Die approbierte Originalversion dieser Diplom-/Masterarbeit ist in der Hauptbibliothek der Technischen Universität Wien aufgestellt und zugänglich.
http://www.ub.tuwien.ac.at

TU UB WIEN Universitätsbibliothek

The approved original version of this diploma or master thesis is available at the main library of the Vienna University of Technology.
http://www.ub.tuwien.ac.at/eng

TECHNISCHE UNIVERSITÄT WIEN
Vienna University of Technology

DIPLOMARBEIT

Finding loop invariants using tree grammars

Ausgeführt am Institut für
Diskrete Mathematik und Geometrie
der Technischen Universität Wien

unter der Anleitung von
Privatdoz. Dr. techn. Stefan Hetzl

durch
Gabriel Ebner
Hütteldorfer Straße 202/14
1140 Wien

Datum Unterschrift

Introduction

Herbrand's theorem was first proved by its namesake Jacques Herbrand in [Her30]: a special case of it says that a formula $\exists x\, \varphi[x]$ with $\varphi[x]$ quantifier-free is valid if and only if there are terms $t_1, \dots, t_n$ such that $\varphi[t_1] \vee \cdots \vee \varphi[t_n]$ is a tautology.

In [Gen35], Gentzen introduced the sequent calculus LK, together with a procedure to eliminate cut inferences in this calculus. Using this result, we can obtain an easy proof of Herbrand's theorem: after applying cut-elimination, the terms $t_1, \dots, t_n$ are explicit in the inferences of the proof.

However, even if the proof contains cut inferences, we can still extract an explicit description of the terms $t_1, \dots, t_n$—at least if the cut formulas in the cut inferences are simple enough. If they are all of the form $\forall y\, \psi[y]$, i.e. prenex formulas with a single universal quantifier, then we can extract a totally rigid tree grammar generating these terms $t_1, \dots, t_n$ as proved in [Het12]. In this grammar, each non-terminal corresponds to a cut in the proof. This extraction then suggests a method to introduce cut inferences: starting from a Herbrand sequent, we find a grammar generating it, and can then choose a cut formula for each non-terminal, giving a proof with cuts; this procedure has been successful in experiments on a large proof database, see [Het+14b] for details.

Conceptually we can view an inductive proof of $\varphi[0] \vdash \forall z \in \mathbb{N}\, \varphi[z]$ as a proof containing infinitely many similar cuts, each with the cut formula $\varphi[i] \to \varphi[i+1]$. And we can again extract a grammar where the non-terminals correspond to these cuts—but due to the similarity of these infinitely many cuts, it suffices to consider one schematic non-terminal. A procedure analogous to cut-introduction was proposed in [EH15]; starting from Herbrand sequents, we can find a grammar generating them, and again (with a bit more work) choose suitable cut formulas—and for an inductive proof, these cut formulas are inductive invariants. These invariants are the crucial and difficult part in an inductive proof; given an inductive invariant, it is relatively easy to complete it into a full inductive proof. So this procedure gives a method for automated inductive theorem proving.

Similar invariants are also studied by a different community. In program verification, loop invariants take a similar place as induction formulas: a loop invariant is a formula that is true at every iteration of a loop; they have been used for the verification of loops since the early efforts of Hoare [Hoa69] and Floyd [Flo67]. Again, finding these invariants is the difficult part; completing the proof usually just requires verifying the validity of quantifier-free formulas.

In this thesis, we will apply and adapt the method for inductive theorem proving using grammars to the verification of loop programs.

Contents

Introduction i

Contents iii

1 Term languages 1
- 1.1 Tree grammars . . . 2
- 1.2 Normal forms . . . 3
  - 1.2.1 Reduction to propositional logic . . . 11
  - 1.2.2 Computing normal forms . . . 13
- 1.3 Minimizing grammars . . . 16
- 1.4 Finding a minimal grammar . . . 17

2 Simple proofs 19
- 2.1 Sequent calculus . . . 19
- 2.2 Extended Herbrand sequents . . . 21
- 2.3 Grammars . . . 27

3 Inductive proofs 29
- 3.1 Schematic sips . . . 32
- 3.2 Finding minimal sip grammars . . . 35
- 3.3 Finding induction formulas . . . 38

4 While programs 41
- 4.1 Operational semantics . . . 41
- 4.2 Hoare logic . . . 43

5 Loop verification 47
- 5.1 Finding slp grammars . . . 50
- 5.2 Finding loop invariants . . . 51
- 5.3 Overview . . . 52
- 5.4 Examples . . . 52
  - 5.4.1 Addition . . . 53
  - 5.4.2 Array initialization . . . 54
  - 5.4.3 Bubble sort . . . 57

Conclusion 61

Bibliography 63

1 Term languages

The following are standard definitions as found in [Com+07] or [BN98].

Definition 1.1. A signature Σ is a set of function symbols, where each $f \in \Sigma$ is associated with a non-negative integer n, the arity of f.

Elements $f \in \Sigma$ with arity 0 are called constant symbols.

Definition 1.2. Let Σ be a signature and X be a set of variables such that $\Sigma \cap X = \emptyset$. The set $\mathcal{T}(\Sigma, X)$ of all terms is inductively defined as:

- $X \subseteq \mathcal{T}(\Sigma, X)$.
- For any $f \in \Sigma$ with arity n: if $t_1, \ldots, t_n \in \mathcal{T}(\Sigma, X)$, then $f(t_1, \ldots, t_n) \in \mathcal{T}(\Sigma, X)$.

Definition 1.3. Let Σ be a signature and X a set of variables such that $\Sigma \cap X = \emptyset$. A substitution is a map $\sigma\colon X \to \mathcal{T}(\Sigma, X)$ from variables to terms.

This map homomorphically extends to a map $\overline{\sigma}\colon \mathcal{T}(\Sigma, X) \to \mathcal{T}(\Sigma, X)$:

$$\begin{aligned} \overline{\sigma}(x) &:= \sigma(x) \text{ for } x \in X \\ \overline{\sigma}(f(t_1, \ldots, t_n)) &:= f(\overline{\sigma}(t_1), \ldots, \overline{\sigma}(t_n)) \end{aligned}$$

We will write σ for $\overline{\sigma}$ if no confusion arises, and often write the application of a substitution σ to a term t in postfix notation, i.e. $t\sigma = \overline{\sigma}(t)$.

Definition 1.4. Let X be a set. The set X^* denotes the finite sequences of elements in X, including the empty sequence ϵ.

For sequences $a, b \in X^*$, the sequence ab denotes their concatenation.

Definition 1.5. Let $s \in \mathcal{T}(\Sigma, X)$ be a term.

The set of positions $\mathrm{Pos}(s) \subseteq \mathbb{N}^*$ is defined recursively as:

- $\mathrm{Pos}(x) = \{\epsilon\}$ for $x \in X$ where ϵ is the empty sequence.
- $\mathrm{Pos}(f(t_1, \ldots, t_n)) = \{\epsilon\} \cup 1\,\mathrm{Pos}(t_1) \cup \cdots \cup n\,\mathrm{Pos}(t_n)$ for $f \in \Sigma$ with arity n.

The size $|s|$ of a term s is the number $|\mathrm{Pos}(s)|$ of its positions.

For $p \in \mathrm{Pos}(s)$, the subterm of s at position p, written $s|_p$ is defined recursively as:

- $t|_\epsilon = t$.
- $f(t_1, \ldots, t_n)|_{iq} = t_i|_q$.

For $p \in \mathrm{Pos}(s)$ and a term $t \in \mathcal{T}(\Sigma, X)$, replacing the subterm at position p by t in s gives the term $s[t]_p$:

- $s[t]_\epsilon = t$.
- $f(s_1, \dots, s_n)[t]_{iq} = f(s_1, \dots, s_i[t]_q, \dots, s_n)$.

Definition 1.6. Let $s, t \in \mathcal{T}(\Sigma, X)$ be terms. The term t is a subterm of s, written $t \trianglelefteq s$, if there exists a position $p \in \mathrm{Pos}(s)$ such that $t = s|_p$. The set $\mathrm{st}(s) = \{t \in \mathcal{T}(\Sigma, X) \colon t \trianglelefteq s\}$ consists of all subterms of s.

The term t is a strict subterm of s, written $t \triangleleft s$, if $t \trianglelefteq s$ and $t \neq s$.

The set of variables in s is the set $\mathrm{Var}(s) := \{v \in X : v \trianglelefteq s\}$ of variables which are subterms of s.

Definition 1.7. Terms $t \in \mathcal{T}(\Sigma, \emptyset)$ are called ground terms. We abbreviate $\mathcal{T}(\Sigma) = \mathcal{T}(\Sigma, \emptyset)$ as the set of ground terms over the signature Σ.

1.1 Tree grammars

Tree grammars and automata are tools from formal language theory to describe families of trees or in our case, terms. For a general introduction to the topic, see [Com+07]. Rigid tree automata were introduced in [JKV11], their recognized languages correspond to those of rigid tree grammars as introduced in [Het12], where they naturally arise as grammars generating Herbrand sequents for simple proofs with cut.

Definition 1.8. A regular tree grammar $G = (\tau, N, \Sigma, P)$ over a signature Σ is composed of a finite set of non-terminals N such that $\Sigma \cap N = \emptyset$, an axiom $\tau \in N$, and a finite set of productions P where each production $\alpha \to t$ is a pair of a non-terminal $\alpha \in N$ and a term $t \in \mathcal{T}(\Sigma, N)$.

Definition 1.9. A totally rigid acyclic tree grammar, short trat grammar, is a tree grammar $G = (\tau, N, \Sigma, P)$ that is acyclic, i.e.:

There exists a partial order $<$ on the set N of non-terminals such that $\alpha < \mathrm{Var}(t)$ for any production $\alpha \to t \in P$.

Definition 1.10. Let $G = (\tau, N, \Sigma, P)$ be a trat grammar.

A single-step derivation $s \to^1 t$ is a pair of terms $s, t \in \mathcal{T}(\Sigma, N)$ together with a position $p \in \mathrm{Pos}(s)$ and a production $\alpha \to k \in P$ such that $s|_p = \alpha$ and $t = s[k]_p$.

A derivation of a term $t \in \mathcal{T}(\Sigma)$ is a sequence of single-step derivations $\tau = t_1 \to^1 \cdots \to_1 t_n = t$ that is rigid: if a non-terminal occurs more than once in the derivation, i.e. $t_i|_p = t_j|_q = \alpha \in N$, then $t|_p = t|_q$.

