closed under contexts* (whenever $s \circledast t$ then $u[s]_p \circledast u[t]_p$ for an arbitrary term u and any position $p \in \mathcal{P}os(u)$) and *closed under substitutions* (whenever $s \circledast t$ then $s\sigma \circledast t\sigma$ for an arbitrary substitution σ).

Lemma 2.49. *The relations $\to_\mathcal{R}^*$, $\to_\mathcal{R}^+$, $\downarrow_\mathcal{R}$, $\uparrow_\mathcal{R}$ are rewrite relations on every TRS $\mathcal{R}$.*

Definition 2.50. A proper (i.e. irreflexive and transitive) order on terms is called *rewrite order* if it is a rewrite relation. A *reduction order* is a well-founded rewrite order, i.e. there is no infinite sequence $(a_i)_{i\in\mathbb{N}}$ where $a_i \succ a_{i+1}$ for all i. A *simplification order* is a rewrite order with the *subterm property*, i.e. $u[t]_p \succ t$ for all terms u, t and positions $p \neq \epsilon$.

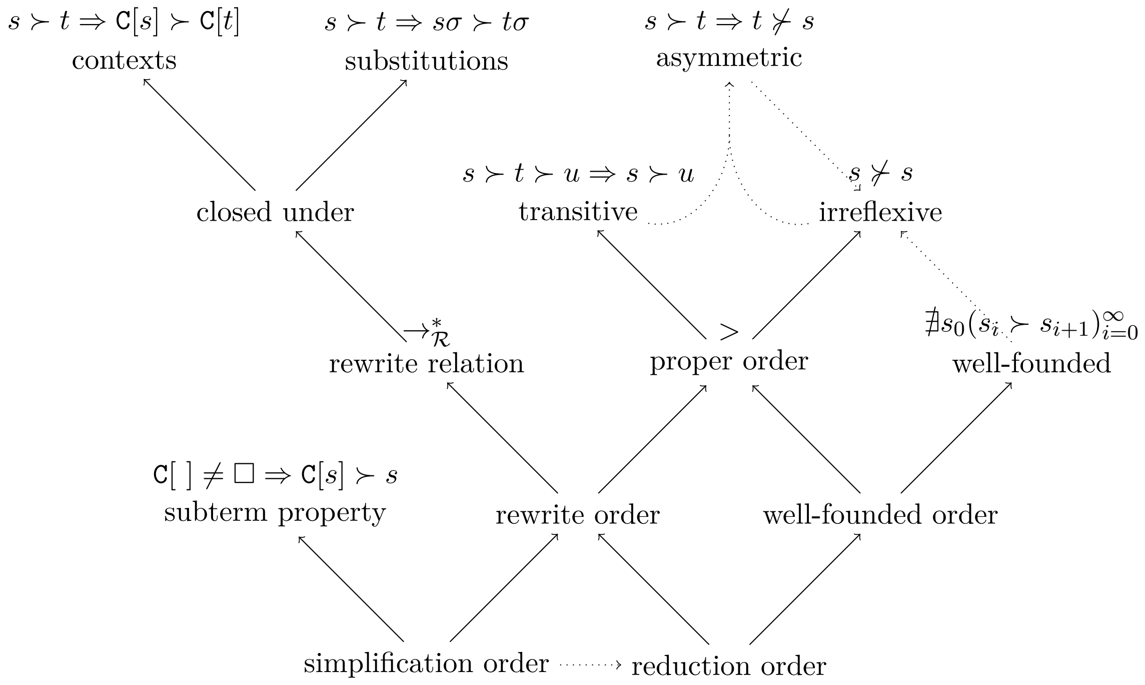

$s \succ t \Rightarrow \mathtt{C}[s] \succ \mathtt{C}[t]$
contexts
$s \succ t \Rightarrow s\sigma \succ t\sigma$
substitutions
$s \succ t \Rightarrow t \not\succ s$
asymmetric
closed under
$s \succ t \succ u \Rightarrow s \succ u$
transitive
$s \not\succ s$
irreflexive
$\rightarrow^*_{\mathcal{R}}$
rewrite relation
$>$
proper order
$\nexists s_0(s_i \succ s_{i+1})_{i=0}^{\infty}$
well-founded
$\mathtt{C}[\,] \neq \Box \Rightarrow \mathtt{C}[s] \succ s$
subterm property
rewrite order
well-founded order
simplification order
reduction order

Figure 2.1: Properties of relations on terms

Figure 2.3 summarizes the properties of relations on terms. The solid arrows mark definitions, e.g. a rewrite order is closed under contexts and substitutions (Definition 2.48); a simplification order is a rewrite order that respects the subterm property (Definition 2.50). The dotted arrows mark derived properties, e.g. every simplification order is a reduction order (Lemma 2.51); transitive and irreflexive relations are always asymmetric, etc.

Lemma 2.51. *Every simplification order is well-founded, hence it is a reduction order.*

Theorem 2.52. *A TRS $\mathcal{R}$ is terminating if and only if there exists a reduction order $\succ$ such that $l \succ r$ for every rewrite rule $l \rightarrow r \in \mathcal{R}$. We call $\mathcal{R}$ simply terminating if $\succ$ is a simplification order.*

2.3.1 Clause and literal orderings

Lemma 2.53. *Any ordering $\succ$ on a set C can be extended to an ordering on multisets over C as follows $N \succ M$ if $N \neq M$ and whenever there is $x \in C$ with $N(x) < M(x)$ then there is $y \succ x$ with $N(y) > M(y)$.*

An ordering $\succ$ on terms can be extended to orderings on literals and clauses.

For the following definition we assume $\succ_{\mathtt{gr}}$ as a a total, well-founded and monotone extension from a total simplification ordering on ground terms to ground clauses [15].

Definition 2.54. We define an order $\succ_{\mathtt{L}}$ on ground closures of literals as an arbitrary total well-founded extension of $\succ_{\mathtt{gr}}$ such that $L \cdot \sigma \succ_{\mathtt{L}} L' \cdot \sigma'$ whenever $L\sigma \succ_{\mathtt{gr}} L'\sigma'$.

We define an order $\succ_{\mathtt{C}}$ on ground closures as an arbitrary total well-founded extension of $\succ_{\mathtt{C}}'$ — an inherently well-founded order defined as extension of $\succ_{\mathtt{gr}}$ such that $C \cdot \tau \succ_{\mathtt{C}}' D \cdot \rho$ whenever $C\tau \succ_{\mathtt{gr}} D\rho$ or $C\theta = D$ for an proper instantiator θ.

Lemma 2.55. *A well-founded and total order on general ground terms always exists.*

Definition 2.56 (Order on literals)**.** We extend a well-founded and total order $\succ$ on general ground terms, i.e general atoms to a well-founded proper order $\succ_{\mathtt{L}}$ on literals such that for all atoms A and B with $A \succ B$ the relations $A \succ_{\mathtt{L}} B$, $\neg A \succ_{\mathtt{L}} \neg B$ and $\neg A \succ_{\mathtt{L}} A$ hold. A (non-ground) literal L is *(strictly) maximal* if there exists a ground substitution τ such for no other literal L' the relation $L'\tau \succ L\tau$ (strictly: $\succcurlyeq$) holds. We write $\succ_{gr}$ to suggest the existence of such a ground substitution τ.

3 Undecidability

A logical calculus, i.e. a formal (purely syntactical) proof system for an underlying logic, is *complete* if every (semantically) valid formula is (syntactically) provable from its premises by the calculus. In other words, every sentence that holds in all possible models for its premises is derivable from its premises by applying rules of the formal system only. Additional we expect a useful calculus to be *sound*, that is, every (syntactically) provable formula (semantically) holds in any model for its premises. Without premises a sentence has to be provable if and only if it holds in any interpretation.

We first state some completeness, undecidability, and other fundamental theorems about first order logic in Section 3.1. Then we enumerate decidable fragments of first order logic which can be described purely syntactically in Section 3.2 on the facing page. We conclude this chapter with a look at decidable first-order theories in Section 3.3, which are not necessarily contained in one of the syntactically describable and decidable fragments of first-order logic.

3.1 Theorems of First Order Logic

The most fundamental theorem about first order logic were introduced and proven in the first half of the 20th century.

Theorem 3.1 (Soundness). *The inference rules of natural deduction (Definition 2.23 on page 6) are sound.*

Proof. We prove the soundness of each inference rule by case distinction and the use of the semantic definition of validity. □

Theorem 3.2 (Completeness, Gödel 1929). *Natural deduction as a sound formal proof system for first order logic is complete.*

Theorem 3.3 (Undecidability, Church 1936, Turing 1937). *The satisfiability problem for first-order logic is undecidable.*

Theorem 3.4 (Trakhtenbort 1950, Craig 1950). *The satisfiability problem for first-order logic on* finite *structures (domains) is undecidable.*

Definition 3.5 (Finite model property). A logic has the finite model property if each non-theorem is falsified by some finite model.

Lemma 3.6 (Refutation). *By definition of the semantics of negation a formula is valid if and only if its negation is not satisfiable.*

Theorem 3.7 (Compactness, Gödel 1930, Maltsev 1936). *If every finite subset of a set of formulas S has a model then S has a model.*

Theorem 3.8 (Löwenheim Skolem, 1915, 1920). *If a set of formulas S has a model then S has a countable model.*

Theorem 3.9 (Herbrand, 1930). *Let S be a set of clauses without equality. Then the following statements are equivalent.*

- *S is satisfiable.*
- *S has a Herbrand model.*
- *Every finite subset of all ground instances of S has a Herbrand model.*

Corollary 3.10. *Let S be a set of clauses without equality. Then S is unsatisfiable if and only if there exists an unsatisfiable finite set of ground instances of S.*

Lemma 3.11. *With Skolemization and Tseytin transformation we can effectively transform a arbitrary first-order formula into an equisatisfiable set of clauses.*

3.2 Decidable Fragments of First Order Logic

This section presents purely syntactical defined fragments of first-order logic where satisfiability is decidable.[1]

Definition 3.12 ([4]). We describe classes of first-order formulae in PNF with triples

$$[\,\Pi, (p_1, p_2, \ldots), (f_1, f_2, \ldots)\,]_{(\approx)} \subseteq [\,all, all, all\,]_{\approx}$$

where $\Pi = \exists\!\!\!\forall_1 \ldots \exists\!\!\!\forall_n$, $\exists\!\!\!\forall_i \in \{\forall, \exists\}$ describes the structure of the quantifier prefix (without variables) of the formulae, the value p_i is the maximal number of predicate symbols with arity i, and the value f_i the maximal number of function symbols with arity i in the signature. The equality symbol is not counted as binary predicate symbol. Instead, the absence or presence of equality in the formulae is indicated by the absence or presence of a subscript $\approx$.

Example 3.13. The monadic predicate calculus includes formulae with arbitrary quantifier prefixes, arbitrary many unary predicate symbols, the equality symbol, but no function symbols.

$$[\,all, (\omega), (1)\,]_{\approx} \quad \supsetneq \quad [\,all, (\omega), (0)\,]_{\approx} \qquad \text{(Löwenheim 1925, Kalmár 1929)}$$

Example 3.14. The Ackermann prefix class contains formulae with arbitrary many existential quantifiers, but just one universal quantifier. It contains arbitrary many predicate symbols with arbitrary arities, the equality symbol, but no function symbols.

$$[\,\exists^*\forall\exists^*, all, (1)\,]_{\approx} \quad \supsetneq \quad [\,\exists^*\forall\exists^*, all, (0)\,]_{=} \qquad \text{(Ackermann 1928)}$$

[1] Definitions and compact overviews follow the presentation "Decidable fragments of first-order and fixed-point logic" by E. Grädel (`http://logic.rwth-aachen.de/~graedel/`).

Remark. One unary function symbol can be added to these fragments of first order logic above without loosing decidability (see Table 3.2).

$[\,\exists^*\forall^*, all, (0)\,]_=$	(Bernays, Schönfinkel 1928, Ramsey 1932)
$[\,\exists^*\forall^2\exists^*, all, (0)\,]$	(Gödel 1932, Kalmár 1933, Schütte 1934)
$[\,all, (\omega), (\omega)\,]$	(Löb 1967, Gurevich 1969)
$[\,\exists^*\forall\exists^*, all, all\,]$	(Gurevich 1973)
$[\,\exists^*, all, all\,]_=$	(Gurevich 1976)

Table 3.1: Decidable prefix classes with finite model property

$[\,all, (\omega), (1)\,]_=$	(Rabin 1969)
$[\,\exists^*\forall\exists^*, all, (1)\,]_=$	(Shelah 1977)

Table 3.2: Decidable prefix classes with infinity axioms.

Lemma 3.15. *Satisfiability is decidable [4] in all prefix classes from Tables 3.1 and 3.2. Each of theses classes is closed under conjunction with respect to satisfiability.*

3.3 Theories in First Order Logic

We follow definitions and examples in [13].

Definition 3.16 (Theory)**.** A *first-order theory* is a pair of a first-order signature and the possible infinite conjunction $\bigwedge_i A_i$ of first-order formulae, i.e. the axioms, over the theory's signature. A theory is *consistent* if the contradiction is not derivable. A theory is satisfiable if there exists a model for its axioms. A *theorem* is a sentence over the theory's signature, i.e. a closed formula, that holds in any model for the theory's axioms.

$$\bigwedge_i A_i \models \mathsf{theorem} \quad \text{or} \quad \bigwedge_i A_i \to \mathsf{theorem}$$

A theory is decidable if it is decidable whether an arbitrary sentence holds in the theory.

Example 3.17. A theory with axioms $\forall x\, \mathsf{P}(x)$ and $\exists x\, \neg\mathsf{P}(x)$ is neither consistent nor satisfiable.

Lemma 3.18. *A first order theory is consistent if and only if it is satisfiable.*

Remark. In refutational theorem proving we show the unsatisfiability of a negated sentence, i.e. a *conjecture*, in conjunction with the axioms to conclude that the conjecture is indeed a theorem.

$$\neg\left(\bigwedge_i A_i \rightarrow \mathsf{conj}\right) \equiv \neg\left(\neg\bigwedge_i A_i \vee \mathsf{conj}\right) \equiv \bigwedge_i A_i \wedge \neg\mathsf{conj}$$

3.3.1 Theory of equality

The following equivalence and congruence axioms form the theory of equality over a first order signature.

Definition 3.19 (Equivalence)**.** A binary relation $\approx$ over a domain is an equivalence relation if and only if the following axioms hold over the given domain.

$$\begin{aligned} &\forall x\ (x \approx x) && \text{reflexivity}\\ &\forall x \forall y\ (x \approx y \rightarrow y \approx x) && \text{symmetry}\\ &\forall x \forall y \forall z\ (x \approx y \wedge y \approx z \rightarrow x \approx z) && \text{transitivity}\end{aligned}$$

Definition 3.20 ($\vec{x}$-Notation)**.** Occasionally we may abbreviate a sequence of n variables by $\vec{x}$. Then we write $f(\vec{x})$ for first-order expression $f(x_1, \ldots, x_n)$ with n-ary function or predicate symbol f, a single equation $\vec{x} \approx \vec{y}$ for the conjunction of n equations $x_1 \approx y_1 \wedge \ldots \wedge x_n \approx y_n$, and $\forall\vec{x}$ for the sequence of quantified variables $\forall x_1 \ldots \forall x_n$.

Definition 3.21 (Congruence schemata)**.** An equivalence relation $\approx$ is a congruence relation if and only if the following formulae hold

$$\begin{aligned} &\forall\vec{x}\forall\vec{y}\ (\vec{x} \approx \vec{y} \rightarrow \mathsf{f}(\vec{x}) \approx \mathsf{f}(\vec{y})) && \text{for all } \mathsf{f} \in \mathcal{F}_{\mathsf{f}}^{(n)}\\ &\forall\vec{x}\forall\vec{y}\ (\vec{x} \approx \vec{y} \rightarrow (\mathsf{P}(\vec{x}) \rightarrow \mathsf{P}(\vec{y}))) && \text{for all } \mathsf{P} \in \mathcal{F}_{\mathsf{P}}^{(n)}\end{aligned}$$

Lemma 3.22. *The equivalence and congruence axioms of equality are provable with natural deduction (Definition 2.23 on page 6, Table 2.1 on page 7, Table 2.2 on page 7, and Table 2.3 on page 8).*

Proof. For brevity we skip the quantifier introductions (and handle variables like constants) for symmetry, transitivity, and congruence. Additionally we just show congruence for a unary function and a unary predicate symbol.

1	$\mathsf{c}_0 \quad \mathsf{c}_0 = \mathsf{c}_0$	$=i$
2	$\forall x\,(x = x)$	$\forall i, 1, \{x \mapsto \mathsf{c}_0\}$

1	$y = y$	$=i$
2	$x = y$	assume
3	$y \neq x$	assume
4	$y \neq y$	$=e, 2, 3$
5	$\bot$	$\neg e, 1, 4$
6	$y = x$	PBC, $3{-}5$
7	$x = y \to y = x$	$\to i, 1{-}5$

1	$\mathsf{f}(y) = \mathsf{f}(y)$	$=i$
2	$x = y$	assume
3	$\mathsf{f}(x) \neq \mathsf{f}(y)$	assume
4	$\mathsf{f}(y) \neq \mathsf{f}(y)$	$=e, 2, 3$
5	$\bot$	$\neg e, 1, 4$
6	$\mathsf{f}(x) = \mathsf{f}(y)$	PBC, $3{-}5$
7	$x = y \to \mathsf{f}(x) = \mathsf{f}(y)$	$\to i, 1{-}5$

1	$x = y \land y = z$	assume
2	$y = z$	$\land e_2$
3	$x = y$	$\land e_1, 2$
4	$x = z$	$=e, 2, 3$
5	$x = y \land y = z \to x = z$	$\to i, 2{-}4$

1	$x = y$	assume
2	$\mathsf{P}(x)$	assume
3	$\mathsf{P}(y)$	$=e, 1, 2$
4	$\mathsf{P}(x) \to \mathsf{P}(y)$	$\to i, 2{-}3$
5	$x = y \to (\mathsf{P}(x) \to \mathsf{P}(y))$	$\to i, 1{-}4$

□

3.3.2 Natural numbers

The following axioms characterize natural numbers, addition, and multiplication.