Definition 1.11. Let G be a trat grammar. The language $\mathcal{L}(G)$ generated by G is the set of all terms derivable in G.

Lemma 1.12. *Let $G = (\tau, N, \Sigma, P)$ be a trat grammar. Then $\mathcal{L}(G)$ is finite, and $|\mathcal{L}(G)| \leq |N|^{|P|+1}$.*

Proof. Let $t \in \mathcal{L}(G)$ be a term, and d a derivation of t. We can define a partial function $p_t : N \to P$ that assigns to each non-terminal α the first production $p_t(\alpha) = \alpha \to \dots$ starting with α that occurs in d, if the non-terminal α does occur at all.

We can recover t from p_t. Use acyclicity to order the non-terminals as $\tau = \alpha_0 < \cdots < \alpha_n$, and define t_i recursively as $t_i = k[\alpha_{i+1}\backslash t_{i+1}, \dots, \alpha_n\backslash t_n]$ if $p_t(\alpha_i) = \alpha_i \to k$ exists, and $t_i = c$ for an arbitrary constant c otherwise.

Each t_i is now a ground term equal to the rigid value of the non-terminal α_i[1]; i.e. if α_i occurs in the derivation at a given position then t_i will be the ground term at the same position in t. Since $t_0 = t$, the mapping $t \mapsto p_t$ is injective. □

1.2 Normal forms

Definition 1.13. A finite set of terms is called a language.

Definition 1.14. Let G be a trat grammar, and L a language. The grammar G covers the language L, if $\mathcal{L}(G) \supseteq L$.

For brevity, we will usually just write "grammar" instead of "trat grammar".

Definition 1.15. Let $G = (\tau, N, \Sigma, P)$ be a grammar. Its size $|G|$ is the number of its productions $|P|$.

Given a language L, we want to find a small grammar G covering this language as that gives a concise description of L. (The language L will correspond to a Herbrand sequent, and the grammar G to a proof with cuts—a smaller grammar will correspond to a better proof.)

Our approach to this problem will be to take a suitable large grammar and make it smaller in the following sense:

Definition 1.16. Let $G = (\tau, N, \Sigma, P)$ and $G' = (\tau, N, \Sigma, P')$ be trat grammars in the same signature and with the same non-terminals.

We say that G' is a sub-grammar of G, and write $G' \subseteq G$, if the productions are subsets: $P' \subseteq P$.

It is always possible to find a grammar covering L, by simply including the productions $\tau \to t$ for all $t \in L$. But this grammar is fairly large; and more importantly, we cannot hope to simplify this naive grammar by taking sub-grammars, i.e. removing productions—each production is clearly necessary.

Fix a set of non-terminals $N = \{\alpha_0, \dots, \alpha_n\}$. Another easy (although infinite) "grammar" that covers L is the maximal grammar: Simply take *all* productions.[2] Clearly, this grammar contains any possible grammar with those non-terminals as a sub-grammar.

---

[1] We will encounter this again in Lemma 1.31, where this will this rigid value will be $\overline{\delta}_d(\alpha_i)$

[2] The "grammar" obtained this way is infinite—but for the purpose of the example, it would be enough to consider the finite grammar consisting only of those productions that are smaller than the maximum size of a term in L.

As an example consider the following language:

$$\Sigma = \{f/2, c/0, d/0\}$$
$$L = \{f(c,c), f(d,d)\}$$

The maximal grammar would then contain the following productions:

$$P = \bigcup_{0 \le i < n} \{\alpha_i \to t \colon t \in \mathcal{T}(\Sigma, \{\alpha_{i+1}, \dots, \alpha_n\})\}$$

However, minimizing this huge list of productions is computationally infeasible, and it would be nice if we could find our small grammar as a subset of a more manageable set of productions.

Observe that some of the productions are unnecessarily general if we are only interested in covering L. For example, consider the production $\alpha_0 \to f(\alpha_3, \alpha_8)$. If we start a derivation with this production, and then apply a production for α_3, we will always have to apply the corresponding production for α_8 if we want to get a term in L:

$$\alpha_0 \to^1 f(\alpha_3, \alpha_8) \to^1 f(c, \alpha_8) \to^1 f(c,c)$$
$$\cdots \to^1 f(c, \alpha_{20}) \to^1 f(c,c)$$

Maybe we apply $\alpha_8 \to c$; or maybe $\alpha_8 \to \alpha_{20}$ and then $\alpha_{20} \to c$, but we will always end up with $f(c,c)$ after we apply $\alpha_3 \to c$.

Instead of $\alpha_0 \to f(\alpha_3, \alpha_8)$, we could have always started with $\alpha_0 \to f(\alpha_3, \alpha_3)$.

Why is this the case? If we take a look at L, the terms have a particular pattern: the two arguments of the function symbol f are always the same. We will formalize this by saying that $f(\alpha_3, \alpha_8)$ satisfies the equation $\alpha_3 = \alpha_8$ in L.

Eberhard and Hetzl showed in [EH14] that we only need productions whose right side does not satisfy any non-trivial equations for the grammar of minimal size covering L.

Let us assume that the signature Σ is infinite. We will formalize normal forms using the theory of free term algebras. These algebras, also called Herbrand universes or finite tree algebras, are widely studied in computer science because they describe inductive data types [Mah88; Opp78].

In this work we are interested in applying term algebras to productions in trat grammars, where non-terminals have a similar purpose as variables—and for this reason we will call variables "non-terminals".

Definition 1.17 (Logic of term algebras)**.** Formulas are constructed from equations $q = r$ of terms possibly containing non-terminals, using the propositional connectives $\bot$, $\top$, $\neg$, $\land$, $\lor$, and $\to$.

Models of this logic are substitutions, and a substitution σ satisfies an equation $q = r$,

in symbols $\sigma \models q = r$, if $q\sigma = r\sigma$. Truth is defined as usual, i.e.:

$$\begin{array}{rcl} \sigma \models \bot & \Leftrightarrow & \bot \\ \sigma \models \top & \Leftrightarrow & \top \\ \sigma \models \neg\varphi & \Leftrightarrow & \neg\, \sigma \models \varphi \\ \sigma \models \varphi \wedge \psi & \Leftrightarrow & \sigma \models \varphi \quad \wedge \quad \sigma \models \psi \\ \sigma \models \varphi \vee \psi & \Leftrightarrow & \sigma \models \varphi \quad \vee \quad \sigma \models \psi \\ \sigma \models \varphi \rightarrow \psi & \Leftrightarrow & \sigma \models \varphi \quad \rightarrow \quad \sigma \models \psi \end{array}$$

A formula φ entails a formula ψ, in symbols $\varphi \models \psi$, if $\sigma \models \varphi$ implies $\sigma \models \psi$ for all substitutions σ; φ is valid, in symbols $\models \varphi$, if $\sigma \models \varphi$ for all σ.

For a term t and finite sets of terms L and L', we introduce the following abbreviations:

$$\begin{array}{rcl} t \in L & \equiv & \bigvee_{s \in L} t = s \\ L \subseteq L' & \equiv & \bigwedge_{t \in L} t \in L' \end{array}$$

For a term t and a language L, we say that t satisfies the equation $q = r$ in L, if $t \in L \models q = r$.

Example 1.18. The following are valid formulas in the logic of term algebras:

$$\begin{array}{l} \models f(c) = f(c) \\ \models f(c) \neq f(d) \\ \models f(\alpha) = f(\beta) \rightarrow \alpha = \beta \\ \models f(\alpha) \neq \alpha \\ \models c \in \{c, d\} \\ \models f(\alpha) \notin \{c, d\} \\ \models f(\alpha) \in \{c, d, f(c)\} \rightarrow f(\alpha) = f(c) \end{array}$$

Lemma 1.19. *Let $q = r$ be a satisfiable equation, then there exist non-terminals $\alpha_1, \dots, \alpha_n$ and terms $t_1, \dots, t_n$ such that $\models q = r \leftrightarrow \alpha_1 = t_1 \wedge \cdots \wedge \alpha_n = t_n$.*

The non-terminals can be chosen to be pairwise distinct and to occur in $q = r$, and the terms t_i can be chosen such that either t_i does not contain the non-terminals $\alpha_1, \dots, \alpha_n$, or that all t_i occur as subterms in $q = r$.

Proof. Break down $q = r$ recursively: if $q = f(q_1, \dots, q_m)$ and $r = f(r_1, \dots, r_m)$, then $\models q = r \leftrightarrow q_1 = r_1 \wedge q_2 = r_2 \wedge \cdots \wedge q_m = r_m$; repeat this process recursively for each smaller equation $q_i = r_i$. Otherwise, q or r needs to be a non-terminal, or $q = r$ would have been unsatisfiable—here, $q = r$ is already in the required form (we might need to reorder the equation though).

By construction, each of the non-terminals α_i was a subterm of $q = r$.

If we have two conjuncts $\alpha_i = t_i$ and $\alpha_j = t_j$ with the same non-terminal $\alpha_i = \alpha_j$ and $t_i, t_j \notin N$, then we can replace $\alpha_i = t_i \wedge \alpha_j = t_j$ by the equivalent $\alpha_i = t_i \wedge t_i = t_j$ and break down the second equation $t_i = t_j$ recursively again. This terminates, since we remove at least one function symbol per reduction step.

This works if we do not have equations between two non-terminals. If we have those, then we need to "regularize" the equations: take the reflexive transitive closure $\approx$ of the relation "$\alpha_i = \alpha_j$ is a conjunct" between non-terminals. Order the non-terminals in each equivalence class $\{\beta_1, \dots, \beta_l\}$, replace $\beta_2, \dots, \beta_l$ by β_1 in the other conjuncts, and replace the non-terminal equations by $\beta_l = \beta_{l-1} \wedge \cdots \wedge \beta_2 = \beta_1$. Doing this after every reduction ensures that the non-terminals α_i are distinct at the end.

To finish the proof, we only need to remove occurrences of the non-terminals α_i on the right-hand side of the equations. Substitute α_1 by t_1 in all equations except $\alpha_1 = t_1$—if α_1 occurred in t_1, the original equation would have already been unsatisfiable. Continue with the other non-terminals. □

Example 1.20. $\models f(\alpha, \gamma) = f(\beta, c) \leftrightarrow \alpha = \beta \wedge \gamma = c$

Remark 1.21. The construction in Lemma 1.19 is reminiscent of the construction of a most general unifier. And indeed, if we view the resulting equations $\alpha_1 = t_1 \wedge \cdots \wedge \alpha_n = t_n$ as a substitution $\sigma = [\alpha_1 \backslash t_1, \dots, \alpha_n \backslash t_n]$, then σ is a most general unifier.

However, in general, most general unifiers may rename variables—and such substitutions would not give equivalent formulas. For example, consider the equation $f(\alpha) = f(\beta)$: The substitution $\sigma = [\alpha \backslash \gamma, \beta \backslash \gamma]$ is clearly a most general unifier, but $\not\models f(\alpha) = f(\beta) \leftrightarrow \alpha = \gamma \wedge \beta = \gamma$.

Definition 1.22. An equation of the form $s = s$ is called trivial.

Definition 1.23. A term k is in normal form relative to a language L, if $k \in L \models q = r$ and $q, r \trianglelefteq k$ implies that $q = r$ is trivial.

Example 1.24. The term $f(\alpha, c)$ is not in normal form relative to $L = \{f(c, c), f(d, d)\}$, since $f(\alpha, c) \in L \models \alpha = c$.

If we consider the language $L = \{f(f(f(c)))\}$, then all of the following terms are in normal form relative to L: $\alpha, f(\alpha), f(f(\alpha)), f(f(f(\alpha)))$, and $f(f(f(c)))$.

Lemma 1.25. *For a term k and a language L, the following are equivalent:*

1. *k is in normal form relative to L*
2. *If $k \in L \models \alpha = r$ and $\alpha, r \trianglelefteq k$, then $\alpha = r$ is trivial.*

Proof. Apply Lemma 1.19. □

The choice of how to define normal forms allows a bit of freedom in specifying what equations are disallowed:

(a) Any equation.

(b) Any equation of subterms or ground terms.

(c) Any equation of subterms.

Each of these choices defines a progressively larger class of normal forms. To illustrate, consider the following language:

$$L = \{f(g(c), h(c)), f(g(d), h(d))\}$$

For example the term $f(\alpha, h(\beta))$ satisfies essentially only the equation $\alpha = g(\beta)$, hence it is not in normal form relative to L according to definition (a), but it is in normal form according to definitions (b) and (c) since $g(\beta)$ is neither a subterm of $f(\alpha, h(\beta))$ nor ground.

The term $f(\alpha, h(c))$ satisfies essentially only the equation $\alpha = g(c)$; it is not in normal form according to definitions (a) or (b), but still in normal form according to definition (c).

Many of the following lemmas are true for each of these definitions (with minor changes), in particular Theorems 1.27 and 1.48 and Lemma 1.28, which guarantee that the set of normal forms can be computed in polynomial time.

Definition (b) is the one originally presented in [EH14]; we are using definition (c) because it allows for more normal forms, which will become necessary for slp grammars.

Lemma 1.26. *Let k be a term and $L \subseteq L'$ languages.*

- *$k \in L' \models k \in L$.*
- *Larger sets satisfy fewer equations: If $k \in L' \models q = r$, then $k \in L \models q = r$ for any equation $q = r$.*
- *Larger sets have more normal forms: If k is in normal form relative to L, then k is in normal form relative to L'.*

Proof. To show that $k \in L' \models k \in L$, we only need to expand the definitions:

$$\begin{aligned} k \in L' &\leftrightarrow \bigvee_{t \in L'} k = t \\ &\leftrightarrow \bigvee_{t \in L} k = t \vee \bigvee_{t \in L' \setminus L} k = t \\ &\rightarrow k \in L \end{aligned}$$

If now $k \in L' \models q = r$ and $\sigma \models k \in L$, then $\sigma \models k \in L'$ by the above argument and $\sigma \models q = r$ by assumption.

We need to show that any equation $q = r$ such that $k \in L' \models q = r$ and $q, r \trianglelefteq k$ is trivial. But if $k \in L' \models q = r$, then $k \in L \models q = r$ as well, and the equation $q = r$ is trivial since k is in normal form relative to L. □

Theorem 1.27. *Let k be a term in normal form relative to L with n non-terminals, then there is a subset $L' \subseteq L$ with $|L'| \leq n$ such that k is still in normal form relative to L.*

Proof. We will construct a sequence of languages $L_0 \subseteq L_1 \subseteq \cdots \subseteq L_n \subseteq L$ such that $|L_i| \leq i+1$ for all $0 \leq i \leq n$ with the aim that $L' = L_n$.

Choose $L_0 = \{t_0\}$ such that $\sigma \models k \in L_0$ for some σ. Now order the non-terminals occurring in k by a total order $<$ such that $\alpha < \beta$ if $\sigma(\alpha)$ is a strict subterm of $\sigma(\beta)$.

This order has the useful property that if $k \in L_i \models \alpha = t$, then any non-terminal β that is a strict subterm of t satisfies $\beta < \alpha$, since $\sigma(\beta)$ is then a strict subterm of $\sigma(\alpha)$. Hence we only need to consider two possibilities: The term t only contains non-terminals $\beta < \alpha$; or $t = \beta$ such that $\sigma(\alpha) = \sigma(\beta)$.

Let the non-terminals be $\alpha_1 < \alpha_2 < \cdots < \alpha_n$.

We will now choose the L_i in such a way that $k \in L_i \not\models \alpha_j = q$ for any non-trivial equation $\alpha_j = q$ with $j \leq i$ and $q \trianglelefteq k$ such that $q \notin \{\alpha_{i+1}, \ldots, \alpha_n\}$. For L_0, this is vacuously true.

For the other L_i, we only need to falsify equations of the form $\alpha_i = q$. Assume that $k \in L_{i-1} \models \alpha_i = q$.

First, consider the case that the equation is of the form $\alpha_i = \alpha_j$ for $j < i$. Choose a term $t \in L$ such that $k = t \models \alpha_i \neq \alpha_j$ and set $L_i = L_{i-1} \cup \{t\}$. This rules out any other equations as well, since if $k \in L_i \models \alpha_i = \alpha_k$, then already $k \in L_{i-1} \models \alpha_j = \alpha_k$, contrary to assumption. If $k \in L_i \models \alpha_i = q$, then $k \in L_{i-1} \models \alpha_j = q$ since $k \in L_0 \models \alpha_i = \alpha_j$.

In the other case, the equation is of the form $\alpha_i = q$ where $q \trianglelefteq k$ is not a non-terminal. Again, choose a term t such that $k = t \models \alpha_i \neq q$. Now, any non-terminal α_j occurring in q has the property that $j < i$, since $\sigma(\alpha_j)$ is a strict subterm of $\sigma(\alpha_i)$. If we have any other equation such that $k \in L_{i-1} \models \alpha_i = r$, then $k \in L_{i-1} \models q = r$, and by Lemma 1.19, there exists an $l < i$ such that $k \in L_{i-1} \models \alpha_l = s$ where $s \trianglelefteq q = r \trianglelefteq k$ only contains non-terminals less than α_i (since both q and r only contained non-terminals less than α_i).

If neither of these cases apply, set $L_i = L_{i-1}$. □

Lemma 1.28. *Let k be a term and L a language. Then there is a term k' in normal form relative to L such that $k \in L \models k = k'$.*

Any such term k' contains no other non-terminals than k. In particular there exists a sequence of equations $k \in L \models \alpha_i = t_i$ such that $\alpha_i, t_i \trianglelefteq k$ for all i, and $k' = k[\alpha_1 \backslash t_1] \cdots [\alpha_n \backslash t_n]$.

The proof of this lemma also explains the name "normal form": The term k' can be seen as the result of applying the rewrite rules $\alpha_i \to t_i$ induced by the equations $k \in L \models \alpha_i = t_i$.

Proof. Let us first define an order where we need to apply each substitution $[\alpha_i \backslash t_i]$ only once: For non-terminals $\alpha, \beta \trianglelefteq k$ set $\alpha < \beta$ if there exists a subterm $t \trianglelefteq k$ such that $k \in L \models \alpha = t$ and $\beta \triangleleft t$. The relation $<$ does not contain any cycles as otherwise $k \in L$ would be unsatisfiable; we can therefore extend $<$ to a total order.

Let $\alpha_1 < \cdots < \alpha_n$ be the list of all non-terminals α_i for which there is a non-trivial equation $\alpha_i = t_i$ with $k \in L \models \alpha_i = t_i$ such that $t_i \trianglelefteq k$, and if t_i is a non-terminal, $\alpha_i < t_i$.

Define $k_i := k[\alpha_1\backslash t_1]\cdots[\alpha_i\backslash t_i]$. Since $k_i \in L \models \alpha_j = t_j$ for each $i < j$, we have $k_i \in L \models k_i = k_j$ for all $i < j$ and in particular $k \in L \models k = k'$.

Assume now towards a contradiction that $k_n \in L \models \alpha = k_n|_p$ for a non-trivial equation $\alpha = k_n|_p$. Since $k \in L \models k = k_n$, we clearly have $k \in L \models \alpha = k_n|_p$. We now want to find a $t \trianglelefteq k$ such that $k \in L \models \alpha = t$ and $\alpha = t$ is non-trivial.

Take the first i such that $p \in \mathrm{Pos}(k_i)$ and $\alpha = k_i|_p$ is non-trivial. If $i = 0$, then we are done. Otherwise $k_i|_p = t_i|_q$ for a position q, and $t_i|_q \trianglelefteq t_i \trianglelefteq k$. We also still have $k \in L \models \alpha = t_i|_q$. □

If a non-terminal occurs in a derivation of a term in a trat grammar, then it always ends up as the same ground term. This is to say, that given a derivation d of a term t in a trat grammar $G = (\tau, N, \Sigma, P)$, the mapping $\delta_d \colon \alpha = t_i|_p \mapsto t|_p$ is actually a function that sends each non-terminal to its eventual ground term. Note that this function can be partial, if a non-terminal $\alpha \in N$ does not occur in the derivation.

This mapping δ_d is a model in the sense of this chapter. In a way, its theory says which productions were used in the derivation: if a production $\alpha \to s \in P$ occurred in the derivation, then clearly $\delta_d \models \alpha = s$.

But what about a non-terminal β that did not occur in the derivation? It will only satisfy the vacuous equality $\delta_d \models \beta = \beta$.

In the end, we would like to characterise the theory of all derivations of a grammar, i.e. find a formula φ_G such that $\delta_d \models \varphi_G$ for all derivations d, and hopefully be able to obtain a derivation from any model $\sigma \models \varphi_G$.