Definition 3.23 (Natural Numbers)**.** We introduce a fresh constant $0 \in \mathcal{F}^{(0)}$, a unary successor symbol $\mathsf{s} \in \mathcal{F}^{(1)}$ and restrict their models with two axioms.

$$\forall x\,(\mathsf{s}(x) \not\approx 0) \qquad \text{zero is smallest}$$

$$\forall x \forall y\,(\mathsf{s}(x) \approx \mathsf{s}(y) \to x \approx y) \qquad \text{injectivity of } \mathsf{s}$$

$$\forall x \forall y\,(x \approx y \to \mathsf{s}(x) \approx \mathsf{s}(y)) \qquad \text{congruence of } \mathsf{s}$$

$$\underbrace{G(0)}_{\text{base}} \land \forall x' \underbrace{(G(x') \to G(\mathsf{s}(x')))}_{\text{step case}} \to \forall x\, G(x) \qquad \text{induction schema}$$

Example 3.24. We may prove $\forall x\,(\mathsf{s}(x) \not\approx x)$ with $G(x) = \mathsf{s}(x) \not\approx x$ by induction.

$$\underbrace{\mathsf{s}(0) \not\approx 0}_{\text{base}} \land \forall x' \underbrace{(\mathsf{s}(x') \not\approx x' \to \mathsf{s}(\mathsf{s}(x')) \not\approx \mathsf{s}(x'))}_{\text{step case}} \to \forall x\, \mathsf{s}(x) \not\approx x$$

Definition 3.25 (Addition)**.** We introduce the binary addition symbol $+ \in \mathcal{F}_{\mathsf{f}}^{(2)}$ with two axioms about defining equalities of sums.

$$\forall x\,(x + 0 \approx x) \qquad \text{addition of zero}$$

$$\forall x \forall y\,(x + \mathsf{s}(y)) \approx \mathsf{s}(x + y) \qquad \text{addition of non-zero}$$

$$\forall x_1 \forall x_2 \forall y_1 \forall y_2\,(x_1 \approx y_1 \land x_2 \approx y_2 \to x_1 + y_1 \approx x_2 + y_2) \qquad \text{congruence of } +$$

Example 3.26.

$$\mathsf{s}(\mathsf{s}(\mathsf{s}(0))) + \mathsf{s}(\mathsf{s}(0)) \approx \mathsf{s}(\mathsf{s}(\mathsf{s}(\mathsf{s}(0))) + \mathsf{s}(0)) \approx \mathsf{s}(\mathsf{s}(\mathsf{s}(\mathsf{s}(\mathsf{s}(0))) + 0)) \approx \mathsf{s}(\mathsf{s}(\mathsf{s}(\mathsf{s}(\mathsf{s}(0)))))$$

Theorem 3.27. *Presburger arithmetic (Mojżesz Presburger, 1929), i.e. the first-order theory that includes the axioms for equality, natural numbers, induction schemata, and addition, is consistent, complete and decidable. The computational complexity of the decision problem is at least doubly exponential* $2^{2^{cn}}$ *(Fischer and Rabin, 1974), but less than triple exponential (Oppen, 1978. Berman, 1980).*

Definition 3.28 (Multiplication)**.** We introduce the binary multiplication symbol $\times \in \mathcal{F}_{\mathsf{f}}^{(2)}$ with two axioms about defining equalities of products.

$$\begin{aligned} &\forall x\,(x \times 0 \approx 0) && \text{multiplication by zero} \\ &\forall x \forall y\,(x \times \mathsf{s}(y) \approx (x \times y) + x) && \text{multiplication by non-zero} \\ &\forall x_1 \forall x_2 \forall y_1 \forall y_2\,(x_1 \approx y_1 \land x_2 \approx y_2 \to x_1 \times y_1 \approx x_2 \times y_2) && \text{congruence of } \times \end{aligned}$$

Theorem 3.29. *Peano Arithmetic (Giuseppe Peano, 1889), i.e. a first-order theory that extends Presburger Arithmetic with multiplication, is incomplete (Gödel‘s second incompleteness theorem in 1932) and undecidable.*

Theorem 3.30. *The axioms of Peano Arithmetic appear consistent (Gentzen, 1936).*

Lemma 3.31 (ACN)**.** *Addition and Multiplication on natural numbers are associative, commutative, and determine neutral elements.*

$$\begin{aligned} &\forall x \forall y \forall z\,(x \circ (y \circ z) \approx (x \circ y) \circ z) && \text{associativity of } \circ \in \{+, \times\} \\ &\forall x \forall y\,(x \circ y \approx y \circ x) && \text{commutativity of } \circ \in \{+, \times\} \\ &\forall x\,(x + 0 \approx x \land 0 + x \approx x) && \text{neutral element for } + \\ &\forall x\,(x \times \mathsf{s}(0) \approx x \land \mathsf{s}(0) \times x \approx x) && \text{neutral element for } \times \end{aligned}$$

4 Automation

Tous les hommes sont mortels
Fosca est un homme

$$\frac{\text{Tous les hommes sont mortels} \quad \text{Fosca est un homme}}{\text{Fosca est mortel}}$$

In this chapter we will demonstrate refutational complete proving procedures. It seems natural to hope for decision procedures for decidable fragments of first order logic (Section 3.2) or decidable first order theories (Section 3.3). We will demonstrate with simple examples that decision procedures do not automatically fall out from refutational complete procedures.

We know by Herbrand'ʃs theorem that each satisfiable set of (non-ground) clauses and each finite set of ground instances of a satisfiable set of (non-ground) clauses has a Herbrand model. And we know by compactness that if every finite subset of a set of clauses is satisfiable then this set is satisfiable. The satisfiability of a set of ground instances is decidable as we have already implicitly stated in Table 3.1 on page 16 with decidable class $[\,\exists^*, all, all\,]_=$. So the central idea of instantiation-based first order theorem proving is to find an unsatisfiable and finite set of ground instances for a given set of clauses. In general a failure in this search does not show satisfiability of the given set of clauses. In practice we make many detours in the search and we experience very finite resources of space and time, while in general there is no bound on the size of a smallest set of unsatisfiable ground instances.

To actually prove a theorem we first make use of the fact that a first-order formula is valid if and only if its negation is unsatisfiable. Second we efficiently transform the negated formula into an *equisatisfiable* set of clauses (e.g. with Tseytin's transformation [18]), i.e. the formula is satisfiable if and only if the set of clauses is satisfiable.

It would be sufficient to just luckily guess an unsatisfiable set of ground instances. Usually instantiation based automated provers generate a sequence of growing sets of ground instances such that an unsatisfiable one will be found for an arbitrary unsatisfiable set of clauses eventually.

First we translate axioms and lemmata into clausal normal form in Sections 4.1, then we look at Gilmore's Prover from 1960 in Section 4.2. After that we look at more modern calculi for refutation based first order theorem proving without equality in Section 4.3 and with equality in Section 4.4.

4.1 Theory axioms in CNF

In the previous chapter we expressed axioms and lemmas of first order theories in FOF syntax. Since Gilmore's prover, resolution and Inst-Gen work on sets of clauses we transform those axioms into (at least) equisatisfiable representations in `CNF` syntax as summarized for equivalence, congruence, natural numbers, and induction in Table 4.1, for addition and multiplication in Table 4.2.

$x \approx x,\ x \not\approx y \vee y \approx x,\ x \not\approx y \vee y \not\approx z \vee x \approx z$	equivalence
$x \not\approx y \vee \mathsf{s}(x) \approx \mathsf{s}(y)$	congruence of s
$\mathsf{s}(x) \not\approx 0,\ \mathsf{s}(x) \not\approx \mathsf{s}(y) \vee x \approx y$	natural numbers
$\neg G(0) \vee \boxed{G(\mathsf{c}_G)} \vee G(x),\ \neg G(0) \vee \boxed{\neg G(\mathsf{s}(\mathsf{c}_G))} \vee G(x)$	induction schema

Table 4.1: The theory of natural numbers in `CNF`

$x_1 \not\approx y_1 \vee x_2 \not\approx y_2 \vee x_1 + y_1 \approx x_2 + y_2$	congruence of +
$x + 0 \approx x,\ x + \mathsf{s}(x) \approx \mathsf{s}(x + y)$	addition
$x_1 \not\approx y_1 \vee x_2 \not\approx y_2 \vee x_1 \times y_1 \approx x_2 \times y_2$	congruence of ×
$x \times 0 \approx 0,\ x \times \mathsf{s}(y) \approx (x \times y) + x$	multiplication

Table 4.2: Addition and multiplication in `CNF`

Example 4.1. For the formula $G(x) = \mathsf{s}(x) \not\approx x$ we state the induction axioms in `CNF` in the theory of natural numbers. We had to introduce a fresh constant $\mathsf{c_s}$ in this satisfiability transformation process.

$$\mathsf{s}(0) \approx 0 \vee \boxed{\mathsf{s}(\mathsf{c_s}) \not\approx \mathsf{c_s}} \vee \mathsf{s}(x) \not\approx x$$
$$\mathsf{s}(0) \approx 0 \vee \boxed{\mathsf{s}(\mathsf{s}(\mathsf{c_s})) \approx \mathsf{s}(\mathsf{c_s})} \vee \mathsf{s}(x) \not\approx x$$

4.2 Gilmore's Prover

In 1960 Paul Gilmore presented a first implementation of an automated theorem prover [8] for first order logic (without equality), which happened to use an instantiation-based approach. The procedure is complete, i.e. for every valid formula a refutation proof can be found eventually.

In practice this prover ran into memory issues or time outs more often than not. We will discuss reasons for this inefficiency after we have described and demonstrated the procedure.

First the negation of a sentence F has to be transformed into an equisatisfiable set of clauses. Then the prover‘s procedure creates a sequence of finite sets of ground instances S_k for the set of clauses $S \approx \neg F$ to prove the validity of a formula F. Each set S_k

contains all possible ground instances of S where all variables are substituted by elements of H_k from definition 2.31 of the Herbrand universe. Each S_k is then transformed into a disjunctive normal form where satisfiability is obvious. The procedure is aborted when an unsatisfiable S_k is encountered.

Procedure 1 (Gilmore‘s Prover)**.** We translate the negation of our formula F into an equisatisfiable set of clauses $\neg F \approx S = \bigcup_{i=1}^{n} \mathcal{C}_i$ with an efficient algorithm [18, 16]. Then we start our first iteration with $k = 0$.

1. We create the set of all ground terms up to term depth k, i.e. the partial Herbrand universe H_k according to Definition 2.31. We use H_k to create the set of clause instances S_k by substituting all variables in each clause by terms from H_k in any possible permutation.

$$S_k = \bigcup_{i=1}^{n} \{ \mathcal{C}_i\sigma \mid \mathcal{C}_i \in S,\, \sigma : \mathcal{V} \to H_k \}$$

2. We translate S_k into an equivalent disjunctive normal form (i.e. a disjunction of conjunctions of literals) where satisfiability is easily checked.

3. When every conjunction contains a pair of complementary literals then we exit the procedure and report unsatisfiability of S, hence validity of F.

   Otherwise we increase k by one and continue with step 1.

Gilmore’s procedure will eventually terminate for an unsatisfiable set of clauses. It enumerates all possible sets of ground instances iteratively and one of them must be unsatisfiable for an unsatisfiable set of clauses. However the number of iterations has no general upper bound. Otherwise it would be a decision procedure for satisfiability in first order logic which does not exist because of undecidability of satisfiability in first order logic.

Lemma 4.2. *Gilmore’s procedure is a decision procedure for monadic first order logic (Examples 3.13 and 4.3) and the Schönfinkel-Bernays fragment (see Table 3.1) of First Order Logic.*

Proof. In the absence of non-constant function symbols the set $H'_{i+1} = \emptyset$ is empty. The procedure can stop after the first iteration because $H_i = H_0$ and $S_i = S_0$ for all $i \geq 0$, i.e. after the first iteration no new terms are added to the Herbrand model and no new ground instances can be generated. □

Following Gilmore’s prover we can easily prove the syllogism from above.

Example 4.3. First we translate the syllogism into a formula F in first order logic.

$$F = A \to (B \to C) \equiv \neg(A \wedge B) \vee C \equiv (A \wedge B) \to C \qquad \text{formula}$$

$$A = \forall x\, (\mathsf{human}(x) \to \mathsf{mortal}(x) \qquad \text{theory}$$

$$B = \mathsf{human}(\mathsf{fosca}) \qquad \text{fact}$$

$$C = \mathsf{mortal}(\mathsf{fosca}) \qquad \text{conjecture}$$

Then we negate the formula to clausal normal form $S_{(4.3)} = A \wedge B \wedge \neg C \equiv \neg F$. Since there is exactly one constant we get $H_0 = \{\mathsf{fosca}\}$ and $S_0 = \{(\neg\mathsf{human}(\mathsf{fosca}) \vee \mathsf{mortal}(\mathsf{fosca})) \wedge \mathsf{human}(\mathsf{fosca}) \wedge \neg\mathsf{mortal}(\mathsf{fosca})\}$ in our first iteration. As last step we transform the single formula in the set of ground instances S_0 into a disjunctive normal form for easy satisfiability checking.

$$\begin{gathered}(\neg\mathsf{human}(\mathsf{fosca}) \wedge \mathsf{human}(\mathsf{fosca}) \wedge \neg\mathsf{mortal}(\mathsf{fosca})) \\ \vee \\ (\mathsf{mortal}(\mathsf{fosca}) \wedge \mathsf{human}(\mathsf{fosca}) \wedge \neg\mathsf{mortal}(\mathsf{fosca}))\end{gathered}$$

Both conjunctions contain complementary literals and we conclude the negated formula is unsatisfiable and the given syllogism holds.

Example 4.4. Let $k \in \mathbb{N}$ be an arbitrary but fixed number. Consider the unsatisfiable set of clauses $S_{(4.4)} = \{\, \neg\mathcal{L}_1,\, \mathcal{L}_2 \,\} = \{\, \neg\mathsf{E}(x, \mathsf{s}(x)),\, \mathsf{E}(\mathsf{s}^k(y), \mathsf{s}(\mathsf{s}^k(y))) \,\}$. The sets of instances S_{i+1} are satisfiable for all $i+1 < k$. The set of instances S_k is clearly unsatisfiable.

$$\begin{aligned}
H'_0 &= \{\, \mathsf{z} \,\} & S_0 &= \{\, \neg\mathsf{E}(\mathsf{z}, \mathsf{s}(\mathsf{z})),\, \mathsf{E}(\mathsf{s}^k(\mathsf{z}),\, \mathsf{s}(\mathsf{s}^k(\mathsf{z}))) \,\} \subsetneq S_k \\
H'_{i+1} &= \{\, \mathsf{s}(\mathsf{s}^i(\mathsf{z})) \,\} & S_{i+1} &\supsetneq \{\, \neg\mathsf{E}(\mathsf{s}^{i+1}(\mathsf{z}), \mathsf{s}(\mathsf{s}^{i+1}(\mathsf{z}))),\, \mathsf{E}(\mathsf{s}^k(\mathsf{s}^{i+1}(\mathsf{z})), \mathsf{s}(\mathsf{s}^k(\mathsf{s}^{i+1}(\mathsf{z})))) \,\} \\
H'_k &= \{\, \mathsf{s}^k(\mathsf{z}) \,\} & S_k &\supsetneq \{\, \neg\mathsf{E}(\mathsf{s}^k(\mathsf{z}), \mathsf{s}(\mathsf{s}^k(\mathsf{z}))),\, \mathsf{E}(\mathsf{s}^k(\mathsf{s}^k(\mathsf{z})), \mathsf{s}(\mathsf{s}^k(\mathsf{s}^k(\mathsf{z})))) \,\}
\end{aligned}$$

We've produced $2 \cdot k$ irrelevant instances, i.e. these clauses did not cause any conflict in propositional satisfiability. In this example the guess for a finite unsatisfiable set of ground instances appears feasible and yields a smaller set.

$$\{\, \neg\mathcal{L}_1\sigma,\, \mathcal{L}_2\sigma \,\} \qquad \sigma = \{x \mapsto \mathsf{s}^k(\mathsf{z}),\, y \mapsto \mathsf{z}\}$$

Example 4.5. Consider the satisfiable set of clauses $S_{4.5} = \{\neg\mathsf{E}(\mathsf{z}, \mathsf{s}(x)\}$. This set is clearly in the decidable Ackermann fragment of first order logic. But the procedure yields an infinite sequence of distinct and satisfiable sets $S_{k \geq 0}$:

$$\begin{aligned}
H'_0 &:= \{\mathsf{z}\} & S_0 &:= \{\neg\mathsf{E}(\mathsf{z}, \mathsf{s}(\mathsf{z}))\} && \text{satisfiable} \\
H'_{i+1} &:= \{\mathsf{s}(\mathsf{s}^i(\mathsf{z}))\} & S_{i+1} &:= S_i \,\dot\cup \{\neg\mathsf{E}(\mathsf{z}, \mathsf{s}(\mathsf{s}^i(\mathsf{z})))\} && \text{satisfiable}
\end{aligned}$$

So Gilmore's prover wouldn't terminate on this simple and decidable problem.

We have observed three main disadvantages in Gilmore's procedure.

1. The generation of instances is unguided. With each iteration exponentially many (mostly useless) instances are created — depending on the number and the arities of used symbols.

$$\begin{aligned}
|S_i| &= \sum_n \left(|\mathcal{F}_{\mathsf{P}}^{(n)}| \cdot |H_i|^n \right) & |H_0| &\geq 1 \\
|S_{i+1}| &= \sum_n \left(|\mathcal{F}_{\mathsf{P}}^{(n)}| \cdot |H_{i+1}|^n \right) & |H_{i+1}| &\geq \sum_{n>0} \left(|\mathcal{F}_{\mathsf{f}}^{(n)}| \cdot |H_i|^n \right)
\end{aligned}$$

This makes disadvantage No. 2 which is invoked at every iteration even worse.

2. The check for unsatisfiability is far from efficient. The transformation from a set of clauses to a formula in disjunctive normal form[1] usually introduces an exponential[2] blow up in the size of the formula — depending on the number of clauses n in the set and the number of literals c_i per clause $\mathcal{C}_i$ we get the disjunction of $\prod_1^n c_i$ conjunctions of n literals.

$$\bigwedge_{i=1}^{n}\left(\bigvee_{j_i=1}^{c_i} p_{(i,j_i)}\right) \equiv \bigvee_{(j_1,\ldots,j_n)}\left(\bigwedge_{i=1}^{n} p_{(i,j_i)}\right) \qquad \text{with } (j_1,\ldots,j_n)\in\prod_{i=1}^{n}\{1,\ldots,c_i\}$$

   In total the number of literals in the set of clauses is $n\cdot\bar{c}_{arith}$, while the equivalent disjunctive normal form contains $(\bar{c}_{geom})^n\cdot n$ literals[3].

$$\{1\}\times\{1,2\}\times\{1,2,3\} = \{(1,1,1),(1,1,2),(1,1,3),(1,2,1),(1,2,2),(1,2,3)\}$$

3. The procedure will not terminate for satisfiable sets when at least one non-constant predicate symbol is used in the set of clauses and one non-constant function symbol is available, e.g. for $S=\{\,\mathsf{P}(\mathsf{f}(x))\,\}$ we get

$$\begin{aligned} H_0 &= \{\,\mathsf{c}\,\} & S_0 &= \{\,\mathsf{P}(\mathsf{f}(\mathsf{c}))\,\}\\ H_{i+1} &= \bigcup_{k=0}^{i+1}\{\,\mathsf{f}^k(\mathsf{c}))\,\} & S_{i+1} &= \{\,\mathsf{P}(\mathsf{f}(t))\mid t\in H_{i+1}\,\}\\ \mathsf{f}^{i+1}(\mathsf{c}) &\in H_{i+1}\setminus H_i & \mathsf{P}(\mathsf{f}(\mathsf{f}^{i+1}(\mathsf{c}))) &\in S_{i+1}\setminus S_i \end{aligned}$$

Issue 2 was already implicitly addressed in 1960 [6] (which also incorporated the basic idea of resolution — on ground instances of terms) and refined in 1962 [5] by Davis, Putnam, Longeman, and Loveland, which was the starting point for the development of efficient propositionally satisfiability checkers, i.e. efficient modern SAT solvers.