To this end, φ_G should say that a derivation picks one production for each non-terminal, i.e. $\delta_d \models \bigvee_{\alpha\to t\in P} \alpha = t$—but this is not true, and if we add $\alpha = \alpha$ to this disjunction, it becomes valid.

Hence we will need to treat unused non-terminals specially; for this propose introduce a new constant symbol Ω, call this extended signature $\overline{\Sigma} = \Sigma \cup \{\Omega\}$. We can now extend δ_d to $\overline{\delta}_d$ such that $\overline{\delta}_d(\beta) = \Omega$ for unused non-terminals β, and $\overline{\delta}_d(\alpha) = \delta_d(\alpha)$ for α that occur in the derivation.

Definition 1.29. Let $G = (\tau, N, \Sigma, P)$ be a trat grammar, its characteristic formula is the following formula φ_G in the signature $\overline{\Sigma}$:

$$\bigwedge_{\alpha\in N} \left(\alpha = \Omega \vee \bigvee_{\alpha\to t\in P} \alpha = t \right)$$

Example 1.30. Consider the trat grammar G with the non-terminals $\tau < \alpha < \beta$ and the following productions:

$$\begin{aligned} &\tau \to f(\alpha) \mid g(\beta) \\ &\alpha \to h(\beta) \\ &\beta \to a \mid b \end{aligned}$$

This is its characteristic formula:

$$\varphi_G \quad \equiv \quad (\tau = f(\alpha) \vee \tau = g(\beta) \vee \tau = \Omega) \wedge (\alpha = h(\beta) \vee \alpha = \Omega) \wedge (\beta = a \vee \beta = b \vee \beta = \Omega)$$

We can now construct a model from a derivation d of $g(a)$:

$$d = \tau \to^1 g(\beta) \to^1 g(a)$$
$$\overline{\delta}_d \models \tau = g(a) \land \beta = a \land \alpha = \Omega$$

Lemma 1.31. *Let $G = (\tau, N, \Sigma, P)$ be a trat grammar. Then $\overline{\delta}_d \models \varphi_G$ for any derivation d.*

Proof. Let the derivation d be the sequence $\tau = t_1, \ldots, t_n$ where t_n is a ground term. We will prove first of all that $\overline{\delta}_d \models t_i|_p = t_n|_p$ for any term t_i and position p. If $t_i|_p$ is a non-terminal, then $\overline{\delta}_d(t_i|_p) = t_n|_p$ by definition. Otherwise, $t_i|_p = f(r_1, \ldots, r_m)$ and assume $\overline{\delta}_d \models t_i|_{pj} = t_n|_{pj}$ for any $1 \leq j \leq m$. But by congruence of equality this implies $\overline{\delta}_d \models t_i|_p = t_n|_p$ as well.

For any non-terminal α, if $\overline{\delta}_d(\alpha) \neq \Omega$, then α occurs in the derivation and there exist t_i and p such that $t_i|_p = \alpha$ and $t_{i+1}|_p = s$ for a production $\alpha \to s$. By the previous remark, $\overline{\delta}_d \models \alpha = t_i|_p = t_n|_p = t_{i+1}|_p = s$. □

Lemma 1.32. *Let $G = (\tau, N, \Sigma, P)$ be a trat grammar, and σ a model of φ_G such that $\sigma(\tau)$ is a ground term in the signature Σ, i.e. not containing Ω. Then there is a derivation of $\sigma(\tau)$ in G.*

Proof. Assume the non-terminals N are ordered as $\tau = \alpha_0 < \alpha_1 < \cdots < \alpha_n$ such that $\alpha_i < \mathrm{Var}(s)$ for any production $\alpha_i \to s \in P$.

We will iteratively construct derivations $\delta_i = (\alpha_0 \to^1 \cdots \to^1 t_i)$ such that t_i only contains non-terminals α_j with $j \geq i$, and $\sigma \models r = \tau$ for any term in the sequence δ_i. Start with the (empty) derivation $\delta_0 = \alpha_0$.

For δ_{i+1}, consider the derivation $\delta_i = (\alpha_0 \to^1 \cdots \to^1 t_i)$. If t_i does not contain any occurrences of α_i, then do nothing and set $\delta_{i+1} = \delta_i$.

Otherwise, $\sigma(\alpha_i) \neq \Omega$ and we can pick a production $p_i = \alpha_i \to s_i \in P$ such that $\sigma \models \alpha_i = s_i$. Apply the production p_i to all occurrences of α_i in t_i. This results in a derivation $\delta_{i+1} = (\delta_i \to^1 \cdots \to^1 t_i \to^1 \ldots t_{i+1})$.

At the end we have the derivation $\delta_{n+1} = (\alpha_0 \to^1 \cdots \to^1 t_{n+1})$. The term t_{n+1} at the end is ground since it only contains non-terminals α_i with $i \geq n+1$, and it is equal to $\sigma(\tau)$ since $\sigma \models t_{n+1} = \tau$. Furthermore the derivation is rigid, since $\sigma \models r = r' = \sigma(\tau)$ for any terms r and r' in δ_{n+1}. Hence δ_{n+1} is a derivation of $\sigma(\tau)$. □

Corollary 1.33. *Let $G = (\tau, N, \Sigma, P)$ be a trat grammar, and $t \in \mathcal{T}(\Sigma)$ a term, then the following are equivalent:*

1. *$t \in \mathcal{L}(G)$*
2. *$\tau = t \land \varphi_G$ is satisfiable*
3. *$\tau = t \land \varphi_G \land N \subseteq \mathrm{st}(t) \cup \{\Omega\}$ is satisfiable*

Lemma 1.34. *Let G be a trat grammar, $\alpha \to k$ one of its productions, L a language, and k' a term such that $k \in \mathrm{st}(L) \models k = k'$.*

Then $\mathcal{L}(G) \cap L \subseteq \mathcal{L}(G') \cap L$ for the trat grammar G' which is obtained from G by replacing the production $\alpha \to k$ by $\alpha \to k'$.

Proof. We have $\varphi_G \wedge \alpha \in \mathrm{st}(L) \cup \{\Omega\} \models \varphi_{G'}$ by case distinction on $\alpha \neq k \vee \alpha = \Omega \vee (\alpha = k \wedge \alpha \in \mathrm{st}(L))$. Applying Corollary 1.33 finishes the proof. □

Definition 1.35. Let L be a language and $\alpha_1 < \cdots < \alpha_n$ an ordered set of non-terminals. A production $\alpha_i \to k$ is in trat-normal form relative to L if k is in normal form relative to $\mathrm{st}(L)$ and does not contain any non-terminals α_j with $j < i$.

Theorem 1.36. *Let L a language and $G = (\tau, N, \Sigma, P)$ a trat grammar that covers L. Then there is a grammar $G' = (\tau, N, \Sigma, P')$ with $|P'| \leq |P|$ where all productions are in trat-normal form relative to L that still covers L.*

Proof. For each production $p = \alpha \to k \in P$ apply Lemma 1.28 to get a production $p' = \alpha \to k'$ in trat-normal form such that $k \in \mathrm{st}(L) \models k = k'$, and let P' be the set of such productions $\alpha \to k'$.

Successively applying Lemma 1.34 shows that $\mathcal{L}(G) \cap L \subseteq \mathcal{L}(G') \cap L$ and hence $L \subseteq \mathcal{L}(G')$. □

1.2.1 Reduction to propositional logic

A classical result for first-order logic is that we can reduce first-order logic with equality to first-order logic without equality in the sense that there is a formula $\mathrm{Th}_=$ (for a fixed finite language) such that the following are equivalent for any φ, see e.g. [TS00]:

- φ is valid in first-order logic with equality.
- $\mathrm{Th}_= \to \varphi$ is valid in first-order logic without equality, i.e. where $=$ is just a binary relation symbol.

The formula $\mathrm{Th}_=$ is a conjunction of Π_1-axioms for equality, such as $\forall x \forall y\, x = y \to f(x) = f(y)$, etc.

A similar reduction can be used to handle uninterpreted function symbols in SMT solving, but here it is essential that the added axioms for equality are quantifier-free, as quantified formulas are difficult to handle using SMT techniques. One of the reductions that can be used to remove uninterpreted function symbols here is due to Ackermann [Ack62], which reduces quantifier-free formulas with uninterpreted function symbols to equality logic. This is done by adding an axiom $t_1 = s_1 \wedge \cdots \wedge t_n = s_n \to f(t_1, \ldots, t_n) = f(s_1, \ldots, s_n)$ for any subterms $f(t_1, \ldots, t_n)$ and $f(s_1, \ldots, s_n)$ occurring in the formula.

In the following, we will develop a similar reduction from the logic of term algebras to propositional logic.

Definition 1.37. Let φ be a formula in the language of terms. For a term t, write $t \trianglelefteq \varphi$ if t is a subterm of φ, i.e. if it is the subterm of the left or right hand side of an equation in φ.

Define the formula $\mathrm{Th}_{\mathcal{T}}^{\varphi}$ as the conjunction of the following formulas, where $\lhd$ is a new binary relation symbol:

- $f(\bar{t}) = f(\bar{s}) \leftrightarrow \bar{t} = \bar{s}$ for any $f(\bar{t}), f(\bar{s}) \trianglelefteq \varphi$,
- $f(\bar{t}) \neq g(\bar{s})$ for any $f(\bar{t}), g(\bar{s}) \trianglelefteq \varphi$ with different function symbols $f \neq g$,
- $t = t \wedge (t = s \leftrightarrow s = t) \wedge (t = s \wedge s = r \rightarrow t = r)$ for any $t, s, r \trianglelefteq \varphi$,
- $t = t' \wedge s = s' \wedge t \lhd s \rightarrow t' \lhd s'$ for any $t, t', s, s' \trianglelefteq \varphi$,
- $t \not\lhd t \wedge (t \lhd s \wedge s \lhd r \rightarrow t \lhd r)$ for any $t, s, r \trianglelefteq \varphi$,
- $t_i \lhd f(t_1, \ldots, t_n)$ for any $f(t_1, \ldots, t_n) \trianglelefteq \varphi$ and $1 \leq i \leq n$.

The atoms $t \not\lhd s$ and $t \neq s$ are abbreviations of $\neg(t \lhd s)$ and $\neg(t = s)$, respectively.

Remark 1.38. The relation $\lhd$ is included here to prevent the propositional model from accidentally talking about infinite terms. For example the assignment $I \models \alpha = f(\beta) \wedge \beta = g(\gamma) \wedge \gamma = h(\alpha)$ would produce the infinite term $\alpha = f(g(h(f(g(h(\ldots))))))$ without the axioms for $\lhd$.

Another way to prevent this situation without introducing a new relation symbol would be to include the following axioms:

- $t_1 \neq s_1 \vee \cdots \vee t_n \neq s_n$ for any $t_i, s_i \trianglelefteq \varphi$ such that $s_1 \lhd t_2 \wedge \cdots \wedge s_{n-1} \lhd t_n \wedge s_n \lhd t_1$.

But unfortunately, there might be exponentially many axioms of this form, while the version of $\mathrm{Th}_{\mathcal{T}}^{\varphi}$ in Definition 1.37 is always of polynomial size.

Lemma 1.39. *Let φ be a formula, then $(\sigma, \lhd) \models_{\mathcal{T}} \mathrm{Th}_{\mathcal{T}}^{\varphi}$ for any substitution σ where $\lhd$ is interpreted as the real subterm relation.*

Lemma 1.40. *Let φ be a formula, then for any propositional model $I \models_{\mathrm{prop}} \mathrm{Th}_{\mathcal{T}}^{\varphi}$, there exists a substitution σ such that $\sigma \models_{\mathcal{T}} t = s$ if and only if $I \models_{\mathrm{prop}} t = s$ for any $t = s$ occurring in φ.*

Proof. The formula $\mathrm{Th}_{\mathcal{T}}^{\varphi}$ ensures that $\lhd^I$ is a partial order on the equivalence classes induced by $=^I$. Extend this order to a total order $<$ on all subterms of φ such that the $=^I$-equivalence classes are ordered by inverse term size. That means the following: if $I \models s \lhd t$, then $s < t$; and if $I \models \alpha = f(\bar{t})$, then $f(\bar{t}) < \alpha$.

We say that a non-terminal α is defined by the term t if $t < \alpha$ and $I \models \alpha = t$.

Now define a map $\sigma \colon \mathrm{st}(\varphi) \rightarrow \mathcal{T}(\Sigma, X)$ by recursion on $(\mathrm{st}(\varphi), <)$:

- $\sigma(f(t_1, \ldots, t_n)) = f(\sigma(t_1), \ldots, \sigma(t_n))$,
- $\sigma(\alpha) = t$ if α is definable by t,

- $\sigma(\alpha) = \alpha$ if α is not definable.

This map σ is a homomorphism and coincides with the substitution defined by the values $\sigma(\alpha)$ for the non-terminals, hence we will identify σ with this substitution.

Let us now prove by $<$-induction that I and σ agree on the truth value of all equations $r = s$ with $r, s \trianglelefteq \varphi$:

1. If $r = s$ is a trivial equation, then both I and σ satisfy it.
2. If r is definable and we picked the definition t, then both I and σ satisfy $r = t$, and the equation $r = s$ is equivalent to the (smaller) equation $t = s$ in both I and σ. If s is definable, a symmetric argument applies.
3. If $r = f(\dots)$ and $s = g(\dots)$ for different function symbols $f \neq g$, then neither I and σ satisfy $r = s$.
4. If $r = f(r_1, \dots, r_n)$ and $s = f(s_1, \dots, s_n)$, then the equation $r = s$ is equivalent to the conjunction of the (smaller) equations $r_1 = s_1 \wedge \dots \wedge r_n = s_n$ in both I and σ.
5. If r and s are non-terminals and neither is definable, then $I \models r \neq s$, and also $\sigma \models r \neq s$ since $\sigma(r) = r \neq s = \sigma(s)$.
6. If r is a non-terminal and not definable, but $s = f(\dots)$, then $I \models r \neq s$ and also $\sigma \models r \neq s$ since $\sigma(r)$ is a non-terminal but $\sigma(s)$ is not. Up to symmetry, this is the last case. □

Theorem 1.41. *Let φ be a formula, then $\varphi \wedge \mathrm{Th}_{\mathcal{T}}^{\varphi}$ is a polynomial-time computable formula such that the following are equivalent:*

- *φ is satisfiable by a substitution.*
- *$\varphi \wedge \mathrm{Th}_{\mathcal{T}}^{\varphi}$ is satisfiable by a propositional assignment to the atoms $t = s$ and $t \trianglelefteq s$ for any $t, s \trianglelefteq \varphi$.*

Proof. The formula $\mathrm{Th}_{\mathcal{T}}^{\varphi}$ is clearly polynomial-time computable.

If φ is satisfiable by a substitution σ, then $(\sigma, \lhd) \models \mathrm{Th}_{\mathcal{T}}^{\varphi}$ by Lemma 1.39. The propositional assignment I given by $I \models t = s \Leftrightarrow \sigma \models t = s$ and $I \models t \lhd s \Leftrightarrow (\sigma, \lhd) \models t \lhd s$ then satisfies $\varphi \wedge \mathrm{Th}_{\mathcal{T}}^{\varphi}$.

On the other hand, if $I \models \varphi \wedge \mathrm{Th}_{\mathcal{T}}^{\varphi}$, then $\sigma \models \varphi$ as well for the σ given by Lemma 1.40. □

1.2.2 Computing normal forms

Definition 1.42. A term t subsumes a term s, written $t \preceq s$, if there exists a substitution σ such that $t\sigma = s$.

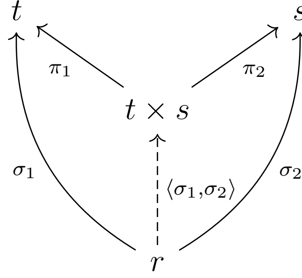
Lemma 1.43. *Any two terms t and s have a minimum $t \times s$ in the subsumption pre-order.*

This minimum $t \times s$ is explicitly given by the following recursive definition where $\alpha_{t,s}$ is a distinct non-terminal for each pair of terms t and s:

$$f(t_1, \dots, t_n) \times f(s_1, \dots, s_n) = f(t_1 \times s_1, \dots, t_n \times s_n)$$
$$t \times s = \alpha_{t,s} \text{ otherwise}$$

Proof. Define the substitutions π_1 and π_2 by $\pi_1(\alpha_{t,s}) := t$ and $\pi_2(\alpha_{t,s}) := s$. Clearly $(t \times s)\pi_1 = t$ and $(t \times s)\pi_2 = s$; this proves that $t \times s$ is a lower bound for t and s.

It remains to prove that any other lower bound of t and s is below $t \times s$ as well; to this end assume that r is another lower bound as well, with $r\sigma_1 = t$ and $r\sigma_2 = s$. Consider now the substitution $\langle\sigma_1, \sigma_2\rangle$ defined by $\langle\sigma_1, \sigma_2\rangle(\alpha) := \alpha\sigma_1 \times \alpha\sigma_2$, we will show that $r\langle\sigma_1, \sigma_2\rangle = t \times s$:



If r is a non-terminal, then this is true by definition. Otherwise, $r = f(r_1, \dots, r_n)$ is a function application; since $r \preceq t$, the term t needs to be of the form $t = f(t_1, \dots, t_n)$ as well, and similarly $s = f(s_1, \dots, s_n)$. Now $t \times s = f(t_1 \times s_1, \dots, t_n \times s_n) = f(r_1\langle\sigma_1, \sigma_2\rangle, \dots, r_n\langle\sigma_1, \sigma_2\rangle) = r\langle\sigma_1, \sigma_2\rangle$ by the induction hypothesis. □

Remark 1.44. The minimum in the subsumption ordering has several names in the literature. In [EH14], it is called "generalized delta vector". Plotkin called it "least general generalization" in [Plo70a; Plo71]. A more modern terminology seems to be "anti-unification" which seems to originate from [Rey70].

Remark 1.45. While any two terms have a minimum, in general they do not need to have a maximum. A maximum of two terms with disjoint sets of non-terminals exists precisely when they are unifiable—in this case the maximum $t + s$ can be computed from the most general unifier $\mathrm{mgu}(t, s)$ as $t + s = t\,\mathrm{mgu}(t, s) = s\,\mathrm{mgu}(t, s)$.

Subsumption is a pre-order and induces an equivalence relation $\approx$ on terms, $t \approx s$ if and only if $t \preceq s$ and $s \preceq t$. Two terms t and s are then equivalent if they only differ by a renaming of their non-terminals—e.g. $f(\alpha, \beta) \approx f(\beta, \alpha)$.

The subsumption lattice $(\mathcal{T}(\Sigma \cup N)/{\approx} \cup \{\Omega\}, +, \times)$ consists of the set of such equivalence classes, together with a new largest element Ω.

This lattice was originally investigated by Plotkin [Plo70b].

Definition 1.46. Let $D \subseteq \mathcal{T}(\Sigma, N)$ be a set of terms. A generalized substitution $\sigma\colon D \to \mathcal{T}(\Sigma, N)$ maps terms from D to terms.

We extend a generalized substitution σ to a function $\overline{\sigma}$ on the full set of terms in the following way:

$$\overline{\sigma}(f(t_1,\dots,t_n)) = \begin{cases} \sigma(f(t_1,\dots,t_n)) \text{ if } f(t_1,\dots,t_n) \in D \\ f(\overline{\sigma}(t_1),\dots,\overline{\sigma}(t_n)) \text{ otherwise} \end{cases}$$

As with substitutions, we write $t\sigma = \overline{\sigma}(t)$ if no confusion arises.

Lemma 1.47. *Let k be a term in normal form relative to the language L, and σ a substitution such that $k\sigma$ subsumes every term in L.*

Then the restriction $\sigma \upharpoonright \mathrm{st}(k)\colon \mathrm{st}(k) \to \mathrm{st}(k)\sigma$ to the subterms of k is injective (and therefore bijective); its inverse is the generalized substitution σ^{-1} defined by $\sigma^{-1}(\sigma(\alpha)) = \alpha$ for any non-terminal α in k.

Proof. Let p and q be positions in k such that $k\sigma|_p = k\sigma|_q$. We would like to show that $k \in L \models k|_p = k|_q$ which would then imply $k|_p = k|_q$.

For any substitution τ such that $\tau \models k \in L$, there is a ρ such that $k\sigma\rho = k\tau \in L$; hence $k\tau|_p = k\sigma|_p\rho = k\sigma|_q\rho = k\tau|_q$ and $\tau \models k|_p = k|_q$.