4.3 Proving without Equality

In this section we already add the equality symbol $\approx$ into our formulae, but we treat it not different from an arbitrary binary predicate symbol with infix notation.

4.3.1 Resolution

Definition 4.6 (Resolution calculus)**.** Let A, B be atoms and $\mathcal{C},\mathcal{D}$ be clauses.

$$\frac{A\vee\mathcal{C} \quad \neg B\vee\mathcal{D}}{(\mathcal{C}\vee\mathcal{D})\sigma}\ \text{Resolution} \qquad\qquad \frac{A\vee B\vee\mathcal{C}}{(A\vee\mathcal{C})\sigma}\ \text{Factoring}$$

[1] In contrast the linear Tseytin transformation yields an equisatisfiable conjunctive normal form.

[2] The existence of a polynomial algorithm for the transformation of an arbitrary propositional formula into an equisatisfiable formula in *disjunctive normal form* (where satisfiability is a linear check) would show that SAT in $\mathcal{P}$ and would prove $\mathcal{P}=\mathcal{NP}$, which remains unknown.

[3] Geometric mean $\bar{c}_{geom} := \left(\prod_1^n c_i\right)^{\frac{1}{n}}$, arithmetic mean $\bar{c}_{arith} := \left(\sum_1^n c_i\right)\cdot\frac{1}{n}$, and $\bar{c}_{geom}\le\bar{c}_{arith}$.

$$\text{where } \sigma = \mathrm{mgu}(A, B)$$

Example 4.7. Modus tollens is a special case of resolution ($F \to G \equiv \neg F \vee G$).

$$\frac{F \to G \quad \neg G}{\neg F}\ \substack{\text{modus}\\ \text{tollens}} \qquad\qquad \frac{\neg F \vee G \quad \neg G}{\neg F}$$

Example 4.8. We easily infer the empty clause and conclude unsatisfiability of the set of clauses $S_{4.8} = \{\, \mathsf{s}(x) \not\approx x,\, \mathsf{s}(\mathsf{s}^k(y)) \approx \mathsf{s}^k(y) \,\}$.

$$\frac{\mathsf{s}(x) \not\approx x \quad \mathsf{s}(\mathsf{s}^k(y)) \approx \mathsf{s}^k(y)}{\square}\ \{x \mapsto \mathsf{s}^k(y)\}$$

Example 4.9. With observe an infinite sequence of resolution steps from satisfiable set $S_{4.9} = \{\, \mathsf{s}(x) \not\approx x,\, \mathsf{s}(y) \not\approx \mathsf{s}(z) \vee y \approx z \,\}$.

$$\frac{\mathsf{s}^{i+1}(x) \not\approx \mathsf{s}^i(x) \quad \mathsf{s}(x') \not\approx \mathsf{s}(y') \vee x' \approx y'}{\mathsf{s}^{i+2}(x) \not\approx \mathsf{s}^{i+1}(x)}\ \{x' \mapsto \mathsf{s}^{i+1}(x), y' \mapsto \mathsf{s}^i(x)\} \tag{$i \geq 0$}$$

We can easily notice disadvantages in resolution.

1. If clauses contain more than two literals the resolution inference rule yields clauses with more literals than the sources.

2. For two clauses $\mathcal{C}$ with c literals and $\mathcal{D}$ with d literals we have to check all pairings of positive literals in $\mathcal{C}$ with negative literals in $\mathcal{D}$ and all pairing of negative literals in $\mathcal{C}$ with positive literals in $\mathcal{D}$ for clashing, i.e. in the worst case we have $c \times d$ pairs to check, which makes disadvantage No. 1 even worse (see Example 4.10). This workload can be reduced with ordered resolution as presented in Section 4.3.2.

3. Resolution is sound but not complete in the presence of equality, e.g. we expect the set of clauses $\{\mathsf{f}(\mathsf{a}) \approx \mathsf{c}, \mathsf{P}(\mathsf{c}), \neg\mathsf{P}(\mathsf{f}(\mathsf{a}))\}$ or even simpler the set with one clause $\{\mathsf{a} \not\approx \mathsf{a}\}$ to be unsatisfiable, but neither resolution nor factoring are applicable. This can be addressed with equality axioms or equality inference rules as presented in Section 4.4.

Example 4.10. One of nine derivable clauses à 4 literals from two clauses à 3 literals.

$$\frac{\mathsf{P}(x, y) \vee \mathsf{P}(\mathsf{a}, z) \vee \mathsf{P}(z, \mathsf{b}) \quad \neg\mathsf{P}(x', y') \vee \neg\mathsf{P}(\mathsf{a}, z') \vee \neg\mathsf{P}(z', \mathsf{b})}{\mathsf{P}(\mathsf{a}, z) \vee \mathsf{P}(z, \mathsf{b}) \vee \neg\mathsf{P}(a, z') \vee \neg\mathsf{P}(z', b)}$$

4.3.2 Ordered resolution

Definition 4.11 (Ordered Resolution)**.** Let A, B be atoms and $\mathcal{C}, \mathcal{D}$ be clauses.

$$\frac{A \vee \mathcal{C} \quad \neg B \vee \mathcal{D}}{(\mathcal{C} \vee \mathcal{D})\sigma}\ \substack{\text{Ordered}\\ \text{Resolution}} \qquad\qquad \frac{A \vee B \vee \mathcal{C}}{(A \vee \mathcal{C})\sigma}\ \substack{\text{Ordered}\\ \text{Factoring}}$$

where $\sigma = \text{mgu}(A, B)$, $A\sigma$ is strictly maximal in $\mathcal{C}\sigma$, $\neg B\sigma$ is maximal in $\mathcal{D}\sigma$.

Example 4.12. One clause à 4 literals is derivable from the two clauses in Example 4.10 on the preceding page with $\mathsf{P}(x, y) \succ \mathsf{P}(\mathsf{a}, z) \succ \mathsf{P}(z, \mathsf{b})$ for $\{x \mapsto \mathsf{a}, y \mapsto \mathsf{a}, z \mapsto \mathsf{b}\}$.

$$\frac{\mathsf{P}(x, y) \lor \mathsf{P}(\mathsf{a}, z) \lor \mathsf{P}(z, \mathsf{b}) \qquad \neg\mathsf{P}(x', y') \lor \neg\mathsf{P}(\mathsf{a}, z') \lor \neg\mathsf{P}(z', \mathsf{b})}{\mathsf{P}(\mathsf{a}, z) \lor \mathsf{P}(z, \mathsf{b}) \lor \neg\mathsf{P}(a, z') \lor \neg\mathsf{P}(z', b)}$$

Example 4.13. With an ordering such that $\mathsf{s}(y) \approx \mathsf{s}(z)) \succ y \approx z$ on atoms and $\neg A \succ A$ the satisfiable set $S_{4.9}$ saturates with ordered resolution, because the strictly maximal literals $\mathsf{s}(x) \not\approx x$ and $\mathsf{s}(y) \not\approx \mathsf{s}(z)$ do not clash and the ordered resolution rule is not applicable.

Definition 4.14 (Order on clauses)**.** multiset order

Lemma 4.15. *Ordered resolution is refutation complete.*

Proof. If no new resolvant can be derived and the empty clause is not in the set of clauses we have a model.

The resolvant $(\mathcal{C} \lor \mathcal{D})\sigma$ is smaller than $(A \lor \mathcal{C})\sigma$ and $(\neg B \lor \mathcal{D})\sigma$
$\neg B \succ_{\mathsf{L}} A$ because $A\sigma = B\sigma = C$ and $\neg X \succ_{\mathsf{L}} X$. □

4.3.3 InstGen

Gilmore's prover blindly constructs new ground instances.

Definition 4.16 (`Inst-Gen`)**.** Let A, B be atoms and $\mathcal{C}, \mathcal{D}$ be clauses.

$$\frac{A \lor \mathcal{C} \qquad \neg B \lor \mathcal{D}}{(A \lor \mathcal{C})\sigma \qquad (\neg B \lor \mathcal{D})\sigma}\ \texttt{Inst-Gen}$$

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

Example 4.17. With `Inst-Gen` we immediately can derive a helpful clause from set $S_{(4.4)} = \{\, \neg\mathsf{E}(x, \mathsf{s}(x)),\ \mathsf{E}(\mathsf{s}^k(y), \mathsf{s}(\mathsf{s}^k(y)))\,\}$ introduced in Example 4.4.

$$\frac{\neg\mathsf{E}(x, \mathsf{s}(x)) \quad \mathsf{E}(\mathsf{s}^k(y), \mathsf{s}(\mathsf{s}^k(y)))}{\neg\mathsf{E}(\mathsf{s}^k(y), \mathsf{s}(\mathsf{s}^k(y))) \qquad \mathsf{E}(\mathsf{s}^k(y), \mathsf{s}(\mathsf{s}^k(y)))}\ \{x \mapsto \mathsf{s}^k(y)\}$$

and we conclude unsatisfiability because of propositional unsatisfiability of

$$\{\, \mathsf{E}(\mathsf{s}^k(\mathsf{c}_\perp), \mathsf{s}(\mathsf{s}^k(\mathsf{c}_\perp))), \neg\mathsf{E}(\mathsf{s}^k(\mathsf{c}_\perp), \mathsf{s}(\mathsf{s}^k(\mathsf{c}_\perp)))\,\}$$

Example 4.18. With `Inst-Gen` we cannot derive any new clause from set $S_{(4.5)} = \{\, \mathsf{E}(\mathsf{z}, y)\,\}$ introduced in Example 4.5 and we conclude satisfiability of the `Inst-Gen`-saturated set $S_{(4.5)}$ because of the propositional satisfiability of $S_{(4.5)}\sigma_\perp$.

Lemma 4.19. *The set of clauses $S_0 = S \cup \{A \lor \mathcal{C}, \neg B \lor \mathcal{D}\}$ is satisfiable if and only if the derived set of clauses $S_1 = S_0 \cup \{(A \lor \mathcal{C})\sigma, (\neg B \lor \mathcal{D})\sigma\}$ with $\sigma = \text{mgu}(A, B)$ is satisfiable.*

Proof. If S_1 is satisfiable then there exists an interpretation that satisfies all clauses in S_1. The same interpretation models all clauses in S_0 because $S_0 \subseteq S_1$. In reverse S_1 cannot be satisfiable if S_0 is not.

□

Procedure 2 (Inst-Gen-Loop)**.** As in Gilmore's prover (Procedure 1) we translate the negation of our formula F into an equisatisfiable set of clauses S_0. Then we introduce a distinct constant symbol $\mathsf{c}_\bot \notin \mathcal{F}(S_0)$ even when there are constant symbols in the signature. We start our first iteration with $k = 0$.

1. We construct a set $S_k\,\sigma_\bot$ of ground instances from S_k where instantiator $\sigma_\bot := \{x \mapsto \mathsf{c}_\bot \mid x \in \mathcal{V}\!ars(S_k)\}$ substitutes all occurring variables with constant symbol $\mathsf{c}_\bot$.

2. We check the decidable satisfiability of $S_k\,\sigma_\bot$ with a modern `SAT` or `SMT`-solver.

   If $S_k\,\sigma_\bot$ is unsatisfiable then we exit the procedure and report unsatisfiability of S, i.e the original formula F is valid.

3. The set $S_k\,\sigma_\bot$ is satisfiable, hence we can retrieve a model $\mathcal{M}_k \models S_k\sigma_\bot$. We select one literal $L_i = \mathrm{sel}(\mathcal{C}_i)$ per clause $\mathcal{C}_i \in S_k$ such that the each grounded selected literal holds in model $\mathcal{M}_k \models L_i\sigma_\bot$ for all $i \leq |S_k|$.

4. We search for pairs of selected literals $(A, \neg B) = (L_i, L_j^c)$ such that the most general unifier $\tau = \mathrm{mgu}(A, B)$ exists.

5. We set $S_{k+1} ::= S_k$ and for each pair of clashing literals (L_i, L_j^c) we apply `Inst-Gen` to the originating clauses $\{\, \mathcal{C}_i, \mathcal{C}_j \,\} = \{\, L_i \lor \mathcal{C}, L_j \lor \mathcal{D} \,\}$ to add new (not necessarily ground) instances to S_{k+1}.

   If no new clauses were added, i.e. $S_{k+1} = S_k$ after all pairs were processed we exit the procedure and report satisfiability of S, i.e. the original formula F is not valid.

6. We increase k by 1 and continue with step 1.

Example 4.20. The selected literals of the first and the second clause change between iterations.

$$\begin{aligned}
S_0 &= \{\, \mathsf{P}(\mathsf{a}) \lor \mathsf{Q}(\mathsf{a}), \mathsf{P}(\mathsf{a}) \lor \neg\mathsf{Q}(y), \neg\mathsf{P}(x) \,\} \\
S_0\sigma_\bot &= \{\, \mathsf{P}(\mathsf{a}) \lor \mathsf{Q}(\mathsf{a}), \mathsf{P}(\mathsf{a}) \lor \neg\mathsf{Q}(\mathsf{c}_\bot), \neg\mathsf{P}(\mathsf{c}_\bot) \,\} && \text{(satisfiable)} \\
& \frac{\mathsf{P}(\mathsf{a}) \lor \mathsf{Q}(*) \quad \neg\mathsf{P}(x)}{\mathsf{P}(\mathsf{a}) \lor \mathsf{Q}(*) \quad \neg\mathsf{P}(\mathsf{a})}\; x \mapsto \mathsf{a} && * \in \{\mathsf{a}, y\} \\
S_1 &= \{\, \mathsf{P}(\mathsf{a}) \lor \mathsf{Q}(\mathsf{a}), \mathsf{P}(\mathsf{a}) \lor \neg\mathsf{Q}(y), \neg\mathsf{P}(x), \neg\mathsf{P}(\mathsf{a}) \,\} \\
S_1\sigma_\bot &= \{\, \mathsf{P}(\mathsf{a}) \lor \mathsf{Q}(\mathsf{a}), \mathsf{P}(\mathsf{a}) \lor \neg\mathsf{Q}(\mathsf{c}_\bot), \neg\mathsf{P}(\mathsf{c}_\bot), \neg\mathsf{P}(\mathsf{a}) \,\} && \text{(satisfiable)} \\
& \frac{\mathsf{P}(\mathsf{a}) \lor \mathsf{Q}(\mathsf{a}) \quad \mathsf{P}(\mathsf{a}) \lor \neg\mathsf{Q}(y)}{\mathsf{P}(\mathsf{a}) \lor \mathsf{Q}(\mathsf{a}) \quad \mathsf{P}(\mathsf{a}) \lor \neg\mathsf{Q}(\mathsf{a})}\; y \mapsto \mathsf{a} \\
S_2 &= \{\, \mathsf{P}(\mathsf{a}) \lor \mathsf{Q}(\mathsf{a}), \mathsf{P}(\mathsf{a}) \lor \neg\mathsf{Q}(y), \neg\mathsf{P}(x), \neg\mathsf{P}(\mathsf{a}), \mathsf{P}(\mathsf{a}) \lor \neg\mathsf{Q}(\mathsf{a}) \,\} \\
S_2\sigma_\bot &= \{\, \mathsf{P}(\mathsf{a}) \lor \mathsf{Q}(\mathsf{a}), \mathsf{P}(\mathsf{a}) \lor \neg\mathsf{Q}(\mathsf{c}_\bot), \neg\mathsf{P}(\mathsf{c}_\bot), \neg\mathsf{P}(\mathsf{a}), \mathsf{P}(\mathsf{a}) \lor \neg\mathsf{Q}(\mathsf{a}) \,\} && \text{(unsatisfiable)}
\end{aligned}$$

Lemma 4.21. *The $\tau = \mathrm{mgu}(A, B)$ in Procedure 2, step 4 is a proper instantiator, i.e. it is not a variable renaming.*

Proof. Assume τ in Procedure 2 is a renaming, then we have $A\tau\sigma_\perp = A\sigma_\perp$, $B\tau\sigma_\perp = B\sigma_\perp$, and by definition of the most general unifier $A\tau = B\tau$. Hence $A\sigma_\perp = B\sigma_\perp$ which contradicts that $M_k \models A\sigma_\perp, \neg B\sigma_\perp$ by definition of step 3. Hence the assumption is false and τ must be a proper instantiator. □

Example 4.22. Let $S_0 = S_{(4.4)}$ be the set of unsatisfiable clauses from Example 4.4. Then the initial set of ground instances $S_0\sigma_\perp = \{\neg\mathsf{E}(\mathsf{c}_\perp, \mathsf{s}(\mathsf{c}_\perp)), \mathsf{E}(\mathsf{s}^k(\mathsf{c}_\perp), \mathsf{s}(\mathsf{s}^k(\mathsf{c}_\perp)))\}$ is satisfiable with domain $A = \{\mathsf{c}_\perp, \mathsf{s}(\mathsf{c}_\perp), \mathsf{s}^k(\mathsf{c}_\perp), \mathsf{s}(\mathsf{s}^k(\mathsf{c}_\perp))\}$ and predicate interpretation $\mathsf{E}^{\mathcal{I}} = \{(\mathsf{s}^k(\mathsf{c}_\perp), \mathsf{s}(\mathsf{s}^k(\mathsf{c}_\perp)))\} \subseteq A^2$. With just two unit clauses we easily find the only pair of clashing literals and compute the unifier $\tau = \{x \mapsto \mathsf{s}(s^k(y))\}$. By application of `Inst-Gen` we construct our next set of clauses $S_1 = S_0 \dot\cup \{\neg\mathsf{E}(\mathsf{s}^k(y), \mathsf{s}(\mathsf{s}^k(y)))\}$ and get an unsatisfiable set of ground instances $S_1\sigma_\perp$.

4.4 Proving with Equality

So far we have treated the equality symbol like any other binary predicate symbol, which can yield models where $\mathsf{a} \not\approx \mathsf{a}$ holds. Understandably, we are only interested in normal models or at least in models that implies the existence of a normal model. We have already seen that a normal Herbrand model might not exist, but we can ensure that we find only desired models.