Let us show $t\sigma\sigma^{-1} = t$ by induction on the subterm $t \in \mathrm{st}(k)$. First consider $t = \alpha$, then $t\sigma\sigma^{-1} = t$ by definition. Otherwise $t = f(t_1,\dots t_n)$ with $t_i\sigma\sigma^{-1} = t_i$. Because σ is injective, $t\sigma \neq \alpha\sigma$ for any non-terminal α, and $f(t_1\sigma,\dots t_n\sigma)\sigma^{-1} = f(t_1\sigma\sigma^{-1},\dots,t_n\sigma\sigma^{-1}) = f(t_1,\dots,t_n)$. □

Theorem 1.48. *Let k be a term in normal form relative to the language L in the non-terminals $\alpha_1,\dots,\alpha_n$, then k can be found using the following algorithm, i.e. there exist L' and σ such that $k = k_{L',\sigma}$.*

1. *Pick a subset $L' \subseteq L$ of size $|L'| \le n+1$.*
2. *Compute the minimum $\prod L'$.*
3. *Pick an injective partial map $\sigma\colon \{\alpha_1,\dots,\alpha_n\} \to \mathrm{st}(\prod L')$.*
4. *Compute $k_{L',\sigma} := (\prod L')\sigma^{-1}$, where the generalized substitution σ^{-1} is defined as in Lemma 1.47.*

Proof. By Theorem 1.27, there is a set $L' \subseteq L$ of size $|L'| \le n+1$ such that k is still in normal form relative to L'. We can assume that k subsumes L', otherwise just take $L'' = \{t \in L' \colon k \preceq t\}$ relative to which k is still in normal form as well.

Since k subsumes L', there is a substitution σ such that $k\sigma = \prod L'$. By Lemma 1.47, we can invert σ using the generalized substitution σ^{-1} and get $k = (\prod L')\sigma^{-1} = k_{L',\sigma}$. □

1.3 Minimizing grammars

Now that we know how small trat grammars can look like, we can shift our focus towards making existing trat grammars smaller.

The problem we treat in this section is the following: given a trat grammar $G = (\tau, N, \Sigma, P)$ that covers a language L, we want to find a smallest (i.e. by number of productions) sub-grammar $G' = (\tau, N, \Sigma, P')$ that still covers L.

This problem translates straightforwardly to a Max-SAT problem which can then be handled by a number of efficient solvers from the SAT community [Arg+08].

Definition 1.49. A partial Max-SAT problem is a propositional CNF formula where some clauses are marked as hard and the others as soft.

A solution is an assignment to the propositional variables that satisfies all the hard clauses, and maximizes the number of satisfied soft clauses.

We will encode $\mathcal{L}(G') \supseteq L$ as hard clauses, and use the soft clauses to minimize the number of productions.

Lemma 1.50. *Let $G = (\tau, N, \Sigma, P)$ be a trat grammar, and t a term. There exists a propositional formula $\psi_{G,t}$ such that the following are equivalent for a sub-grammar $G' = (\tau, N, \Sigma, P') \subseteq G$:*

- *There is a propositional assignment I satisfying $\psi_{G,t}$ such that $P' = \{p \in P\colon I \models p \in P'\}$. (The formula $p \in P'$ is a different atom for each production $p \in P$.)*
- *The term t is derivable in G'.*

Proof. Define $\psi_{G,t}$ as the following formula:

$$\tau = t \wedge \bigwedge_{\alpha \in N} \left(\alpha = \Omega \vee \bigvee_{\alpha \to s \in P} (\alpha \to s \in P' \wedge \alpha = s) \right) \wedge \mathrm{Th}_{\mathcal{T}}$$

By Corollary 1.33, the assertion $t \in \mathcal{L}(G')$ is equivalent to the satisfiability of $\tau = t \wedge \varphi_{G'}$. For a fixed $P' \subseteq P$, this formula is equivalent to $\psi_{G,t}$. Theorem 1.41 then shows the equi-satisfiability in propositional logic. □

Remark 1.51. The formula $\mathrm{Th}_{\mathcal{T}}$ can be simplified considerably in this case. Since the equations come from the productions of an acyclic grammar, we can safely drop the axioms for the relation $\lhd$ which otherwise prevent infinite terms.

In general we can eliminate the definitions for equality of compound terms, and reflexivity and symmetry of equality. This leaves us with equations of only the form $\alpha = t$ and consequences of the transitivity of equality $\bigwedge_i \alpha_i = t_i \to \bigwedge_j \beta_j = s_j$, and $\bigwedge_i \alpha_i = t_i \to \bot$.

For this particular problem however, we can do even better: if $\psi_{G,t}$ is satisfiable, then by Corollary 1.33 there is a model which only assigns subterms of t or Ω to non-terminals. In such a model, we only need to define equality relative to ground terms—it suffices to include the following axioms:

- $\alpha = s \to (\alpha = k \leftrightarrow \bigwedge_i \beta_i = r_i)$ and $\alpha = k \wedge \bigwedge_i \beta_i = r_i \to \alpha = s$, where the term s is a subterm of t or Ω, the equations $\alpha = k$ occurs in $\psi_{G,t}$, and $s = k[\beta_1 \backslash r_1, \dots, \beta_n \backslash r_n]$.
- $\alpha = s \to \alpha \neq k$, where the term s is a subterm of t or Ω, the equations $\alpha = k$ occurs in $\psi_{G,t}$, and $k \not\preceq s$.
- $\alpha = s \to \alpha \neq r$, where the terms $r \neq s$ are subterms of t or Ω.

The resulting formula then has $\alpha = \Omega \vee \bigvee_{s \trianglelefteq t} \alpha = s$ as a consequence for each α (this is due to φ_G), and the substitution obtained by setting $\sigma(\alpha) = s$ if $\alpha = s$ is satisfied for the unique ground term s is indeed a model.

Corollary 1.52. *Let $G = (\tau, N, \Sigma, P)$ be a trat grammar, and L a language. There exists a propositional formula $\psi_{G,L}$ such that the following are equivalent for a sub-grammar $G' = (\tau, N, \Sigma, P) \subseteq G$:*

- *There is an assignment I satisfying $\psi_{G,L}$ such that $P' = \{p \in P \colon I \models p \in P'\}$.*
- *G' covers L.*

Proof. Take $\psi_{G,L} \equiv \bigwedge_{t \in L} \psi_{G,t}$, where the (propositional) variables other than $p \in P$ are renamed to be different in each $\psi_{G,t}$. □

Corollary 1.53. *Let $G = (\tau, N, \Sigma, P)$ be a trat grammar, and L a language. There exists a Max-SAT problem $\tilde{\psi}_{G,L}$ such that the following are equivalent for a sub-grammar $G' = (\tau, N, \Sigma, P') \subseteq G$:*

- *There is a solution I to $\tilde{\psi}_{G,L}$ such that $P' = \{p \in P \colon I \models p \in P'\}$.*
- *G' is a smallest sub-grammar of G covering L.*

Proof. Convert the formula $\psi_{G,L}$ into CNF. The formula $\tilde{\psi}_{G,L}$ consists of the clauses of $\psi_{G,L}$ (which are hard), and for each $p \in P$ a soft clause $\neg(p \in P')$. The number of satisfied soft clauses is equal to $|P| - |P'|$, therefore a solution that satisfies the most soft clauses corresponds to a grammar G' with the smallest number of productions. □

1.4 Finding a minimal grammar

For this section fix a language L, a signature Σ and a set of non-terminals $N = \{\alpha_0, \dots, \alpha_n\}$. We want to find a trat grammar $G = (\alpha_0, N, \Sigma, P)$ of minimal size that covers L.

Lemma 1.54. *There is a trat grammar $H = (\alpha_0, N, \Sigma, Q)$ that contains a grammar of minimal size covering L as a sub-grammar.*

This grammar can be computed by an algorithm whose runtime is polynomially bounded in $|\mathrm{st}(L)|$ for a fixed n. The number of productions in H and the number of subterms of right-hand sides of productions in H is then also polynomially bounded for each n.

Proof. First compute the set K of normal forms relative to the language $\mathrm{st}(L)$ using Theorem 1.48. The number of $(n+2)$-element subsets $L' \subseteq \mathrm{st}(L)$ we need to consider here is polynomially bounded in $|\mathrm{st}(L)|$; for each subset L', the term $\prod L'$ can be computed linearly in the size of $\mathrm{st}(L')$ and has size less than $|\mathrm{st}(L)|$, hence the number of maps σ is polynomially bounded by $|\mathrm{st}(L)|$ as well. Inverting σ is polynomial-time computable as well.

For each non-terminal α_i and $k \in K$, add the production $\alpha_i \to k$ to Q if $\alpha_i < \mathrm{Var}(k)$. This is polynomial-time as well.

The grammar H consists now of all productions in trat-normal form, and contains a grammar of minimal size covering L by Theorem 1.36. □

Theorem 1.55. *There is a polynomial-time computable Max-SAT problem such that any solution I gives a grammar of minimal size covering L by setting $P := \{p \in Q \colon I \models p \in P\}$ where $p \in P$ is a different propositional variable for each $p \in Q$.*

The runtime is polynomially bounded in $|\mathrm{st}(L)|$ for fixed n.

Proof. Compute the grammar H as in Lemma 1.54 and then use Corollary 1.53 to produce the Max-SAT problem; Corollary 1.53 needs to convert a formula into an equisatisfiable CNF formula—this is polynomial-time computable due to [Tse83]. □

2 Simple proofs

2.1 Sequent calculus

In [Gen35], Gentzen introduced the sequent calculus LK as a proof system for classical first-order logic. In this section we will give basic definitions, and state classic results. For more details, the reader is deferred to an introduction to proof theory such as [TS00].

Classical first-order logic deals with the validity of formulas, such as e.g. $\forall x\, R(f(x))$, intuitively meaning that the relation R is true for the function f applied to any x in some unspecified universe. We will first define which of these relation and function symbols such as R and f may occur in a formula:

Definition 2.1. A (first-order) signature $\Sigma = (\Sigma_r, \Sigma_t)$ is pair of disjoint sets Σ_r of relational symbols, and Σ_t of function symbols, where each symbol is associated with a non-negative integer n, its arity.

In the example above, R was a unary (i.e. 1-ary) relation symbol, f was a unary function symbol; meaning that they each take one argument.

Let us now give a concrete description of the formulas we are considering:

Definition 2.2. Let X be a set of variables, and Σ a signature.

Atoms are equations $t_1 = t_2$, or relations $R(t_1, \ldots, t_n)$ for $R \in \Sigma_r$ and $t_i \in \mathcal{T}(\Sigma_t, X)$.