4.4.1 Adding equality axioms

Theorem 4.23. *[9] Any set of clauses (a formula) has a* normal *model if and only if it has a model that satisfies the equality axioms, i.e. reflexivity, symmetry, transitivity, and congruence for all function symbols $\mathsf{f} \in \mathcal{F}_\mathsf{f}$ and all predicate symbols $\mathsf{P} \in \mathcal{F}_\mathsf{P}$.*

Example 4.24 (Ordered resolution with equality axioms)**.** We add the equality axioms to a small set of clauses $S = \{\mathsf{s}(x) \not\approx 0,\ \mathsf{s}(x) \not\approx \mathsf{s}(y) \vee x \approx y\}$ and mark maximal literals.

$$x \approx x,\ x \approx y \vee y \not\approx x,\ x \approx z \vee x \not\approx y \vee y \not\approx z \qquad \approx\text{-equivalence}$$

$$\mathsf{s}(x) \approx \mathsf{s}(y) \vee x \not\approx y \qquad \mathsf{s}\text{-congruence}$$

$$\mathsf{s}(x) \not\approx 0,\ \mathsf{s}(x) \not\approx \mathsf{s}(y) \vee x \approx y \qquad S$$

With ordered resolution we cannot infer new clauses from clauses in S. But we can apply rules of ordered resolution to pairs of equality axioms and clauses, although most derivations are ignorable.

$$\frac{\textcolor{blue}{x \approx x} \quad x' \approx y' \lor \textcolor{blue}{y' \not\approx x'}}{x \approx x}\ \{x' \mapsto x, y' \mapsto x\} \qquad R, S \vdash R$$

$$\frac{\textcolor{blue}{x \approx x} \quad x' \approx z' \lor \textcolor{blue}{x' \not\approx y'} \lor y' \not\approx z'}{x \approx z' \lor x \not\approx z'}\ \{x' \mapsto x, y' \mapsto x\} \qquad R, T \vdash \mathsf{true}$$

$$\frac{\textcolor{blue}{x \approx x} \quad x' \approx z' \lor x' \not\approx y' \lor \textcolor{blue}{y' \not\approx z'}}{x' \approx x \lor x' \not\approx x}\ \{y' \mapsto x, z' \mapsto x\} \qquad R, T \vdash \mathsf{true}$$

$$\frac{x \approx y \lor \textcolor{blue}{y \not\approx x} \quad \textcolor{blue}{\mathsf{s}(x') \approx \mathsf{s}(y')} \lor x' \not\approx y'}{\mathsf{s}(y') \approx \mathsf{s}(x') \lor x' \not\approx y'}\ \{y \mapsto \mathsf{s}(x'), x \mapsto \mathsf{s}(y')\} \qquad S, C \vdash C_S$$

$$\frac{x \approx z \lor \textcolor{blue}{x \not\approx y} \lor y \not\approx z \quad \textcolor{blue}{\mathsf{s}(x') \approx \mathsf{s}(y')} \lor x' \not\approx y'}{\mathsf{s}(x') \approx z \lor \mathsf{s}(y') \not\approx z \lor x' \not\approx y'}\ \{x \mapsto \mathsf{s}(x'), y \mapsto \mathsf{s}(y')\} \qquad T, C \vdash ?$$

$$\frac{x \approx z \lor x \not\approx y \lor \textcolor{blue}{y \not\approx z} \quad \textcolor{blue}{\mathsf{s}(x') \approx \mathsf{s}(y')} \lor x' \not\approx y'}{x \approx \mathsf{s}(y') \lor x \not\approx \mathsf{s}(x') \lor x' \not\approx y'}\ \{y \mapsto \mathsf{s}(x'), z \mapsto \mathsf{s}(y')\} \qquad T, C \vdash ?$$

$$\frac{\textcolor{blue}{\mathsf{s}(x) \approx \mathsf{s}(y)} \lor x \not\approx y \quad x' \approx y' \lor \textcolor{blue}{\mathsf{s}(x') \not\approx \mathsf{s}(y')}}{x \not\approx y \lor x \approx y}\ \{x' \mapsto x, y' \mapsto y\} \qquad C, I \vdash \mathsf{true}$$

Example 4.25. We extend our set with clause $\mathsf{s}(\mathsf{s}(x)) \approx \mathsf{s}(0)$ that clashes with injectivity of s.

$$\frac{s(x') \not\approx 0 \qquad \dfrac{x' \approx y' \lor \textcolor{blue}{\mathsf{s}(x') \not\approx \mathsf{s}(y')} \quad \mathsf{s}(\mathsf{s}(x)) \approx \mathsf{s}(0)}{\mathsf{s}(x) \approx 0}\ \{\, x' \mapsto \mathsf{s}(x), y' \mapsto 0 \,\}}{\square}\ \{\, x' \mapsto x \,\}$$

Example 4.26 (`Inst-Gen` with equality axioms)**.** The default grounding for `Inst-Gen` substitutes *all* variables with one constant function symbol. We notice that the selection process is unfortunate, because the selected *positive* literals of each axiom but congruence clash with $\mathsf{s}(x) \not\approx 0$.

$$\begin{array}{rr}
\textcolor{green}{\mathsf{c}_\perp \approx \mathsf{c}_\perp} & \text{reflexivity} \\
\mathsf{c}_\perp \approx \mathsf{c}_\perp \lor \textcolor{red}{\mathsf{c}_\perp \not\approx \mathsf{c}_\perp} & \text{symmetry} \\
\mathsf{c}_\perp \approx \mathsf{c}_\perp \lor \textcolor{red}{\mathsf{c}_\perp \not\approx \mathsf{c}_\perp} \lor \textcolor{red}{\mathsf{c}_\perp \not\approx \mathsf{c}_\perp} & \text{transitivity} \\
\textcolor{green}{\mathsf{s}(\mathsf{c}_\perp) \approx \mathsf{s}(\mathsf{c}_\perp)} \lor \textcolor{red}{\mathsf{c}_\perp \not\approx \mathsf{c}_\perp} & \text{congruence} \\
\mathsf{c}_\perp \approx \mathsf{c}_\perp \lor \textcolor{red}{\mathsf{s}(\mathsf{c}_\perp) \not\approx \mathsf{s}(\mathsf{c}_\perp)} & \text{injectivity} \\
\textcolor{green}{\mathsf{s}(\mathsf{c}_\perp) \not\approx 0} &
\end{array}$$

$$\frac{0 \not\approx \mathsf{s}(x') \quad x \approx y \lor y \not\approx x}{0 \approx \mathsf{s}(x') \lor \mathsf{s}(x') \approx 0}\ x \mapsto 0, y \mapsto \mathsf{s}(x')$$

$$\frac{0 \not\approx \mathsf{s}(x') \quad x \approx y \lor \mathsf{s}(x) \not\approx \mathsf{s}(y)}{0 \approx \mathsf{s}(x') \lor \mathsf{s}(x') \approx 0}\ x \mapsto 0, y \mapsto \mathsf{s}(x')$$

$$\frac{\mathsf{s}(x') \not\approx \mathsf{s}(y') \lor x' \approx y' \quad \mathsf{s}(x) \not\approx x}{\boxed{\mathsf{s}(\mathsf{s}(x)) \not\approx \mathsf{s}(x)} \lor \mathsf{s}(x) \approx x} \; \{x' \mapsto \mathsf{s}(x), y' \mapsto x\}$$

$$\frac{\mathsf{s}(x') \not\approx \mathsf{s}(y') \lor x' \approx y' \quad \boxed{\mathsf{s}^{i+2}(x) \not\approx \mathsf{s}^{i+1}(x)} \lor \mathsf{s}^{i+1}(x) \approx \mathsf{s}^{i}(x)}{\mathsf{s}(\mathsf{s}^{i+2}(x)) \not\approx \mathsf{s}(\mathsf{s}^{i+1}(x)) \lor \mathsf{s}^{i+2}(x) \approx \mathsf{s}^{i+1}(x)} \; \{x' \mapsto \mathsf{s}^{i+2}(x), y' \mapsto \mathsf{s}^{i+1}(x)\}$$

for all $i \geq 0$.

$$\begin{array}{rr}
\mathsf{c}_{x_0} \approx \mathsf{c}_{x_0} & \approx\text{-reflexivity} \\
\mathsf{c}_{x_1} \approx \mathsf{c}_{y_1} \lor \mathsf{c}_{y_1} \not\approx \mathsf{c}_{x_1} & \approx\text{-symmetry} \\
\mathsf{c}_{x_2} \approx \mathsf{c}_{z_2} \lor \mathsf{c}_{x_2} \not\approx \mathsf{c}_{y_2} \lor \mathsf{c}_{y_2} \not\approx \mathsf{c}_{z_2} & \approx\text{-transitivity} \\
\mathsf{s}(\mathsf{c}_{x_3}) \approx \mathsf{s}(\mathsf{c}_{y_3}) \lor \mathsf{c}_{x_3} \not\approx \mathsf{c}_{y_3} & \mathsf{s}\text{-congruence} \\
\mathsf{s}(\mathsf{c}_{x_4}) \not\approx 0 & 0 \notin \mathrm{img}(\mathsf{s}) \\
\mathsf{s}(\mathsf{c}_{x_5}) \not\approx \mathsf{s}(\mathsf{c}_{y_5}) \lor \mathsf{c}_{x_5} \approx \mathsf{c}_{y_5} & \mathsf{s}\text{-injectivity}
\end{array}$$

4.4.2 Equality inference rules

Instead of adding equality axioms for an equality predicate symbol, we add specific equality inference rules for completeness.

Superposition

As in ordered resolution the unsatisfiability of a set of clauses is shown if and only if the empty clause can be derived. `-MISSING» Proof of completeness Reference «MISSING-`

Definition 4.27. Let A, B be predicates (not equations), $\mathcal{C}, \mathcal{C}', \mathcal{D}$ clauses, and s, s', t, u, v terms. The *superposition calculus* includes the following inference rules

- ordered resolution and ordered factoring

$$\frac{A \lor \mathcal{C} \quad \neg B \lor \mathcal{D}}{(\mathcal{C} \lor \mathcal{D})\sigma} \;(\mathsf{oR}) \qquad\qquad \frac{A \lor B \lor \mathcal{C}}{(A \lor \mathcal{C}')\sigma} \;(\mathsf{oF})$$

  where unifier $\sigma = \mathrm{mgu}(A, B)$ exists, instance $A\sigma$ is strictly maximal in $\mathcal{C}\sigma$, and instance $\neg B\sigma$ is maximal in $\mathcal{D}\sigma$.

- ordered paramodulation

$$\frac{s \approx t \lor \mathcal{C} \quad \neg A[s'] \lor \mathcal{D}}{(\mathcal{C} \lor \neg A[t] \lor \mathcal{D})\,\sigma} \;() \qquad\qquad \frac{s \approx t \lor \mathcal{C} \quad A[s'] \lor \mathcal{D}'}{(\mathcal{C} \lor A[t] \lor \mathcal{D}')\,\sigma} \;()$$

- superposition

  $$\frac{s \approx t \lor \mathcal{C} \quad u[s'] \not\approx v \lor \mathcal{D}}{(\mathcal{C} \lor u[t] \not\approx v \lor \mathcal{D})\,\sigma}\ () \qquad\qquad \frac{s \approx t \lor \mathcal{C} \quad u[s'] \approx v \lor \mathcal{D}}{(\mathcal{C} \lor u[t] \approx v \lor \mathcal{D})\,\sigma}\ (S_+)$$

  where unifier $\sigma = \mathrm{mgu}(s, s')$ exists, $s' \notin \mathcal{V}$, $t\sigma \not\succeq s\sigma$, $v\sigma \not\succeq u[s']\sigma$, $(s \approx t)\sigma$ is strictly maximal in $\mathcal{C}\sigma$, $\neg A[s']$ and $u[s'] \not\approx v$ are maximal in $\mathcal{C}\sigma$, $A[s']$ and $u[s'] \approx v$ are strictly maximal in $\mathcal{D}\sigma$, $(s \approx t)\sigma \not\succeq (u[s'] \approx v)\sigma$.

- equality resolution and equality factoring

  $$\frac{s \not\approx s' \lor \mathcal{C}}{\mathcal{C}\sigma}\ (\mathsf{R}_{\approx}) \qquad\qquad \frac{u \approx v \lor s \approx s' \lor \mathcal{C}'}{(v \not\approx s' \lor u \approx s' \lor \mathcal{C}')\sigma}\ (\mathsf{F}_{\approx})$$

  where $\sigma = \mathrm{mgu}(s, s')$ exists, $(s \not\approx s')\sigma$ is maximal in $\mathcal{C}$, $(s \approx s')\sigma$ is strictly maximal in $\mathcal{C}'$, $(s \approx s')\sigma \not\succeq (u \approx v)$. (????)

Example 4.28. With Superposition calculus no derivation rule is applicable to clauses of the set $S = \{\mathsf{s}(x) \not\approx 0, \mathsf{s}x \not\approx \mathsf{s}y \lor x \approx y\}$ because the maximal literals are both negations.

$$\mathsf{s}(x_1) \not\approx 0 \qquad \mathsf{s}(x_2) \not\approx \mathsf{s}(y_2) \lor x_2 \approx y_2$$

The saturated set does not contain the empty clause, hence we conclude it's satisfiability.

Inst-Gen-Eq

Here the general approach for proving unsatisfiability of a set of clauses is the same as with `Inst-Gen`. We approximate the satisfiability of the set of clauses with a `SAT`- or `SMT`-solver and in the case of satisfiability we use the propositional model for selecting literals. Then we searched for clashing selected literals to derive instances of contained clauses, that would refine the propositional approximation.

Definition 4.29. The *unit superposition calculus* includes

- unit paramodulation (UP)

  $$\frac{s \approx t \quad L[s']}{(L[t])\,\sigma}\ (UP)$$

  where $\sigma = \mathrm{mgu}(s, s')$ is defined, $s' \notin \mathcal{V}$, $s(\sigma)\theta \succ t(\sigma)er\theta$ for some grounding substitution θ;

- unit superposition (US_-, US_+)

  $$\frac{s \approx t \quad u[s'] \not\approx v}{(u[t] \not\approx v)\,\sigma}\ (US_-) \qquad\qquad \frac{s \approx t \quad u[s'] \approx v}{(u[t] \approx v)\,\sigma}\ (US_+)$$

  where $\sigma = \mathrm{mgu}(s, s')$ is defined, $s' \notin \mathcal{V}$, $s\sigma\theta \succ t\sigma\theta$, $u[s']\sigma\theta \succ v\sigma\theta$

  for some grounding substitution θ;

- unit equality resolution ($UR_{\approx}$), and unit resolution (UR)

$$\frac{s \not\approx t}{\Box}\ (UR_{\approx}) \qquad\qquad \frac{A \quad \neg B}{\Box}\ (UR)$$

where s and t (A and B respectively) are unifiable.

At a first glance `Inst-Gen-Eq` is expected to behave similar to the application of Superposition. But actually it shares a disadvantage with `Inst-Gen` (see Example 4.26) as we can see in the following example.

Example 4.30. Let $S = \{\, \mathsf{s}(x_1) \not\approx \mathsf{s}(y_1) \vee x_1 \approx y_1,\, \mathsf{s}(x_2) \not\approx x_2 \,\}$. We start with $S_0 = S$, construct the `SMT`-encoding for $S_{0_\perp}$ and select one literal per clause from S_0 into L_1. The selection is unambiguous by any model. We easily derive the empty clause from the set of selected literals L_0 by first applying unit superposition first and unit equality resolution afterwards.

$$\begin{aligned} S_0 &= \{\, \mathsf{s}(x_1) \not\approx \mathsf{s}(y_1) \vee x_1 \approx y_1,\, \mathsf{s}(x_2) \not\approx x_2 \,\} \\ S_{0_\perp} &= \{\, \mathsf{s}(\mathsf{c}_\perp) \not\approx \mathsf{s}(\mathsf{c}_\perp) \vee \mathsf{c}_\perp \approx \mathsf{c}_\perp,\, \mathsf{s}(\mathsf{c}_\perp) \not\approx \mathsf{c}_\perp \,\} \\ L_0 &= \{x_1 \approx y_1, \mathsf{s}(x_2) \not\approx x_2\} \end{aligned}$$

$$\dfrac{\dfrac{x_1 \approx y_1 \quad [\mathsf{s}(x_2)] \not\approx x_2}{y_1 \not\approx x_2}\ \sigma_1 = \{x_1 \mapsto \mathsf{s}(x_2)\}}{\Box}\ \{y_1 \mapsto x_2\}$$

Since the clauses just contributed to the first step we instantiate $\mathcal{C}_3' = \mathcal{C}_1 \cdot \sigma_1$. For convenience we rename the variables $\mathcal{C}_3 = \mathcal{C}_3' \cdot \rho$. We ignore $\mathcal{C}_2$ which would just yield a variant of itself.

$$\begin{aligned} \mathcal{C}_3 &= \mathsf{s}(\mathsf{s}(x_3)) \not\approx \mathsf{s}(y_3) \vee \mathsf{s}(x_3) \approx y_3 \\ \mathcal{C}_{i+3} &= \mathsf{s}^{i+2}(x_{i+3}) \approx \mathsf{s}(y_{i+3}) \vee \mathsf{s}^{i+1}(x_{i+3}) \approx y_{i+3} && i = 0 \text{ (base case)} \end{aligned}$$

We now show by induction that `Inst-Gen-Eq` yields an infinite sequence of distinct clauses $\mathcal{C}_{i+3}$ for $i \in \mathbb{N}$. The base case $i = 0$ is already covered. We assume for simplicity and without loss of generality that the literal $\mathsf{s}^{i+1}(x_{i+3}) \approx y_{i+3}$ will never be selected.[4] We then can derive the contradiction from the selected literals of the first and the newest

[4] Otherwise we quickly derive the unit clause $\mathsf{s}^{i+1}(x_{i+3}) \not\approx x_2$ that prohibits the selection.

clause and instantiate the first clause with the new unifier σ_{i+2}.