The set of formulas is the closure of the set atoms under propositional connectives and quantifiers, and contains the following:

- φ for any atom φ,
- $\varphi \land \psi$ for any formulas φ and ψ,
- $\varphi \lor \psi$ for any formulas φ and ψ,
- $\varphi \to \psi$ for any formulas φ and ψ,
- $\neg\varphi$ for any formula φ,
- $\forall x\, \varphi$ for any formula φ and variable x,
- $\exists x\, \varphi$ for any formula φ and variable x.

The proof system LK does not simply prove formulas; it proves sequents:

Definition 2.3. A sequent $\Gamma \vdash \Delta$ is a pair of multisets of formulas Γ and Δ. We call Γ the antecedent, and Δ the succedent.

A sequent such as $\varphi_1, \varphi_2, \varphi_3 \vdash \psi_1, \psi_2$ has the intended meaning $(\varphi_1 \wedge \varphi_2 \wedge \varphi_3) \to (\psi_1 \vee \psi_2)$; i.e. that the conjunction of the antecedent implies the disjunction of the succedent.

Definition 2.4. Let $\Gamma \vdash \Delta$ by a sequent. An LK-proof of $\Gamma \vdash \Delta$ is a tree composed of the rules in Figure 2.1 ending in $\Gamma \vdash \Delta$, this sequent is then called the end-sequent of the proof.

$$\text{for } \varphi \text{ atomic: } \varphi \vdash \varphi \quad \mathrm{Ax} \qquad\qquad \bot \vdash \quad \bot_l$$

$$\frac{\Gamma \vdash \Delta}{\varphi, \Gamma \vdash \Delta}\, w_l \qquad\qquad \frac{\Gamma \vdash \Delta}{\Gamma \vdash \Delta, \varphi}\, w_r$$

$$\frac{\varphi, \varphi, \Gamma \vdash \Delta}{\varphi, \Gamma \vdash \Delta}\, c_l \qquad\qquad \frac{\Gamma \vdash \Delta, \varphi, \varphi}{\Gamma \vdash \Delta, \varphi}\, c_r$$

$$\frac{\varphi, \psi, \Gamma \vdash \Delta}{\varphi \wedge \psi, \Gamma \vdash \Delta}\, \wedge_l \qquad\qquad \frac{\Gamma \vdash \varphi, \Delta \qquad \Gamma \vdash \psi, \Delta}{\Gamma \vdash \Delta, \varphi \wedge \psi}\, \wedge_r$$

$$\frac{\varphi, \Gamma \vdash \Delta \qquad \psi, \Gamma \vdash \Delta}{\varphi \vee \psi, \Gamma \vdash \Delta}\, \vee_l \qquad\qquad \frac{\Gamma \vdash \Delta, \varphi, \psi}{\Gamma \vdash \Delta, \varphi \vee \psi}\, \vee_r$$

$$\frac{\Gamma \vdash \Delta, \varphi \qquad \psi, \Gamma \vdash \Delta}{\varphi \to \psi, \Gamma \vdash \Delta}\, \to_l \qquad\qquad \frac{\varphi, \Gamma \vdash \Delta, \psi}{\Gamma \vdash \Delta, \varphi \to \psi}\, \to_r$$

$$\frac{\varphi[x\backslash t], \Gamma \vdash \Delta}{\forall x\, \varphi, \Gamma \vdash \Delta}\, \forall_l \qquad\qquad \frac{\Gamma \vdash \Delta, \varphi[x\backslash \xi] \qquad (*)}{\Gamma \vdash \Delta, \forall x\, \varphi}\, \forall_r$$

$$\frac{\varphi[x\backslash \xi], \Gamma \vdash \Delta \qquad (*)}{\exists x\, \varphi, \Gamma \vdash \Delta}\, \exists_l \qquad\qquad \frac{\Gamma \vdash \Delta, \varphi[x\backslash t]}{\Gamma \vdash \Delta, \exists x\, \varphi}\, \exists_r$$

$$\frac{\Gamma \vdash \Delta, \varphi \qquad \varphi, \Gamma \vdash \Delta}{\Gamma \vdash \Delta}\, \mathrm{Cut}$$

$$\vdash t = t \quad =_{\mathrm{Ax}} \qquad \frac{\Gamma[t], s = t \vdash \Delta[t]}{\Gamma[s], s = t \vdash \Delta[s]} =_1 \qquad \frac{\Gamma[s], s = t \vdash \Delta[s]}{\Gamma[t], s = t \vdash \Delta[t]} =_2$$

Figure 2.1: The system **LK**, including cut and equality rules; where $(*)$ is the eigenvariable condition: in the indicated two rules, ξ may not occur freely in Γ, Δ or φ.

If we look at these rules, all rules except the quantifier rules, cut rule, and equation rules have one important property: All formulas occurring on the top of the rule are subformulas of formulas on the bottom:

Lemma 2.5 (Subformula property)**.** *Let π be an LK-proof of $\Gamma \vdash \Delta$, not containing the cut, quantifier, or equation rules. Then any formula occurring in any sequent in π is a subformula of a formula in the end-sequent.*

For proofs containing quantifier or equation rules, a weaker version of the subformula property holds: in such proofs, every formula $\varphi[\overline{t}]$ is a subformula of a formula $\psi[\overline{s}]$ in the end-sequent or an equation, i.e. they are subformulas modulo term substitution.

Let us finally state that LK indeed proves exactly the valid formulas of classical first-order logic:

Theorem 2.6. *LK is sound and complete for classical first-order logic.*

2.2 Extended Herbrand sequents

In this section we will generalize Herbrand sequents to so-called extended Herbrand sequents; Herbrand's theorem states that provable sequents have Herbrand sequents, to state the special case we are interested in, we first need to give a few definitions:

Definition 2.7. Formulas of the form $\forall x_1 \ldots \forall x_i\, \varphi$ with φ quantifier-free are called Π_1 formulas; formulas of the form $\exists x_1 \ldots \exists x_i\, \varphi$ with φ quantifier-free are called Σ_1 formulas.

A sequent $\Gamma \vdash \Delta$ is called Σ_1 if all formulas in Γ are Π_1 and all formulas in Δ are Π_1.

Definition 2.8. An instance of a Π_1 formula $\forall x_1 \ldots \forall x_i\, \varphi$ with terms $t_1, \ldots, t_i$ is the formula $\varphi[x_1 \backslash t_1, \ldots, x_i \backslash t_i]$.

An instance of a Σ_1 formula $\exists x_1 \ldots \exists x_i\, \varphi$ with terms $t_1, \ldots, t_i$ is given by the formula $\varphi[x_1 \backslash t_1, \ldots, x_i \backslash t_i]$.

Definition 2.9. A sequent $\Gamma \vdash \Delta$ is called a tautology if it is provable in LK without the equality rules, and a quasi-tautology if it is provable in LK.

Definition 2.10. Let $\Gamma \vdash \Delta$ be a Σ_1-sequent. A Herbrand-sequent for $\Gamma \vdash \Delta$ is a quasi-tautological sequent $\Gamma' \vdash \Delta'$ where Γ' and Δ' are sets of instances of formulas in Γ and Δ, respectively.

Theorem 2.11 (Herbrand's theorem)**.** *Every valid Σ_1-sequent has a Herbrand sequent.*

Example 2.12. The sequent $P(c), \forall x\, P(x) \to P(f(x)) \vdash P(f(f(c)))$ is Σ_1. One possible Herbrand sequent is $P(c), P(c) \to P(f(c)), P(f(c)) \to P(f(f(c))) \vdash P(f(f(c)))$.

Note that there are many possible Herbrand sequents. If $\Gamma' \vdash \Delta'$ is an Herbrand sequent, then $\Gamma', \Gamma'' \vdash \Delta', \Delta''$ is one as well. This is an important observation: in the following, we will describe Herbrand sequents by grammars, and due to this fact we will not need to find a grammar generating the exact set of formulas of a Herbrand sequent, but only a superset of it.

The following theorem is due to Gentzen [Gen35]; it is for this theorem that he introduced sequent calculus:

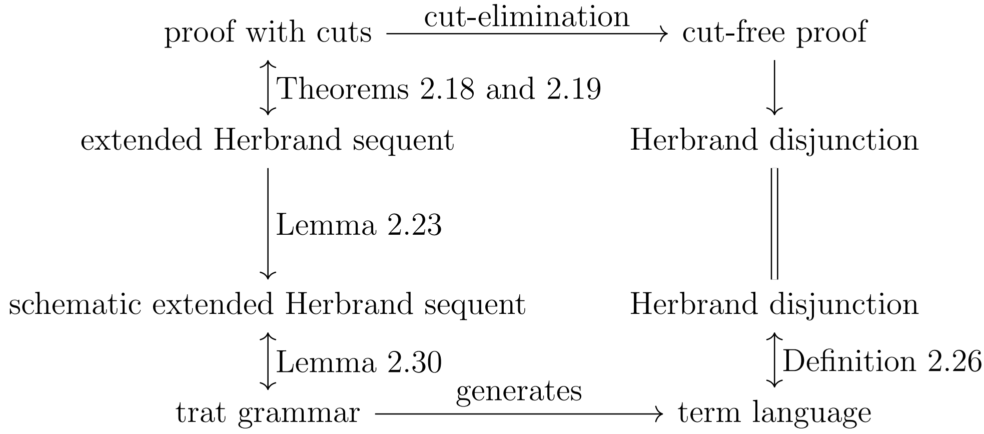


Figure 2.2: Simple proofs with cut and their grammars.

Theorem 2.13 (Cut-elimination)**.** *Let π be a proof in LK of the sequent $\Gamma \vdash \Delta$. Then there is a proof π^* in LK without the cut-rule of this sequent.*

If we have a cut-free proof of a Σ_1-sequent, then we can extract an Herbrand sequent from it by only looking at the quantifier rules in the proofs—they will provide all needed instances. We will now extend this extraction method to a larger class of proofs including restricted usage of the cut rule. In this case we will not directly get Herbrand sequents, but a generalization called extended Herbrand sequents.

Definition 2.14. An LK-proof is called regular if all eigenvariables in strong quantifier inferences are distinct.

Definition 2.15. A simple cut is a cut inference where the cut formula is of the form $\forall x\, \varphi$ where φ is quantifier-free.

A simple proof is an LK-proof where all cut inferences are simple.