$$\begin{aligned} S_{i+1} &= S_i \mathbin{\dot\cup} \{\, \mathsf{s}^{i+2}(x_{i+3}) \not\approx \mathsf{s}(y_{i+3}) \lor \mathsf{s}^{i+1}(x_{i+3}) \approx y_{i+3} \,\} && i \geq 0 \text{ (IH)} \\ S_{(i+1)_\bot} &= S_{0_\bot} \mathbin{\dot\cup} \{\, \mathsf{s}^{i+2}(\mathsf{c}_\bot) \not\approx \mathsf{s}(\mathsf{c}_\bot) \lor \mathsf{s}^{i+1}(\mathsf{c}_\bot) \approx \mathsf{c}_\bot \,\} \\ L_{i+1} &= L_i \mathbin{\dot\cup} \{\, \mathsf{s}^{i+2}(x_{i+3})) \not\approx \mathsf{s}(y_{i+3}) \,\} \end{aligned}$$

$$\cfrac{x_1 \approx y_1 \quad [\mathsf{s}^{i+2}(x_{i+3})] \not\approx \mathsf{s}(y_{i+3})}{\cfrac{y_1 \not\approx \mathsf{s}(y_{i+3})}{\square}\ \{y_1 \mapsto \mathsf{s}(y_{i+3})\}}\ \sigma_{i+2} = \{x_1 \mapsto \mathsf{s}^{i+2}(x_{i+3})\}$$

$$\begin{aligned} \mathcal{C}'_{(i+1)+3} &= \mathcal{C}_1 \cdot \sigma_{i+2} = \mathsf{s}(\mathsf{s}^{i+2}(x_{i+3})) \not\approx \mathsf{s}(y_{i+3}) \lor \mathsf{s}^{i+2}(x_{i+4}) \approx y_{i+3} \\ \mathcal{C}_{(i+1)+3} &= \mathsf{s}^{(i+1)+2}(x_{i+4}) \not\approx \mathsf{s}(y_{i+4}) \lor \mathsf{s}^{(i+1)+1}(x_{i+4}) \approx y_{i+4} && \text{(step case)} \end{aligned}$$

$$\begin{aligned} S_0 &= \{\, \boxed{\mathsf{s}(x') \not\approx \mathsf{s}(y') \lor x' \approx y'}, \mathsf{s}(x) \not\approx 0 \,\} \\ S_{0_\bot} &= \{\, \mathsf{s}(\mathsf{c}_\bot) \not\approx \mathsf{s}(\mathsf{c}_\bot) \lor \mathsf{c}_\bot \approx \mathsf{c}_\bot, \mathsf{s}(\mathsf{c}_\bot) \not\approx 0 \,\} \end{aligned}$$

$$\cfrac{x' \approx y' \quad \mathsf{s}(x) \not\approx x}{\cfrac{y' \not\approx x}{\square}\ y' \mapsto 0}\ x' \mapsto \mathsf{s}(x)$$

$$\begin{aligned} S_1 &= S_0 \mathbin{\dot\cup} \{ \boxed{\mathsf{s}(\mathsf{s}(x)) \not\approx \mathsf{s}(0) \lor \mathsf{s}(x) \approx 0} \} \\ S_{i+1} &= S_i \mathbin{\dot\cup} \{\, \mathsf{s}^{i+2}(x) \not\approx \mathsf{s}^{i+1}(0) \lor \mathsf{s}^{i+1}(x) \approx \mathsf{s}^{i}(0) \,\} \\ S_{(i+1)_\bot} &= S_i \mathbin{\dot\cup} \{\, \mathsf{s}^{i+2}(\mathsf{c}_\bot) \not\approx \mathsf{s}^{i+1}(0) \lor \mathsf{s}^{i+1}(\mathsf{c}_\bot) \approx \mathsf{s}^{i}(0) \,\} \end{aligned}$$

$$\cfrac{x' \approx y' \quad [\mathsf{s}^{i+2}(x)] \not\approx \mathsf{s}^{i+1}(0)}{\cfrac{y' \not\approx \mathsf{s}^{i+1}(0)}{\square}\ y' \mapsto \mathsf{s}^{i+1}(0)}\ x' \mapsto \mathsf{s}^{i+2}(x)$$

$$S_{i+2} = S_{i+1} \mathbin{\dot\cup} \{ \boxed{\mathsf{s}^{i+3}(x) \not\approx \mathsf{s}^{i+2}(0) \lor \mathsf{s}^{i+2}(x) \approx \mathsf{s}^{i+1}(0)} \}$$

4.5 Roundup of Calculi

Calculus	Equality	Exit condition	Implementations
Gilmore	axioms	$S_i \equiv \bigvee \square$	Gilmore
Resolution	axioms	$\square \in S_i$	
`Inst-Gen`	axioms	$\neg\texttt{SAT}(S_i\bot)$	iProver
Superposition	derivation rules	$\square \in S_i$	Vampire
`Inst-Gen-Eq`	derivation rules	$\neg\texttt{SMT}(S_i\bot)$	iProverEq

5 Completeness of `Inst-Gen-Eq`

An empty calculus, i.e. a calculus without any rules, is obviously sound, but not complete.

We want to restrict our calculus as far as possible, such that there is a chance of termination

As we have already seen in examples `Inst-Gen-Eq` may terminate for a set of clauses without finding a unsatisfiable set of ground instances. We have claimed that in this case the set of clauses is satisfiable.

We present the completeness proof from [7] in a more unnested form.

5.1 Unit paramodulation

Definition 5.1. A closure is a pair of a clause C and a substitution σ, conveniently written as $C \cdot \sigma$. Two closures $C \cdot \sigma = D \cdot \tau$ are the same if C is a variant of D and $C\sigma$ is a variant of $D\tau$. A closure $C \cdot \sigma$ represents a clause $C\sigma$, i.e. the result of applying substitution σ to C. A ground closure represents a ground clause.

For the following definition we assume $\succ_{\mathtt{gr}}$ as a a total, well-founded and monotone extension from a total simplification ordering on ground terms to ground clauses [15].

Definition 5.2. We define an order $\succ_{\mathtt{L}}$ on ground closures of literals as an arbitrary total well-founded extension of $\succ_{\mathtt{gr}}$ such that $L \cdot \sigma \succ_{\mathtt{L}} L' \cdot \sigma'$ whenever $L\sigma \succ_{\mathtt{gr}} L'\sigma'$.

We define an order $\succ_{\mathtt{C}}$ on ground closures as an arbitrary total well-founded extension of $\succ'_{\mathtt{C}}$ — an inherently well-founded order defined as extension of $\succ_{\mathtt{gr}}$ such that $C \cdot \tau \succ'_{\mathtt{C}} D \cdot \rho$ whenever $C\tau \succ_{\mathtt{gr}} D\rho$ or $C\tau = D\rho$ and $C\theta = D$ for an proper instantiator θ.

Lemma 5.3. *A well-founded and total order on general ground terms always exists.*

Definition 5.4 (Order on literals)**.** We extend a well-founded and total order $\succ$ on general ground terms, i.e general atoms to a well-founded proper order $\succ_{\mathtt{L}}$ on literals such that for all atoms A and B with $A \succ B$ the relations $A \succ_{\mathtt{L}} B$, $\neg A \succ_{\mathtt{L}} \neg B$ and $\neg A \succ_{\mathtt{L}} A$ hold. A (non-ground) literal L is *(strictly) maximal* if there exists a ground substitution τ such for no other literal L' the relation $L'\tau \succ L\tau$ (strictly: $\succeq$) holds. We write $\succ_{gr}$ to suggest the existence of such a ground substitution τ.

Lemma 5.5. *A total simplification order over ground terms always exists [15].*

Definition 5.6. We assume $\succ_{\mathtt{gr}}$, $\succ_{\mathtt{L}}$, $\succ_{\mathtt{C}}$ as total, well-founded, and monotone extensions of a total simplification order over ground terms to ground (literal) clauses and ground (literal) closures such that the following properties hold

$$\begin{aligned} s\sigma \not\approx t\sigma &\succ_{\mathbf{gr}} s\sigma \approx t\sigma \\ L\sigma \lor C\sigma &\succ_{\mathbf{gr}} L\sigma \end{aligned}$$

$$\begin{aligned} L\sigma \succ_{\mathbf{gr}} L'\sigma' \quad &\Rightarrow \quad L \cdot \sigma \succ_{\mathtt{L}} L' \cdot \sigma' \\ \left.\begin{matrix} C\tau \succ_{\mathbf{gr}} D\rho \\ \text{or} \\ C\tau = D\rho, C\theta = D \end{matrix}\right\} \quad &\Rightarrow \quad C \cdot \tau \succ_{\mathtt{C}} D \cdot \rho \end{aligned}$$

for arbitrary terms s, t, literals L, L', clauses C, D, and ground substitutions σ, τ, ρ where θ is a proper instantiator. Note that the order on unit closures is slightly different than on literal closures, e.g.:

$$\begin{aligned} (\mathsf{f}(x) \approx x) \cdot \{x \mapsto \mathsf{a}\} &\succ_{\mathtt{C}} (\mathsf{f}(\mathsf{a}) \approx \mathsf{a}) \cdot \emptyset \qquad\qquad \theta = \{x \mapsto \mathsf{a}\} \\ (\mathsf{f}(x) \approx x) \cdot \{x \mapsto \mathsf{a}\} &\not\succ_{\mathtt{L}} (\mathsf{f}(\mathsf{a}) \approx \mathsf{a}) \cdot \emptyset \end{aligned}$$

Definition 5.7 (Unit paramodulation [7]).

$$\frac{(\ell \approx r) \cdot \sigma \quad L[\ell'] \cdot \sigma'}{L[r]\theta \cdot \rho}\ \theta \qquad\qquad \frac{(s \not\approx t) \cdot \tau}{\square}\ \mu$$

where

- $\ell\sigma \succ_{\mathbf{gr}} r\sigma$, $\theta = \mathrm{mgu}(\ell, \ell')$, $\ell\sigma = \ell'\sigma' = \ell'\theta\rho$, $\ell' \notin \mathcal{V}$
- $s\tau = t\tau$, $\mu = \mathrm{mgu}(s, t)$

Example 5.8. Let $\mathsf{f}(s, t, u) \succ_{\mathbf{gr}} \mathsf{g}(s')$ for all ground terms r, s, t, s'.

$$\frac{\left(\overbrace{\mathsf{f}(x, y, \mathsf{c})}^{\ell} \approx \overbrace{\mathsf{g}(x)}^{r}\right) \cdot \sigma \qquad \left(\mathsf{h}(\overbrace{\boxed{\mathsf{f}(x', \mathsf{h}(y'), z')}}^{\ell'}) \not\approx \mathsf{g}(x')\right) \cdot \sigma'}{\left(\mathsf{h}(\underbrace{\boxed{\mathsf{g}(x')}}_{r\theta}) \not\approx \mathsf{g}(x')\right) \cdot \rho}\ \theta$$

$$\begin{aligned} \sigma &= \{x \mapsto \mathsf{a}, y \mapsto \mathsf{h}(\mathsf{b})\} & \sigma' &= \{x' \mapsto \mathsf{a}, y' \mapsto \mathsf{b}, z' \mapsto \mathsf{c}\} \\ \theta &= \{x \mapsto x', y \mapsto \mathsf{h}(y'), z' \mapsto \mathsf{c}\} & \rho &= \{x' \mapsto \mathsf{a}, y' \mapsto \mathsf{b}\} \\ \ell'\theta &= \mathsf{f}(x', \mathsf{h}(y'), \mathsf{c}) & \ell'\theta\rho &= \mathsf{f}(\mathsf{a}, \mathsf{h}(\mathsf{b}), \mathsf{c}) = \ell\sigma = \ell'\sigma' \end{aligned}$$

Lemma 5.9. *Let R be a ground rewrite system and UP is applicable to $(l \approx r) \cdot \sigma, L[l'] \cdot \sigma'$ with conclusion $L[r]\theta \cdot \rho$. If σ, σ' are irreducible w.r.t. R then ρ is irreducible w.r.t. R.*

Proof. Assume otherwise. Hence there is a $x \in \mathcal{Vars}(l'\theta)$ such that $x\theta\rho$ is reducible by R. □

Example 5.10. The set of literal closures $\{\, (\mathsf{f}(x) \approx \mathsf{b}) \cdot \{x \mapsto \mathsf{a}\}, \mathsf{a} \approx \mathsf{b}, \mathsf{f}(\mathsf{b}) \not\approx \mathsf{b} \,\}$ is inconsistent, but the empty clause is not derivable if $\mathsf{a} \succ_{\mathtt{gr}} \mathsf{b}$.

Definition 5.11 (UP-Redundancy)**.** Let $\mathcal{L}$ be a set of literal closures. We define

- $\mathrm{irred}_R(\mathcal{L}) = \{\, L \cdot \sigma \in \mathcal{L} \mid \sigma \text{ is irreducible w.r.t. } R \,\}$

  for an arbitrary ground rewrite system R
- $\mathcal{L}_{L\cdot\sigma\succ_{\mathrm{L}}} = \{\, L' \cdot \sigma' \in \mathcal{L} \mid L \cdot \sigma \succ_{\mathrm{L}} L' \cdot \sigma' \,\}$.
- Literal closure $L \cdot \sigma$ is UP-redundant in $\mathcal{L}$ if

  $$R \cup \mathit{irred}_R(\mathcal{L}_{L\cdot\sigma\succ_{\mathrm{L}}}) \vDash L\sigma$$

  for every ground rewrite system R

  oriented by $\succ_{\mathtt{gr}}$ where σ is irreducible w.r.t. R.
- $\mathcal{R}_{UP}(\mathcal{L})$ denotes the set of all UP-redundant closures in $\mathcal{L}$.

Definition 5.12 (UP-Saturation)**.** The UP-saturation process is a sequence $\{\mathcal{L}_i\}_{i=0}^{\infty}$ where $\mathcal{L}_{i+1}$ is constructed from $\mathcal{L}_i$ by removing redundant literal closures in $\mathcal{L}_i$ or by adding inferences of $\mathcal{L}_i$ until saturation.

$$\mathcal{L}_{i+1} = \begin{cases} \mathcal{L}_i \backslash L \cdot \sigma & \text{if} \quad R \cup \mathrm{irred}_R(\mathcal{L}_{i,L\cdot\sigma\succ_{\mathrm{L}}}) \vDash L\sigma \\[2ex] \mathcal{L}_i \cup \square & \text{if} \quad \begin{cases} (s \not\approx t) \cdot \tau \in \mathcal{L}_i \\ s\tau = t\tau,\ \mu = \mathrm{mgu}(s,t) \end{cases} \\[2ex] \mathcal{L}_i \cup L[r]\theta \cdot \rho & \text{if} \quad \begin{cases} (\ell \approx r) \cdot \sigma,\ L[\ell'] \cdot \sigma' \in \mathcal{L}_i \\ \ell\sigma \succ_{\mathtt{gr}} r\sigma,\ \theta = \mathrm{mgu}(\ell,\ell'), \\ \ell' \notin \mathcal{V},\ \ell\sigma = \ell'\sigma' = \ell'\theta\rho, \end{cases} \\[2ex] \mathcal{L}_i & \qquad \text{otherwise} \end{cases}$$

The set of persistent closures $\mathcal{L}^{\infty}$ is the lower limit of $\mathcal{L}_i$.

Definition 5.13 (UP-Fairness)**.** The UP-saturation process is UP-fair if for every UP-inference with premises in $\mathcal{L}^{\infty}$ the conclusion is UP-redundant w.r.t. $\mathcal{L}_j$ for some j.

Definition 5.14. For a set of literals $\mathcal{L}$ we define the saturated set of literal closures $\mathcal{L}^{sat} = \mathcal{L}^{\infty} \backslash \mathcal{R}_{UP}(\mathcal{L}^{\infty})$ for some UP-saturation process $\{\mathcal{L}_i\}_{i=0}^{\infty}$ with $\mathcal{L}_0 = \mathcal{L}$.

Lemma 5.15. *The set $\mathcal{L}^{sat}$ is unique because for any two UP-fair saturation processes $\{\mathcal{L}_i\}_{i=0}^{\infty}$ and $\{\mathcal{L}'_i\}_{i=0}^{\infty}$ with $\mathcal{L}_0 = \mathcal{L}'_0$ we have*

$$\mathcal{L}^{\infty} \backslash \mathcal{R}_{UP}(\mathcal{L}^{\infty}) = \mathcal{L}'^{\infty} \backslash \mathcal{R}_{UP}(\mathcal{L}'^{\infty})$$

Definition 5.16 (Inst-Redundancy)**.** Let S be a set of clauses.

- A ground closure C is Inst-redundant in S if for some k
  - $C'_i \in S$, $C_i = C'_i \cdot \sigma'_i$, $C \succ_{\mathsf{C}} C_i$ for $i \in 1 \ldots k$
  - such that $C_1, \ldots, C_k \models C$
- A (possible non-ground) clause C is called Inst-redundant in S

  if each ground closure $C \cdot \sigma$ is Inst-redundant in S.
- $R_{Inst}(S)$ denotes the set of all Inst-redundant clauses in S.

Example 5.17. $S = \{\, \mathsf{f}(x) \approx x,\ \mathsf{f}(\mathsf{a}) \approx \mathsf{a},\ \mathsf{f}(\mathsf{f}(x)) \approx \mathsf{f}(x)$
$R_{Inst}(S) = \{\, \mathsf{f}(\mathsf{f}(x)) \approx \mathsf{f}(x) \,\}$

Definition 5.18 (S-Relevance)**.** Let S be a set of clauses S, let $I_\perp$ be a model of $S\perp$.

- A selection function sel maps clauses to literals such that

$$\mathrm{sel}(C) \in C \qquad\qquad I_\perp \models \mathrm{sel}(C)\perp$$

- The set of S-relevant literal closures

$$\mathcal{L}_S = \left\{ L \cdot \sigma \mid \begin{array}{l} L \vee C \in S,\ L = \mathrm{sel}(L \vee C) \\ (L \vee C) \cdot \sigma \text{ is not Inst-redundant in S,} \end{array} \right\}$$

Definition 5.19 (Inst-Saturation)**.** Let $\mathcal{L}_S^{sat}$ denote the saturation process of $\mathcal{L}_S$. Then a set of clauses S is Inst-saturated w.r.t. a selection function sel, if $\mathcal{L}_S^{sat}$ does not contain the empty literal clause.

Theorem 5.20. *If a set of clauses S is Inst-saturated, and $S\perp$ is satisfiable, then S is also satisfiable.*

Proof. We assume that S is not satisfiable.

1. We construct a candidate model $\mathcal{I}$ in Definition 5.21 on the next page.
2. We can show that $\mathcal{I}$ is a model by Lemma 5.26 on page 39.

   That contradicts our assumption.

We discard our assumption and conclude that S is satisfiable. □

Definition 5.21 (Candidate Model Construction)**.** Let S be an Inst-saturated set of clauses, i.e. $\Box \notin \mathcal{L}_S^{sat}$, $\mathrm{SAT}(S\bot)$.

Let $L = L' \cdot \sigma \in \mathcal{L}_S^{sat}$. We define inductively:

- $I_L = \{\, \epsilon_M \mid L \succ_{\mathtt{L}} M \,\}$ IH: ϵ_M is defined for any $M \mid L \succ_{\mathtt{L}} M$
- $R_L = \{s \to t \mid s \approx t \in I_L, s \succ_{\mathtt{gr}} t\}$
- $\epsilon_L = \begin{cases} \emptyset & \text{if } L'\sigma \text{ reducible by } R_L \\ \emptyset & \text{if } I_L \vDash L'\sigma \text{ or } I_L \vDash \overline{L'}\sigma \quad \text{(defined)} \\ \{L'\sigma\} & \text{otherwise} \quad \text{(productive)} \end{cases}$
- $R_S = \bigcup_{L \in \mathcal{L}_S^{sat}} R_L$ R_S is convergent and interreduced
- $I_S = \bigcup_{L \in \mathcal{L}_S^{sat}} \epsilon_L$ I_S is consistent, $L\sigma \in I_S$ is irreducible by R_S

Let $\mathcal{I}$ be an arbitrary consistent extension of I_S
in all the following lemmata.

Lemma 5.22. *If any $L \cdot \sigma \in \mathcal{L}_S$, irreducible by R_S exists with $\mathcal{I} \nvDash L\sigma$ then there is a $L' \cdot \sigma' \in \mathrm{irred}_{R_S}(\mathcal{L}_S^{sat})$ with $\mathcal{I} \nvDash L'\sigma'$.*

Proof. We have two cases

- If $L \cdot \sigma$ is not UP-redundant in $\mathcal{L}_S^{sat}$, then $L' \cdot \sigma' = L \cdot \sigma$. ✓
- If $L \cdot \sigma$ is UP-redundant in $\mathcal{L}_S^{sat}$. By construction σ is irreducible by R_S. Then we have

$$R_S \cup \mathrm{irred}_{R_S}(\{L' \cdot \sigma' \in \mathcal{L}_S^{sat} \mid L \cdot \sigma \succ_{\mathtt{L}} L' \cdot \sigma'\}) \models L\sigma$$

  with $\mathcal{I} \nvDash L'\sigma'$. ✓

□

Lemma 5.23. *Whenever*

$$M \cdot \tau = \min_{\succ_{\mathtt{L}}} \Big\{\, L' \cdot \tau' \mid L' \cdot \sigma' \in \mathrm{irred}_{R_S}(\mathcal{L}_S^{sat}),\ L'\sigma' \textit{ false in } \mathcal{I} \,\Big\}$$

is defined, then $M \cdot \tau$ is irreducible by R_S.

Proof. Assume $M \cdot \tau$ is reducible by $(\ell \to r) \in R_S$
and $(\ell \to r)$ is produced by $(\ell' \approx r') \cdot \rho \in \mathcal{L}_S^{sat}$.

Now UP-inference is applicable because τ is irreducible by R_S,

$$\frac{(\ell' \approx r') \cdot \rho \quad M[\ell''] \cdot \tau}{M[r']\theta \cdot \mu}\ UP$$

μ is irreducible by R_S, and $M[r']\theta\mu$ is false in $\mathcal{I}$. . . .

We have two cases

- If $M[r']\theta \cdot \mu$ is not UP-redundant in $\mathcal{L}_S^{sat}$ then $M[r']\theta \cdot \mu \in \mathcal{L}_S^{sat}$.

  Now $M \cdot \tau \succ_{\mathtt{L}} M[r']\theta \cdot \mu \in \mathrm{irred}_{R_S}(\mathcal{L}_S^{sat})$
  contradicts minimality of $M \cdot \tau$.

- If $M[r']\theta \cdot \mu$ is UP-redundant in $\mathcal{L}_S^{sat}$ then

  $$R_S \cup \mathrm{irred}_{R_S}\{M' \cdot \tau' \in \mathcal{L}_S^{sat} \mid M[r']\theta \cdot \mu \succ_{\mathtt{L}} M'\tau'\} \models M[r']\theta\mu$$

  Hence there is $M' \cdot \tau' \in \mathcal{L}_S^{sat}$ false in $\mathcal{I}$ such that $M \cdot \tau \succ_{\mathtt{L}} M[r']\theta \cdot \mu \succ_{\mathtt{L}} M' \cdot \tau'$, $M' \cdot \tau'$ contradicts minimality of $M \cdot \tau$.

Hence $M \cdot \tau$ is irreducible by R_S. □ □

Lemma 5.24. *Let $M \cdot \tau \in \mathcal{L}_S^{sat}$, irreducible by R_S, and defined (not productive). From $\mathcal{I} \not\models M\tau$ follows that M is not an equation $(s \approx t)$.*

Proof. Assume $M = (s \approx t)$. Then we have

- $I_{M\cdot\tau} \models (s \not\approx t)\tau$
- All literals in $I_{M\cdot\tau}$ are irreducible by $R_{M\cdot\tau}$
- $s\tau$ and $t\tau$ are irreducible by $R_{M\cdot\tau}$
- $R_{M\cdot\tau}$ is a convergent term rewrite system

Hence $(s \not\approx t)\tau \in I_{M\cdot\tau}$ is produced to $I_{M\cdot\tau}$ by some $(s' \not\approx t') \cdot \tau'$, but $(s' \not\approx t')\tau' \succ_{\mathtt{gr}} (s \approx t)\tau$ and $(s' \not\approx t') \cdot \tau' \succ_{\mathtt{L}} M \cdot \tau$.

□

Lemma 5.25. *Let $M \cdot \tau \in \mathcal{L}_S^{sat}$, irreducible by R_S, and defined (not productive). From $\mathcal{I} \not\models M\tau$ follows that M is not an inequation $(s \not\approx t)$.*

Proof. Assume $M \cdot \tau$ is inequation $(s \not\approx t) \cdot \tau$. We have

- $I_{M\cdot\tau} \models (s \approx t)\tau$
- $s\tau$ and $t\tau$ are irreducible by $R_{M\cdot\tau}$

Hence $s\tau = t\tau$ and equality resolution is applicable.
Contradiction to $\Box \notin \mathcal{L}_S^{sat}$.

□

Lemma 5.26. *$\mathcal{I}$ is a model for all ground instances of S*

Proof. Assume $\mathcal{I}$ is not a model.
Hence a minimal ground closure $D = \min_{\succ_{\mathtt{C}}}\{\, C' \cdot \sigma \mid C' \in S, \mathcal{I} \not\models C'\sigma \,\}$, an instance of a clause in S, false in $\mathcal{I}$, must exist. Further on

- $D = D' \cdot \sigma$ is not Inst-redundant.

  Otherwise by Definition 5.16 there are $D_1, \ldots, D_n \models D$, $D \succ_{\mathsf{C}} D_i$ for all i, and D_j false in $\mathcal{I}$ for one j, which contradicts minimality.

- $x\sigma$ irreducible by R_S for every variable x in D'.

  Otherwise let $(\ell \to r)\tau \in R_L$ and $x\sigma = x\sigma[l\tau]_p$ for some variable x in D'. We define substitution σ' with $x\sigma' = x\sigma[r\tau]_p$ and $y\sigma' = y\sigma$ for $y \neq x$. $D'\sigma'$ is false in $\mathcal{I}$ and $D \succ_{\mathsf{C}} D' \cdot \sigma'$, which contradicts minimality.

Since D is not Inst-redundant in S, we have for some literal L, that $D' = L \vee D''$, $\mathrm{sel}(D') = L$, $L \cdot \sigma \in \mathcal{L}_S$, $L\sigma$ is false in $\mathcal{I}$.

Hence the following literal closure

$$M \cdot \tau = \min_{\succ_{\mathsf{L}}} \Big\{ L' \cdot \tau' \mid L' \cdot \sigma' \in \mathrm{irred}_{R_S}(\mathcal{L}_S^{sat}),\, \mathcal{I} \not\models L' \cdot \sigma' \Big\}$$

exists by Lemma 5.22, is irreducible by Lemma 5.23, and is not productive. Since $\mathcal{I} \not\models M \cdot \tau$ the literal M cannot be an equation by Lemma 5.24 or an inequation by Lemma 5.25. We have derived a contradiction from our only assumption.

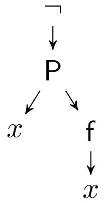
Therefore $\mathcal{I}$ is a model for all instances of S. □

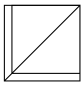
6 Algorithms and Data Structures

6.1 Sets of Clauses

6.1.1 Terms, Atoms, and Literals

The recursive definition of the syntax of terms, atoms and literal suggest a tree structure.





6.1.2 Sets

Clauses are multiset of literals.

6.2 Saturation

In the previous chapter we just applied derivation rules on pairs of clauses chosen haphazardly from the set of clauses to derive and add new clauses until we could conclude unsatisfiability of the set of clauses.

In practice such a human approach may fail for an unsatisfiable set of clauses just because an important clause pairing has been overlooked and infinite inferences can be drawn from a satisfiable subset of the set of clauses. An automated procedure has to notice when all relevant derivations have already been processed.

First will discuss the given clause algorithm in Section 6.2.1 as a saturation process that eventually determines all possible derivate clauses with respect to a given calculus, e.g. ordered resolution.

6.2.1 Given Clause Algorithm

Procedure 3 (Given Clause Algorithm)**.** We start with a set of clauses $S = P \dot\cup U$, where P is a set of *passive* clauses and U is the initially empty set of *active* (or used) clauses.

$(\lightning^?_{\mathsf{true}})$ Whenever we can conclude unsatisfiability of S we exit the procedure with $\neg$`SAT`.

1. If P is empty we exit and return `SAT`.
2. We select a passive clause, i.e. the given clause. $(\lightning^{?}_{\mathsf{true}})$
3. We iterate over the active clauses and check for applicable inference rule for each pair of an active and the given clause to derive additional clauses which we add to the set of passive clauses. $(\lightning^{?}_{\mathsf{true}})$
4. We move the given clause from passive to active and continue with step 1.

In the following examples the given clause is boxed, the active clauses are on the left of the given clause, and the passive clauses are on the right. Each line represents one iteration of our procedure 3. In both examples we start with the unsatisfiable set of clauses $S = \{\, \mathsf{P(a)} \lor \mathsf{Q}(a), \mathsf{P(a)} \lor \neg\mathsf{Q}(y), \neg\mathsf{P}(x) \,\}$.

Ordered Resolution

$(\lightning^{?}_{\mathsf{true}}) \Leftarrow \square \in S_i$. We conclude unsatisfiability of S whenever the empty clause is found (already present or derived). New clauses are disjunctions of instances of an active and the given clause where conflicting literals were removed, in other words the union of the two instances without the contradicting literals.

$$\frac{\mathsf{P}(\mathsf{f}(x), y) \lor \mathcal{C} \qquad \neg\mathsf{P}(x', \mathsf{g}(y')) \lor \mathcal{D}}{(\mathcal{C} \lor \mathcal{D})\sigma} \; \sigma = \{x' \mapsto \mathsf{f}(x), y \mapsto \mathsf{g}(y')\}$$

Example 6.1 (Ordered Resolution)**.** We assume $\mathsf{P(a)} \succ \mathsf{Q(a)}$, underline the maximal literal the given clauses, color conflicts red, and derive the empty clause in the fifth iteration.

$^{1:}\boxed{\underline{\mathsf{P(a)}} \lor \mathsf{Q}(a)}$	$^{2:}\mathsf{P(a)} \lor \neg\mathsf{Q}(y)$	$^{3:}\neg\mathsf{P}(x)$				
$^{1:}\mathsf{P(a)} \lor \mathsf{Q}(a)$	$^{2:}\boxed{\underline{\mathsf{P(a)}} \lor \neg\mathsf{Q}(y)}$	$^{3:}\neg\mathsf{P}(x)$				
$^{1:}\textcolor{red}{\mathsf{P(a)}} \lor \mathsf{Q}(a)$	$^{2:}\textcolor{red}{\mathsf{P(a)}} \lor \neg\mathsf{Q}(y)$	$^{3:}\boxed{\underline{\neg\mathsf{P}(x)}}$	$^{1,3:}\mathsf{Q(a)}$	$^{2,3:}\neg\mathsf{Q}(y)$		
$^{1:}\mathsf{P(a)} \lor \mathsf{Q}(a)$	$^{2:}\mathsf{P(a)} \lor \textcolor{red}{\neg\mathsf{Q}(y)}$	$^{3:}\neg\mathsf{P}(x)$	$^{1,3:}\boxed{\underline{\mathsf{Q(a)}}}$	$^{2,3:}\neg\mathsf{Q}(y)$	$^{2,(1,3):}\mathsf{P(a)}$	
$^{1:}\mathsf{P(a)} \lor \textcolor{red}{\mathsf{Q}(a)}$	$^{2:}\mathsf{P(a)} \lor \neg\mathsf{Q}(y)$	$^{3:}\neg\mathsf{P}(x)$	$^{1,3:}\textcolor{red}{\mathsf{Q(a)}}$	$^{2,3:}\boxed{\underline{\neg\mathsf{Q}(y)}}$	$^{2,(1,3):}_{1,(2,3):}\mathsf{P(a)}$	$^{(1,3),(2,3):}\square$

InstGen

$(\lightning^{?}_{\mathsf{true}}) \Leftarrow \neg\mathtt{SAT}(S_i\bot)$. We conclude unsatisfiability of S whenever (a subset of) $S_i\bot$ — a set of ground instances — is unsatisfiable. We consider only proper instances of an active and the given clause as probable new clauses to extend the set of clauses.

$$\frac{\mathsf{P}(\mathsf{f}(x), y) \lor \mathcal{C} \qquad \neg\mathsf{P}(x', \mathsf{g}(y')) \lor \mathcal{D}}{\mathsf{P}(\mathsf{f}(x), \mathsf{g}(y')) \lor \mathcal{C}\sigma \qquad \neg\mathsf{P}(\mathsf{f}(x), \mathsf{g}(y')) \lor \mathcal{D}\sigma} \; \sigma = \{x' \mapsto \mathsf{f}(x), y \mapsto \mathsf{g}(y')\}$$

Example 6.2 (InstGen)**.** The active and given clauses are already encoded and given to a SAT solver. The basic given clause algorithm would stop and fail after (4) since $S_4\bot$ is still satisfiable and there is no conflict between the underlined selected literal of the given clause and any selected literals of any of the active clauses.

(1)	$^{1:}\boxed{\underline{\mathsf{P(a)}} \lor \mathsf{Q(a)}}$	$^{2:}\mathsf{P(a)} \lor \neg\mathsf{Q}(\mathsf{c}_\bot/y)$	$^{3:}\neg\mathsf{P}(\mathsf{c}_\bot/x)$	
(2)	$^{1:}\underline{\mathsf{P(a)}} \lor \mathsf{Q(a)}$	$^{2:}\boxed{\underline{\mathsf{P(a)}} \lor \neg\mathsf{Q}(\mathsf{c}_\bot/y)}$	$^{3:}\neg\mathsf{P}(\mathsf{c}_\bot/x)$	
(3)	$^{1:}\underline{\mathsf{P(a)}} \lor \mathsf{Q(a)}$	$^{2:}\underline{\mathsf{P(a)}} \lor \neg\mathsf{Q}(\mathsf{c}_\bot/y)$	$^{3:}\boxed{\underline{\neg\mathsf{P}(\mathsf{c}_\bot/x)}}$	$^{1,3:}_{2,3:}\neg\mathsf{P(a)}$
(4)	$^{1:}\mathsf{P(a)} \lor \mathsf{Q(a)}$	$^{2:}\mathsf{P(a)} \lor \neg\mathsf{Q}(\mathsf{c}_\bot/y)$	$^{3:}\underline{\neg\mathsf{P}(\mathsf{c}_\bot/x)}$	$^{1,3:}_{2,3:}\boxed{\underline{\neg\mathsf{P(a)}}}$

But the model did change in (4) and the selected literals of two of the active clauses had to be changed too. Active clauses with changed selected literals have to be moved back to the passive clauses. Then we hit a contradiction of ground instances in (7').

$(4')$	$^{3:}\underline{\neg\mathsf{P}(\mathsf{c}_\bot/x)}$	$^{1,3:}_{2,3:}\boxed{\underline{\neg\mathsf{P(a)}}}$	$^{1:}\mathsf{P(a)} \lor \mathsf{Q(a)}$	$^{2:}\mathsf{P(a)} \lor \neg\mathsf{Q}(\mathsf{c}_\bot/y)$	
$(5')$	$^{3:}\underline{\neg\mathsf{P}(\mathsf{c}_\bot/x)}$	$^{1,3:}_{2,3:}\underline{\neg\mathsf{P(a)}}$	$^{1:}\boxed{\mathsf{P(a)} \lor \underline{\mathsf{Q(a)}}}$	$^{2:}\mathsf{P(a)} \lor \neg\mathsf{Q}(\mathsf{c}_\bot/y)$	
$(6')$	$^{3:}\underline{\neg\mathsf{P}(\mathsf{c}_\bot/x)}$	$^{1,3:}_{2,3:}\underline{\neg\mathsf{P(a)}}$	$^{1:}\mathsf{P(a)} \lor \underline{\mathsf{Q(a)}}$	$^{2:}\boxed{\mathsf{P(a)} \lor \underline{\neg\mathsf{Q}(\mathsf{c}_\bot/y)}}$	$^{2,1:}\mathsf{P(a)} \lor \neg\mathsf{Q(a)}$
$(7')$	$^{3:}\neg\mathsf{P}(\mathsf{c}_\bot/x)$	$^{1,3:}_{2,3:}\neg\mathsf{P(a)}$	$^{1:}\mathsf{P(a)} \lor \mathsf{Q(a)}$	$^{2:}\mathsf{P(a)} \lor \neg\mathsf{Q}(\mathsf{c}_\bot/y)$	$^{2,1:}\boxed{\mathsf{P(a)} \lor \neg\mathsf{Q(a)}}$

6.2.2 Bookkeeping

6.3 Term Indexing

In refutation theorem proving (a lot of) clauses are produced. For each generated clause we may have to find existing variants, generalizations, or instances of it. In a more general view we want to find existing clauses that subsume our clause or are subsumed by a our clause to avoid logical redundancies in our set[1] of clauses. Further we search for existing clauses that contain clashing literals to a literal in our clause. This search may be restricted by a given order (e.g. Superposition) or a propositional model (e.g. `Inst-Gen-Eq`). Last but not least we may search for subterms in literals of existing clauses that matches a term in an equation.

Naively we can just scan through all existing clauses and check each clause for the desired qualities. The workload for processing a generated clause is proportional to the number of existing clauses and the workload for checking a pair of clauses. The latter

[1] At least we might expect that by adding clauses that are syntactical identical to clauses already in the set we do not increase the cardinality of the set.

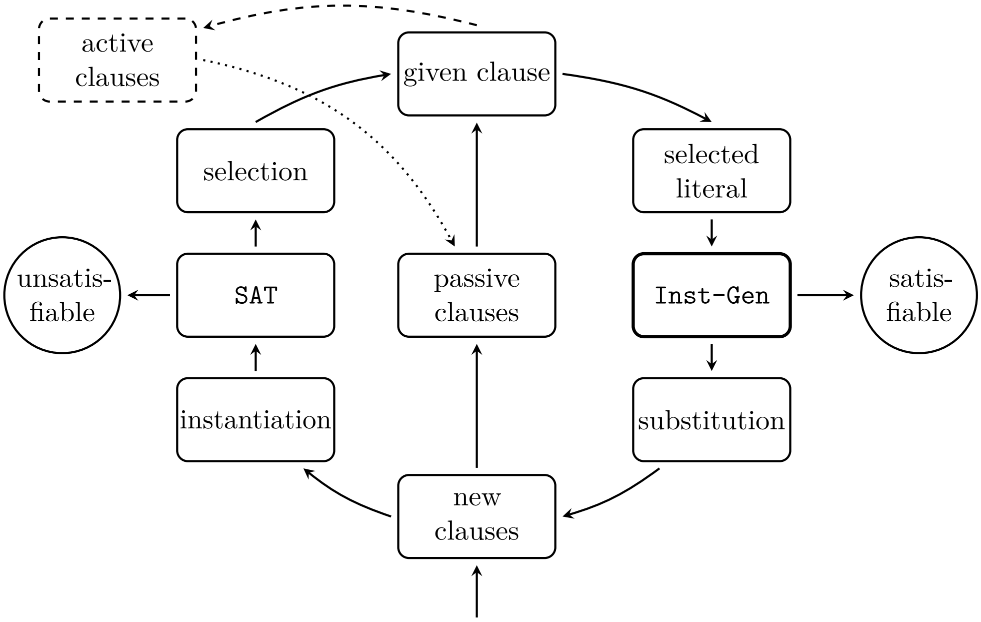


Figure 6.1: Proving loop with `SAT` and `Inst-Gen`

includes unification, which is at least linear to the size of clauses [1], while Robinson's unification algorithm [17] is exponential in the worst case. The complexity for this direct approach of processing n clauses of fixed size (i.e. constant unification costs per pair) is $\mathcal{O}(n^2)$.

6.4 Termindexing samples

6.4.1 Motivation

Example 6.3 (forward subsumption).

$$S = \{{}^{1:}\mathsf{P}(x,y), {}^{2:}\neg\mathsf{P}(\mathsf{a},z)\} \cup \{{}^{3:}\mathsf{P}(\mathsf{a},z')\} \qquad \mathcal{C}_1 \text{ subsumes } \mathcal{C}_3$$

$$\frac{\mathsf{P}(x,y) \quad \neg\mathsf{P}(\mathsf{a},z)}{\square}\ \{x \mapsto \mathsf{a}, y \mapsto z\} \qquad \text{Resolution}$$

$$S\bot = \{\mathsf{P}(\bot,\bot), \neg\mathsf{P}(\mathsf{a},\bot), \mathsf{P}(\mathsf{a},\bot)\} \qquad \text{InstGen / SMT}$$

Goal 1. A sound, refutation complete, and effective calculus.

1. Reduce search space
   - Ordered Resolution, Strategies, ...
   - ... with selection functions for clauses and literals

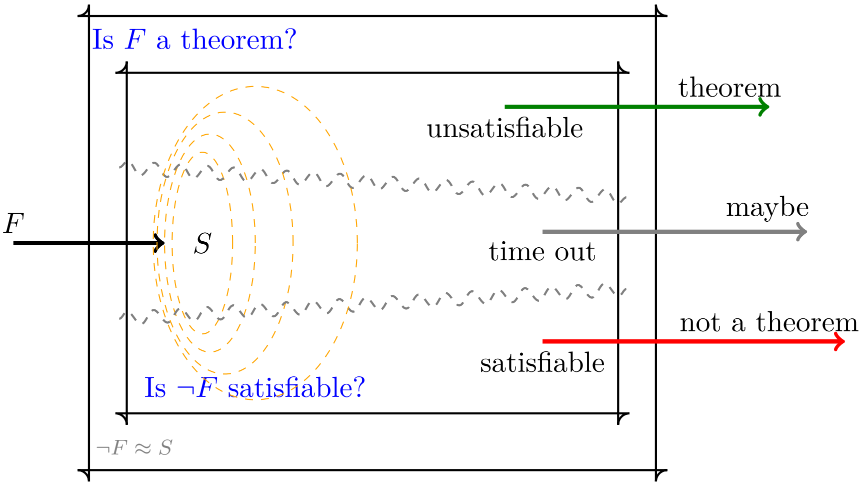
2. Reduce redundancy
   - e.g. discard clauses that are subsumed by other clauses
   - …depending on the calculus

3. Quickly find
   - variants — variant removal
   - instances — backward subsumption
     instance$(s,t) \Leftrightarrow \exists\sigma\ s = t\sigma$
   - generalizations — forward subsumption
     generalization$(s,t) \Leftrightarrow \exists\sigma\ s\sigma = t$
   - unifiable terms — resolution, demodulation
     unifiable$(s,t) \Leftrightarrow \exists\sigma\ s\sigma = t\sigma$

   of a query term in a given set of terms.

Definition 6.4.

$$
\begin{aligned}
\text{variant}(s,t) &\Leftrightarrow \exists\sigma\ s\sigma = t \text{ and } \sigma \text{ is renaming} \\
\text{instance}(s,t) &\Leftrightarrow \exists\sigma\ s = t\sigma && s \text{ is instance of } t \\
\text{generalization}(s,t) &\Leftrightarrow \exists\sigma\ s\sigma = t && s \text{ is generalization of } t \\
\text{unifiable}(s,t) &\Leftrightarrow \exists\sigma\ s\sigma = t\sigma
\end{aligned}
$$



Definition 6.5 (Term Indexing)**.** Term indexing is about data structures and algorithms for fast retrieval of matching terms.

6.4.2 Position

Example 6.6. Variable normalization Variants of terms generate the same position strings

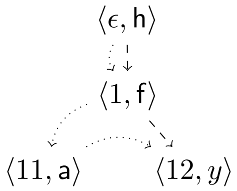
- if variable names are ignored $\mathsf{f}(y,z) \Rightarrow \langle\epsilon,\mathsf{f}\rangle\langle 1,*\rangle\langle 2,*\rangle$
- or normalized $\mathsf{f}(y,z) \Rightarrow \langle\epsilon,\mathsf{f}\rangle\langle 1,x_1\rangle\langle 2,x_2\rangle$
  $\mathsf{f}(y,y) \Rightarrow \langle\epsilon,\mathsf{f}\rangle\langle 1,x_1\rangle\langle 2,x_1\rangle$

In the first case even non-variants of terms generate the same strings.

Definition 6.7. We define the set of *position strings*

$$\mathcal{P}os^{\mathcal{F}}(t) = \begin{cases} \{\langle\epsilon,x\rangle\} & \text{if } t = x \in \mathcal{V} \\ \{\langle\epsilon,f\rangle\} \cup \{\langle ip,s\rangle \mid \langle p,s\rangle \in \mathcal{P}os^{\mathcal{F}}(t_i)\} & \text{if } t = f(t_1,\ldots,t_n) \end{cases}$$

Example 6.8 (Term traversals).



$\langle\epsilon,\mathsf{h}\rangle\langle 1,\mathsf{f}\rangle\langle 12,y\rangle$ path from root to leaf
$\langle\epsilon,\mathsf{h}\rangle\langle 1,\mathsf{f}\rangle\langle 11,\mathsf{a}\rangle\langle 12,y\rangle$ pre-order traversal

Notation 1. We abbreviate

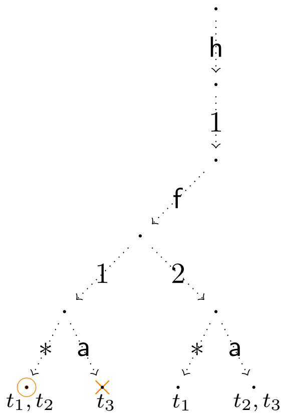
- path strings $\langle\epsilon,\mathsf{h}\rangle\langle 1,\mathsf{f}\rangle\langle 12,*\rangle$ $\mathsf{h.1.f.2.}*$
- and pre-order traversal strings $\langle\epsilon,\mathsf{h}\rangle\langle 1,\mathsf{f}\rangle\langle 11,*\rangle\langle 12,*\rangle$ $\mathsf{h.f.a.}*$
  when the arities of function symbols are fixed.

6.4.3 Path Indexing

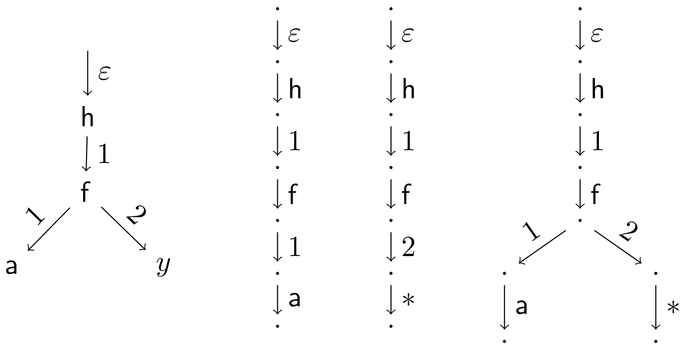
Example 6.9. Build

${}^{t_1:}\mathsf{h}(\mathsf{f}(x,y)), {}^{t_2:}\mathsf{h}(\mathsf{f}(x,\mathsf{a})), {}^{t_3:}\mathsf{h}(\mathsf{f}(\mathsf{a},\mathsf{a}))$

$$\begin{aligned} t_1 &\Rightarrow \{\mathsf{h.1.f.1.}*, \mathsf{h.1.f.2.}*\} \\ t_2 &\Rightarrow \{\mathsf{h.1.f.1.}*, \mathsf{h.1.f.2.a}\} \\ t_3 &\Rightarrow \{\mathsf{h.1.f.1.a}, \mathsf{h.1.f.2a}\} \end{aligned}$$

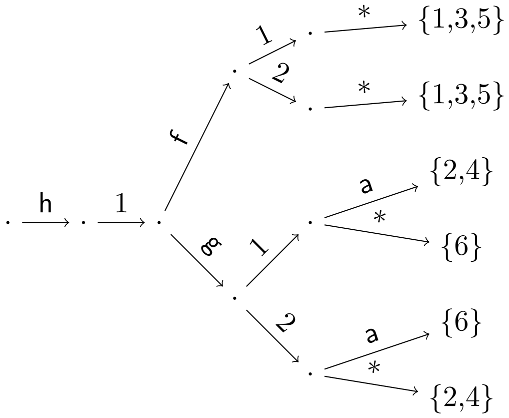


Example 6.10. Path strings The path-strings of $\mathsf{h}(\mathsf{f}(\mathsf{a},y))$ are $\mathsf{h.1.f.1.a}$ and $\mathsf{h.1.f.2.}*$.



Example 6.11 (Prefix Trees)**.**

$$\{{}^{1:}\mathsf{h}(\mathsf{f}(x,x)), {}^{2:}\mathsf{h}(\mathsf{g}(\mathsf{a},x)), {}^{3:}\mathsf{h}(\mathsf{f}(y,z)) {}^{4:}\mathsf{h}(\mathsf{g}(\mathsf{a},y)), {}^{5:}\mathsf{h}(\mathsf{f}(y,x)), {}^{6:}\mathsf{h}(\mathsf{g}(y,a))\}$$



$\mathsf{h}(\mathsf{g}(y,x)) \mapsto \{\mathsf{h.1.g.1.}*, \mathsf{h.1.g.2.}*\}$

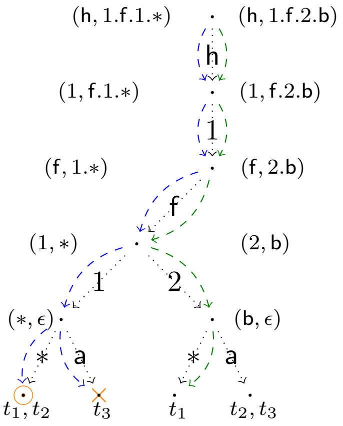
Example 6.12. Retrieve

$${}^{t_1:}\mathsf{h}(\mathsf{f}(x,y)), {}^{t_2:}\mathsf{h}(\mathsf{f}(x,\mathsf{a})), {}^{t_3:}\mathsf{h}(\mathsf{f}(\mathsf{a},\mathsf{a}))$$

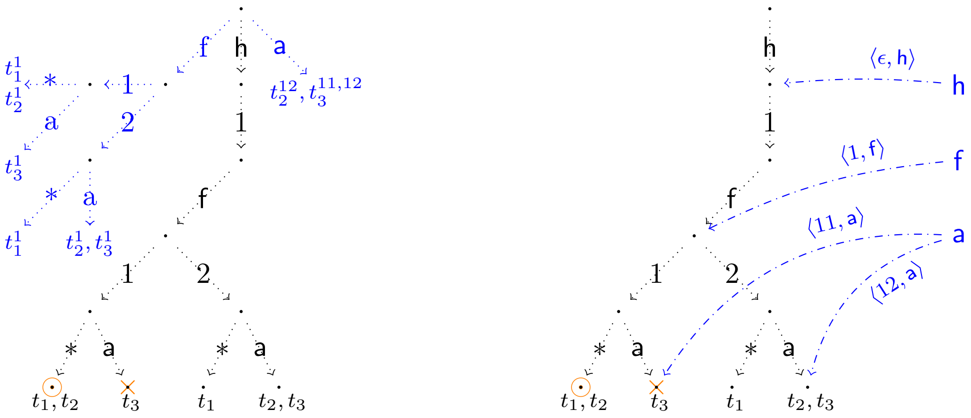
$$\mathsf{h}(\mathsf{f}(z,\mathsf{b})) \Rightarrow \{\mathsf{h.1.f.1.}*, \mathsf{h.1.f.2.b}\}$$

$$u : \mathsf{h}(\mathsf{f}(z,\mathsf{b})) \mapsto \{t_1, t_2, t_3\} \cap \{t_1\}$$
$$i : \mathsf{h}(\mathsf{f}(z,\mathsf{b})) \mapsto \{t_1, t_2, t_3\} \cap \{\}$$
$$g : \mathsf{h}(\mathsf{f}(z,\mathsf{b})) \mapsto \{t_1, t_2\} \cap \{t_1\}$$
$$v : \mathsf{h}(\mathsf{f}(z,\mathsf{b})) \mapsto \{t_1, t_2\} \cap \{\}$$

$$v : \mathsf{h}(\mathsf{f}(z,z)) \mapsto \{t_1, t_2\} \cap \{t_1\}$$

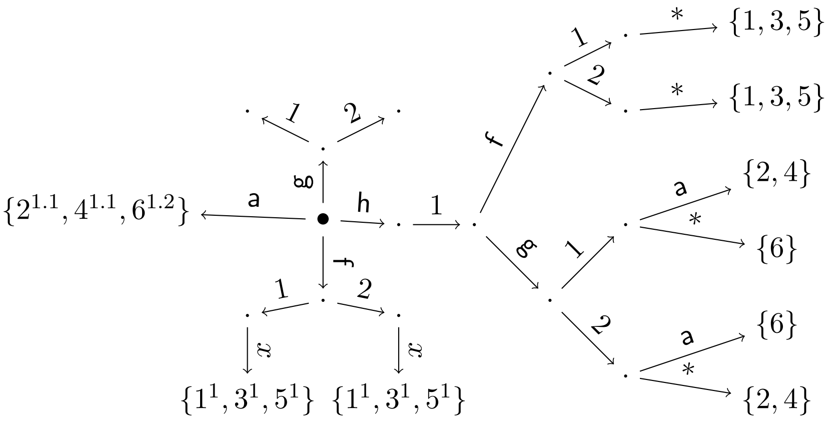


Example 6.13. Demodulation (Subterms) ${}^{t_1:}\mathsf{h}(\mathsf{f}(x,y)), {}^{t_2:}\mathsf{h}(\mathsf{f}(x,\mathsf{a})), {}^{t_3:}\mathsf{h}(\mathsf{f}(\mathsf{a},\mathsf{a})), \ldots, \mathsf{f}(x,\mathsf{a}) \approx x$



Example 6.14. Subterm trees

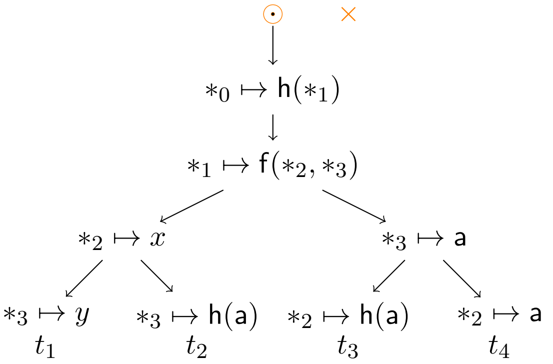
$$\{{}^{1:}\mathsf{h}(\mathsf{f}(x,x)), {}^{2:}\mathsf{h}(\mathsf{g}(\mathsf{a},x)), {}^{3:}\mathsf{h}(\mathsf{f}(y,z)) {}^{4:}\mathsf{h}(\mathsf{g}(\mathsf{a},y)), {}^{5:}\mathsf{h}(\mathsf{f}(y,x)), {}^{6:}\mathsf{h}(\mathsf{g}(y,a))\}$$



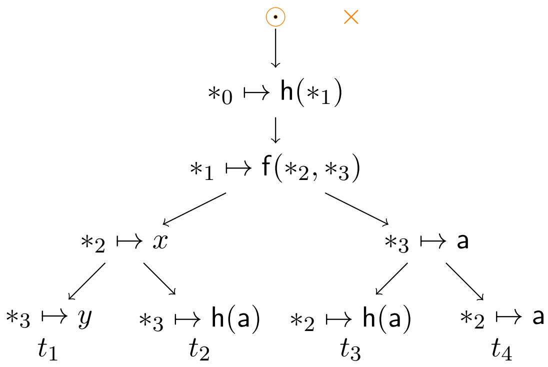
6.4.4 Substitution Tree

Example 6.15. Substitution tree build

${}^{t_1:}\mathsf{h}(\mathsf{f}(x,y)), {}^{t_2:}\mathsf{h}(\mathsf{f}(x,\mathsf{h}(\mathsf{a}))), {}^{t_3:}\mathsf{h}(\mathsf{f}(\mathsf{h}(\mathsf{a}),\mathsf{a})), {}^{t_4:}\mathsf{h}(\mathsf{f}(\mathsf{a},\mathsf{a}))$


$*_0 \mapsto \mathsf{h}(*_1)$
$*_1 \mapsto \mathsf{f}(*_2, *_3)$
$*_2 \mapsto x$
$*_3 \mapsto \mathsf{a}$
$*_3 \mapsto y$
t_1
$*_3 \mapsto \mathsf{h}(\mathsf{a})$
t_2
$*_2 \mapsto \mathsf{h}(\mathsf{a})$
t_3
$*_2 \mapsto \mathsf{a}$
t_4

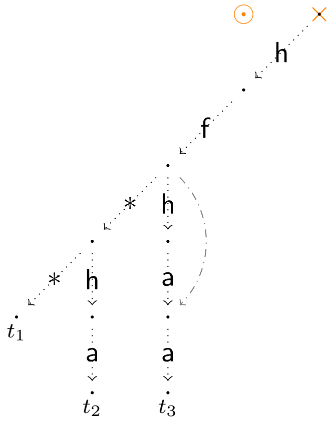
Example 6.16. Substitution tree retrieve


⊙
×
$*_0 \mapsto \mathsf{h}(*_1)$
$*_1 \mapsto \mathsf{f}(*_2, *_3)$
$*_2 \mapsto x$
$*_3 \mapsto \mathsf{a}$
$*_3 \mapsto y$
t_1
$*_3 \mapsto \mathsf{h}(\mathsf{a})$
t_2
$*_2 \mapsto \mathsf{h}(\mathsf{a})$
t_3
$*_2 \mapsto \mathsf{a}$
t_4

$^{t_1:}\mathsf{h}(\mathsf{f}(x,y)),^{t_2:}\mathsf{h}(\mathsf{f}(x,\mathsf{h}(\mathsf{a}))),^{t_3:}\mathsf{h}(\mathsf{f}(\mathsf{h}(\mathsf{a}),\mathsf{a})),^{t_4:}\mathsf{h}(\mathsf{f}(\mathsf{a},\mathsf{a}))$

6.4.5 Discrimination Tree

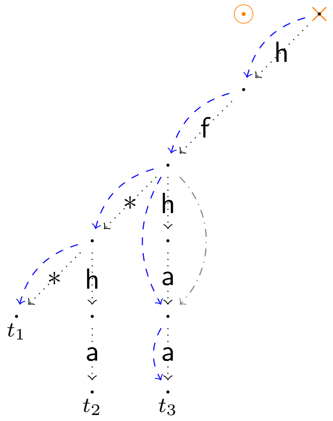
Example 6.17. Build



$^{t_1:}\mathsf{h}(\mathsf{f}(x,y)),^{t_2:}\mathsf{h}(\mathsf{f}(x,\mathsf{h}(\mathsf{a}))),^{t_3:}\mathsf{h}(\mathsf{f}(\mathsf{h}(\mathsf{a}),\mathsf{a}))$

$$t_1 \Rightarrow \mathsf{h.f.}*.*$$
$$t_2 \Rightarrow \mathsf{h.f.}*\mathsf{.h.a}$$
$$t_3 \Rightarrow \mathsf{h.f.h.a.a}$$

Example 6.18. Retrieve

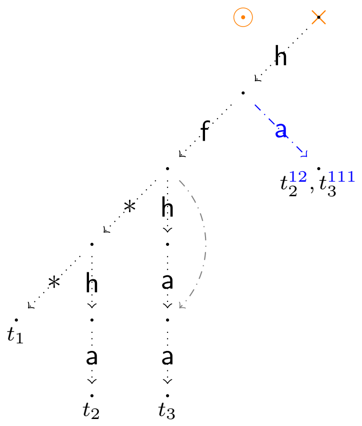


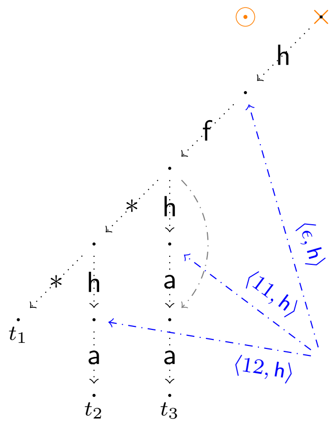
$^{t_1:}\mathsf{h}(\mathsf{f}(x,y)),^{t_2:}\mathsf{h}(\mathsf{f}(x,\mathsf{h}(\mathsf{a}))),^{t_3:}\mathsf{h}(\mathsf{f}(\mathsf{h}(\mathsf{a}),\mathsf{a}))$

$$\mathsf{h}(\mathsf{f}(x',\mathsf{a})) \Rightarrow \mathsf{h.f.}*\mathsf{.a}$$

$$u : \mathsf{h}(\mathsf{f}(x',\mathsf{a})) \mapsto \{t_1,t_3\}$$
$$i : \mathsf{h}(\mathsf{f}(x',\mathsf{a})) \mapsto \{t_3\}$$
$$g : \mathsf{h}(\mathsf{f}(x',\mathsf{a})) \mapsto \{t_1\}$$
$$v : \mathsf{h}(\mathsf{f}(x',\mathsf{a})) \mapsto \{\ \}$$

Example 6.19. Subterms $^{t_1:}\mathsf{h}(\mathsf{f}(x,y)),^{t_2:}\mathsf{h}(\mathsf{f}(x,\mathsf{h}(\mathsf{a}))),^{t_3:}\mathsf{h}(\mathsf{f}(\mathsf{h}(\mathsf{a}),\mathsf{a}))$





6.4.6 Clashing literals

6.4.7 Variants

6.4.8 Instances

6.4.9 Subsumption

7 FLEA

Grau, teurer Freund, ist alle Theorie.
Und grün des Lebens goldner Baum.[1]

Faust 1 (Mephistopheles)
Johann Wolfgang von Goethe

In this chapter we introduce `FLEA` — our **F**irst Order **L**ogic with **E**quality Theorem **A**ttester. It is — as the reader may already suspect — a modest implementation of an instantiation based theorem prover for first order clauses with equality.

7.1 Installation

`FLEA` is open source (but still without license) and available on `GitHub`[2]. It depends on a working `Swift 3.1`[3] toolchain at build time and the local installation of `Yices 2`[4] and `Z3`[5] at build and run time. After installation of these three prerequisites you may check the success with commands and results from Listing 7.1.

Not a strict prerequisite but we recommend to download, to unpack `TPTP-v6.4.0.tgz` (or newer) from the `TPTP` website[6], and to create a symbolic link to the unpacked library in your home directory.

```
$ yices - V
Yices 2.5.1          # or newer
$ z3 --version
Z3 version 4.5.1     # or newer
$ swift -version
Swift version 3.1    # or newer
$ ls \( \text{\textasciitilde} \)/TPTP
Axioms    Documents    Generators    Problems    README    Scripts
```

Listing 7.1: Check toolchain and libraries

After we have downloaded and installed external tool chain and libraries we clone `FLEA` and install the parsing lib as in Listing 7.2.

```
$ git clone https://github.com/AleGit/FLEA # download sources
$ cd FLEA
```

[1] All theory is gray, my friend. But forever green is the tree of life.
[2] github.com/AleGit/FLEA
[3] Instructions and binaries available on swift.org
[4] Instruction and binary available on yices.csl.sri.com
[5] Instructions and source code available on github.com/Z3Prover/z3
[6] www.cs.miami.edu/~tptp

```
$ swift build                           # downloads dependencies, but
  fails
$ pushd Packages/CTptpParsing-1.0.0     # or 1.0.1 or ...
$ sudo make install                     # install parsing lib
$ popd
```

Listing 7.2: Download `FLEA` and install parsing lib

```
$ Scripts/z3headers.sh                  # workaround for headers
$ Scripts/ctests.sh                     # build and run all tests
```

Listing 7.3: Build and and run all `FLEA` tests.

7.2 Usage

7.3 Data struture

```
protocol Node: Hashable {
  associatedtype Symbol : Hashable
  var symbol: Symbol { get set }
  var nodes: [Self]? { get set }
}
```

Listing 7.4: Simplified definition of general terms

7.4 Encodings

We can simply encode first order atoms purely propositional when we construct the name for the propositional atom from a predicate or equation recursively as concatenation of symbols. After installation of these prerequisites you may check if everything went well.

Definition 7.1. We derive the propositional identifier of a general term as follows

$$\xi(t) = \begin{cases} \bot & \text{if } t = x \in \mathcal{V}, \bot \notin \mathcal{F} \\ \mathsf{c} & \text{if } t = c \in \mathcal{F}^{(0)} \\ f \cdot \xi(t_1) \cdot \ldots \cdot \xi(t_n) & \text{if } t = f(t_1, \ldots, t_n), f \in \mathcal{F}^{(n)} \end{cases}$$

where $\bot, \cdot \notin \mathcal{F}$ are distinct symbols that do not occur in the signature of the set of clauses.

Example 7.2. We encode a simple predicate and a simple equation as follows.

$$\xi(\mathsf{p}(\mathsf{f}(x,y), g(y))) = \mathsf{p} \cdot \mathsf{f} \cdot \bot \cdot \bot \cdot \mathsf{g} \cdot \bot$$
$$\xi(\mathsf{f}(x,y) \approx \mathsf{g}(y)) = \approx \cdot \mathsf{f} \cdot \bot \cdot \bot \cdot \mathsf{g} \cdot \bot$$

Definition 7.3. We define den SMT-Encoding as follows

$$\Xi(t) = \begin{cases} \mathsf{c}_0 & : \quad U & \text{if } t = x \in \mathcal{V}, \mathsf{c}_0 \notin \mathcal{F}^{(0)} \\ \mathsf{c} & : \quad U & \text{if } t = c \in \mathcal{F}_{\mathsf{f}}^{(0)} \\ \mathsf{f} & : \quad (U^n \to U)\ \Xi(t_1)\ \ldots\ \Xi(t_n) & \text{if } t = \mathsf{f}(t_1, \ldots, t_n), \mathsf{f} \in \mathcal{F}_{\mathsf{f}}^{(n)} \\ \mathsf{P} & : \quad \mathsf{Bool} & \text{if } t = p \in \mathcal{F}_{\mathsf{P}}^{(0)} \\ \mathsf{P} & : \quad (U^n \to \mathsf{Bool})\ \Xi(t_1)\ \ldots\ \Xi(t_n) & \text{if } t = \mathsf{P}(t_1, \ldots, t_n), \mathsf{P} \in \mathcal{F}_{\mathsf{P}}^{(n)} \end{cases}$$

```
struct Yices {
 static func setUp() {
  yices_init()
 }
 static func tearDown() {
  yices_exit()
 }
}
```

Listing 7.5:

```
extension Yices {
 static var bool_tau = yices_bool_type()
 static var free_tau: type_t { return namedType("\( \tau \)") }

 /// Get or create (uninterpreted) type ‘name‘.
 static func namedType(_ name: String) -> type_t {
  var tau = yices_get_type_by_name(name)
  if tau == NULL_TYPE {
   tau = yices_new_uninterpreted_type()
   yices_set_type_name(tau, name)
  }
  return tau
 }

 /// Get or create an uninterpreted global ‘symbol‘ of type ‘term_tau‘.
 static func typedSymbol(symbol: String, term_tau: type_t) -> term_t {
  var c = yices_get_term_by_name(symbol)
  if c == NULL_TERM {
   c = yices_new_uninterpreted_term(term_tau)
   yices_set_term_name(c, symbol)
  }
  return c
 }
```

Listing 7.6:

```
 /// Get or create a global constant ‘symbol‘ of type ‘term_tau‘
 static func constant(_ symbol: String, term_tau: type_t) -> term_t {
  return typedSymbol(symbol, term_tau: term_tau)
 }

 /// Create a homogenic domain tuple
 static func domain(_ count: Int, tau: type_t) -> [type_t] {
  return [type_t](repeating: tau, count: count)
 }

 /// Get or create a function symbol of type domain -> range
 static func function(_ symbol: String, domain: [type_t], range: type_t)
   -> term_t {
  let f_tau = yices_function_type(UInt32(domain.count), domain, range)
  return typedSymbol(symbol, term_tau: f_tau)
 }
}
```

Listing 7.7:

```
yices_init()
let B = yices_bool_type()
let U = Yices.namedType(name: "τ")
```

Listing 7.8: SMT encoding

```
func encodeSAT<N:Node>(term: N) -> term_t {
  switch n.symbol.type {
    case .negation:
      return yices_not( encode( term.nodes!.first!) )

    case .predicate, .equation:
      return typedSymbol( ξ(term), term_tau: boolType)
  }
}
```

Listing 7.9: Propositional encoding

```
func encodeEUF<N:Node>(term: N) -> term_t {
  switch term.symbol.type {
  case .negation:
    return yices_not( encodeEUF( term.nodes.first!) )

  case .predicate:
    return application(term.symbol, nodes:term.nodes!, term_tau: boolType
    )

  case .equation:
    return typedSymbol( ξ(term), term_tau: boolType)
  }
}
```

Listing 7.10: EUF encoding

```
let boolType : type_t = yices_bool_type(void)

let freeType : type_t = yices_new_uninterpreted_type(void)
yices_set_type_name(freeType, "τ") // pretty printing

func typedSymbol(_ symbol: String, term_tau: type_t) -> term_t {
  var t = yices_get_term_by_name(symbol)
  if t == NULL_TERM {
    t = yices_new_uninterpreted_term(term_tau)
    yices_set_term_name(t, symbol)
  }
  return t
}

func constant(_ symbol: String, term_tau: type_t) -> term_t {
  return typedSymbol(symbol, term_tau: term_tau)
}
```

Listing 7.11: Yices types, symbols, and constants

7.4.1 QF_EUF

```
func domain(_ count: Int, tau: type_t) -> [type_t] {
  return [type_t](repeating: tau, count: count)
}
```

Listing 7.12:

```
func function(_ symbol: String,
    domain: [type_t], range: type_t) -> term_t {
  let f_tau = yices_function_type(UInt32(domain.count), domain, range)
  return typedSymbol(symbol, term_tau: f_tau)
}
```

Listing 7.13:

```
func application(_ symbol: String,
      args: [term_t], term_tau: type_t) -> term_t {
  let f = function(symbol,
  domain:domain(args.count, tau:Yices.free_tau), range: term_tau)
  return yices_application(f, UInt32(args.count), args)
}
```

Listing 7.14:

Definition 7.4 (Flattening of clauses)**.**

Example 7.5.

$$\mathsf{P}(t_1, \ldots, t_n) \Rightarrow y_1 \neq t_1 \vee \ldots y_n \neq t_n \vee \mathsf{P}(y_1, \ldots, y_n)$$

$$s \not\approx t \Rightarrow y_s \not\approx s \vee y_t \not\approx t \vee y_s \approx y_t$$

7.5 Experiments

7.5.1 The TPTP library

$$
\begin{aligned}
F &= (F_1 \land \ldots \land F_n) \rightarrow G \\
\neg F &\equiv \neg\,(\neg(F_1 \land \ldots \land F_n) \lor G) \\
&\equiv (F_1 \land \ldots \land F_n) \land \neg G
\end{aligned}
$$

8 Related Work

8.1 Purely equational logic

We can transform predicates into equations by introducing one new and distinct constant symbol $\bullet$ (smaller than any other term), and a new function symbol f_P for every predicate symbol P:

$$\begin{aligned} P(t_1,\dots,t_n) &\quad\Rightarrow\quad f_P(t_1,\dots,t_n) \approx \bullet \\ \neg P(t_1,\dots,t_n) &\quad\Rightarrow\quad f_P(t_1,\dots,t_n) \not\approx \bullet \end{aligned}$$

8.2 Term Rewrite Systems

We can transform a set of selected literals into a term rewrite system when we introduce two new constant symbols — true and false — to transform each literal into a rewrite rule.

$$\begin{aligned} P(t_1,\dots,t_n) &\quad\Rightarrow\quad P(t_1,\dots,t_n) \to \mathsf{true} \\ \neg P(t_1,\dots,t_n) &\quad\Rightarrow\quad P(t_1,\dots,t_n) \to \mathsf{false} \\ s \approx t &\quad\Rightarrow\quad s \approx t \to \mathsf{true} \\ r \not\approx u &\quad\Rightarrow\quad r \approx u \to \mathsf{false} \end{aligned}$$

When we find a not unifiable critical peek, then the set of literals is inconsistent.

Example 8.1. Consider the unsatisfiable set $S = \{\mathsf{P}(x), \neg\mathsf{P}(\mathsf{f}(y))\}$. We derive the term rewrite system

$$\begin{array}{lrr} (1) & \mathsf{P}(x) \to \mathsf{true} & \ell_1 \to r_1 \\ (2) & \mathsf{P}(\mathsf{f}(y)) \to \mathsf{false} & \ell_2 \to r_2 \end{array}$$

with overlap $\langle (1), p, (2) \rangle$ with $p = \epsilon$, $\sigma = \mathrm{mgu}(l_1, l_2|_\epsilon) = \{x \mapsto \mathsf{f}(y)\}$. From overlap $(\ell_2\sigma[r_1\sigma]_\epsilon, \epsilon, \ell_2\sigma, r_2\sigma) = (\mathsf{true}, \epsilon, \mathsf{P}(\mathsf{f}(y)), \mathsf{false})$ we obtain critical pair $\mathsf{true} \approx \mathsf{false}$.

9 Conclusion

All work and no play
makes Jack a dull boy

James Howell, Paramoigraphy
(Proverbs), 1659.

10 Future Work

List of Figures

List of Tables

Bibliography

[1] L. Albert, R. Casas, and F. Fages. Average-case analysis of unification algorithms. *Theoretical Computer Science*, 113(1):3 – 34, 1993.

[2] F. Baader and T. Nipkow. *Term Rewriting and All That.* Cambridge University Press, New York, NY, USA, 1998.

[3] A. Biere, A. Biere, M. Heule, H. van Maaren, and T. Walsh. *Handbook of Satisfiability: Volume 185 Frontiers in Artificial Intelligence and Applications.* IOS Press, Amsterdam, The Netherlands, The Netherlands, 2009.

[4] E. Börger, E. Grädel, and Y. Gurevich. *The classical decision problem.* Perspectives in Mathematical Logic. Springer-Verlag, Berlin, 1997. With an appendix by Cyril Allauzen and Bruno Durand.

[5] M. Davis, G. Logemann, and D. Loveland. A machine program for theorem-proving. *Commun. ACM*, 5(7):394–397, July 1962.

[6] M. Davis and H. Putnam. A computing procedure for quantification theory. *J. ACM*, 7(3):201–215, July 1960.

[7] H. Ganzinger and K. Korovin. Integrating Equational Reasoning into Instantiation-Based Theorem Proving. In *18th CSL 2004. Proceedings*, volume 3210 of *LNCS*, pages 71–84, 2004.

[8] P. C. Gilmore. A proof method for quantification theory: Its justification and realization. *IBM Journal of Research and Development*, 4(1):28–35, Jan 1960.

[9] J. Harrison. *Handbook of Practical Logic and Automated Reasoning.* Cambridge University Press, New York, NY, USA, 1st edition, 2009.

[10] M. Huth and M. Ryan. *Logic in Computer Science: Modelling and Reasoning About Systems.* Cambridge University Press, New York, NY, USA, 2004.

[11] D. Kroening and O. Strichman. *Decision Procedures: An Algorithmic Point of View.* Springer Publishing Company, Incorporated, 1 edition, 2008.

[12] A. Middeldorp. Lecture Notes – Term Rewriting, 2015.

[13] A. Middeldorp. Lecture Notes – Logic, 2016.

[14] G. Moser. Lecture Notes – Module Automated Reasoning, 2013.

[15] R. Nieuwenhuis and A. Rubio. Paramodulation-based theorem proving. In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, volume I, pages 371–443. Elsevier, 2001.

[16] D. A. Plaisted and S. Greenbaum. A structure-preserving clause form translation. *Journal of Symbolic Computation*, 2(3):293 – 304, 1986.

[17] J. A. Robinson. A machine-oriented logic based on the resolution principle. *J. ACM*, 12(1):23–41, Jan. 1965.

[18] G. S. Tseitin. On the complexity of derivations in the propositional calculus. *Studies in Constructive Mathematics and Mathematical Logic*, Part II:115–125, 1970.