Definition 2.16. Let $\Gamma \vdash \Delta$ be a Σ_1-sequent. An extended Herbrand-sequent for $\Gamma \vdash \Delta$ is a quasi-tautological sequent $\Gamma', \Pi \vdash \Delta'$ of the following form:

$$\Gamma', \underbrace{\varphi_1 \to \bigwedge_j \varphi_1[\alpha_1 \backslash t_{1,j}], \ldots, \varphi_n \to \bigwedge_j \varphi_n[\alpha_n \backslash t_{n,j}]}_{\Pi} \vdash \Delta'$$

- $\alpha_1, \ldots, \alpha_n$ are variables.
- Γ' consists of instances of formulas in Γ, possibly containing the new variables α_i.
- Each φ_i is a quantifier-free formula containing only variables α_k for $k \geq i$.
- Each term $t_{i,j}$ only contains variables α_k for $k > i$.
- Δ' consists of instances of formulas in Δ, possibly containing the new variables α_i.

Example 2.17. Consider the following proof π of $P(c), \forall x\, P(x) \to P(f(x)) \vdash P(f^4(c))$ (where the subproofs ψ and ψ' are cut-free):

$$\dfrac{\dfrac{\dfrac{[\psi]}{P(\alpha) \to P(f(\alpha)), P(f(\alpha)) \to P(f^2(\alpha)) \vdash P(f^2(\alpha))}}{\dfrac{\forall x\, P(x) \to P(f(x)) \vdash P(f^2(\alpha))}{\forall x\, P(x) \to P(f(x)) \vdash \forall x\, P(x) \to P(f^2(x))}} \quad \dfrac{\dfrac{[\psi']}{P(c), P(c) \to P(f^2(c)), P(f^2(c)) \to P(f^4(c)) \vdash P(f^4(c))}}{P(c), \forall x\, P(x) \to P(f^2(x)) \vdash P(f^4(c))}}{P(c), \forall x\, P(x) \to P(f(x)) \vdash P(f^4(c))}$$

We will assign to this proof the following extended Herbrand sequent:

$$P(c), P(\alpha_1) \to P(f(\alpha_1)), P(f(\alpha_1)) \to P(f^2(\alpha_1)),$$
$$(P(\alpha_1) \to P(f^2(\alpha_1))) \to (P(c) \to P(f^2(c))) \wedge (P(f^2(c)) \to P(f^4(c))) \vdash P(f^4(c))$$

With the notation from Definition 2.16, we have:

$$\begin{aligned}
\Gamma' &= \{P(c), P(\alpha_1) \to P(f(\alpha_1)), P(f(\alpha_1)) \to P(f^2(\alpha_1))\} \\
\Delta' &= \{P(f^4(c))\} \\
\varphi_1 &= P(\alpha_1) \to P(f^2(\alpha_1)) \\
t_{1,1} &= c \\
t_{1,2} &= f^2(c)
\end{aligned}$$

Theorem 2.18. *There is mapping Φ that transforms simple proofs of Σ_1-sequents into cut-free proofs of an extended Herbrand sequent of those Σ_1-sequents.*

Proof. We will recursively define this mapping Φ:

$$\Phi\left(\begin{array}{c}[\pi] \\ \Gamma \vdash \Delta\end{array}\right) = \begin{array}{c}[\Phi\pi] \\ \Gamma', \Pi \vdash \Delta'\end{array}$$

Note that due to the sub-formula property, all occurring formulas are in prenex form without quantifier alternations. Axioms already end in extended Herbrand sequents. Weakenings, contractions, and quantifier rules are absorbed into the extended Herbrand sequent:

$$\Phi\left(\dfrac{\begin{array}{c}[\pi] \\ \Gamma \vdash \Delta\end{array}}{\varphi, \Gamma \vdash \Delta}\, w_l\right) = \begin{array}{c}[\Phi\pi] \\ \Gamma', \Pi \vdash \Delta'\end{array}$$

$$\Phi\left(\dfrac{\begin{array}{c}[\pi] \\ \varphi, \varphi, \Gamma \vdash \Delta\end{array}}{\varphi, \Gamma \vdash \Delta}\, c_l\right) = \begin{array}{c}[\Phi\pi] \\ \varphi'_1, \ldots, \varphi'_i, \Gamma', \Pi \vdash \Delta'\end{array}$$

$$\Phi\left(\dfrac{\begin{array}{c}[\pi] \\ \forall x_2 \ldots \forall x_j\, \varphi[x_1 \backslash t], \Gamma \vdash \Delta\end{array}}{\forall x_1 \ldots \forall x_j\, \varphi, \Gamma \vdash \Delta}\, \forall_l\right) = \begin{array}{c}[\Phi\pi] \\ \varphi'_1, \ldots, \varphi'_i, \Gamma', \Pi \vdash \Delta'\end{array}$$

In these three cases, the resulting sequent of the transformed subproof is already an extended Herbrand sequent and we are done. The same is true for w_r, c_r, $\forall_r$, $\exists_l$, and $\exists_r$.

For the propositional rules remember that all formulas are in prenex form and hence subformulas of conjunctions, disjunctions, and implications are always quantifier-free. Hence we only need to introduce contractions or weakening to make sure these subformulas occur exactly once:

$$\Phi\left(\dfrac{\genfrac{}{}{0pt}{}{[\pi]}{\varphi, \psi, \Gamma \vdash \Delta}}{\varphi \wedge \psi, \Gamma \vdash \Delta}\wedge_l\right) = \dfrac{\dfrac{\genfrac{}{}{0pt}{}{[\Phi\pi]}{\varphi, \ldots, \varphi, \psi, \ldots, \psi, \Gamma', \Pi \vdash \Delta'}}{\varphi, \psi, \Gamma', \Pi \vdash \Delta'}c_l, w_l}{\varphi \wedge \psi, \Gamma' \vdash \Delta'}\wedge_l$$

$$\Phi\left(\dfrac{\genfrac{}{}{0pt}{}{[\pi_1]}{\Gamma \vdash \Delta, \varphi} \quad \genfrac{}{}{0pt}{}{[\pi_2]}{\Gamma \vdash \Delta, \psi}}{\Gamma \vdash \Delta, \varphi \wedge \psi}\wedge_r\right) = \dfrac{\dfrac{\genfrac{}{}{0pt}{}{[\Phi\pi_1]}{\Gamma_1', \Pi_1 \vdash \Delta_1', \varphi, \ldots, \varphi}}{\Gamma_1', \Gamma_2', \Pi_1, \Pi_2 \vdash \Delta_1', \Delta_2', \varphi}w, c \quad \dfrac{\genfrac{}{}{0pt}{}{[\Phi\pi_2]}{\Gamma_2', \Pi_2 \vdash \Delta_2', \psi, \ldots, \psi}}{\Gamma_1', \Gamma_2', \Pi_1, \Pi_2 \vdash \Delta_1', \Delta_2', \psi}w, c}{\Gamma_1', \Gamma_2', \Pi_1, \Pi_2 \vdash \Delta_1', \Delta_2', \varphi \wedge \psi}\wedge_r$$

The other connectives can be handled similarly. For the equality rules we might again need to reintroduce the equality via weakening:

$$\Phi\left(\dfrac{\genfrac{}{}{0pt}{}{[\pi]}{\Gamma[t], s = t \vdash \Delta[t]}}{\Gamma[s], s = t \vdash \Delta[s]}=_1\right) = \dfrac{\dfrac{\genfrac{}{}{0pt}{}{[\Phi\pi]}{\Gamma'[t], s = t, \ldots, s = t, \Pi \vdash \Delta'[t]}}{\Gamma'[t], s = t, \Pi \vdash \Delta'[t]}c_l, w_l}{\Gamma'[s], s = t, \Pi \vdash \Delta'[s]}=_1$$

Cuts are the only rules for which we add formulas to Π, each formula in Π will describe the cut formula and its instances in the proof. Consider now a simple cut:

$$\pi = \dfrac{\genfrac{}{}{0pt}{}{[\pi_1]}{\Gamma \vdash \Delta, \forall x\, \varphi[x]} \quad \genfrac{}{}{0pt}{}{[\pi_2]}{\forall x\, \varphi[x], \Gamma \vdash \Delta}}{\Gamma \vdash \Delta}\text{Cut}$$

In general, the transformation of the left subproof can end in multiple instances of $\forall x\, \varphi[x]$, but these are always instantiated by an eigenvariable; say these instances are $\varphi[\alpha_1], \ldots, \varphi[\alpha_k]$. We can then use the substitution $\sigma = [\alpha_1 \backslash \alpha, \ldots, \alpha_k \backslash \alpha]$for a fresh variable α to unify these instances, and then contract away the extra instances. So if $\Phi\pi_1$ ends with the sequent $\Gamma_1', \Pi_1' \vdash \Delta_1', \varphi[\alpha_1], \ldots, \varphi[\alpha_k]$, then $\Phi\pi$ is given by:

$$\Phi\pi = \dfrac{\dfrac{\genfrac{}{}{0pt}{}{[(\Phi\pi_1)\sigma]}{\Gamma_1', \Pi_1' \vdash \Delta_1', \varphi[\alpha], \ldots, \varphi[\alpha]}}{\Gamma_1', \Pi_1' \vdash \Delta_1', \varphi[\alpha]}c_r, w_r \quad \dfrac{\genfrac{}{}{0pt}{}{[\Phi\pi_2]}{\varphi[t_1], \ldots, \varphi[t_n], \Gamma_2', \Pi_2' \vdash \Delta_2'}}{\bigwedge_j \varphi[t_j], \Gamma_2', \Pi_2' \vdash \Delta_2'}\wedge_l}{\Gamma_1', \Gamma_2', \Pi_1', \Pi_2', \varphi[\alpha] \to \bigwedge_j \varphi[t_j] \vdash \Delta_1', \Delta_2'}\to_l$$

It remains to verify that the end-sequents produced this way are indeed extended Herbrand sequents. In each case we only introduced instances that can occur in an extended Herbrand sequent; in the cut rule we introduced a formula in Π—we have to

verify that the eigenvariables can be ordered in such a way as to satisfy the free variable restrictions in the definition of an extended Herbrand sequent.

For an eigenvariable α in the end-sequent of $\Phi\pi$, let $C(\alpha)$ be the corresponding cut inference in π. Order now two eigenvariables $\alpha \leq \beta$ if and only if $C(\alpha)$ occurs before $C(\beta)$ in an in-order traversal of the proof tree π, i.e. where recursively for each inference we process first the left (or only) subproof (if it exists), then the inference itself, and then the right subproof (again, if it exists). Visually this means that $\alpha < \beta$ if we draw $C(\alpha)$ to the left of $C(\beta)$.

For convenience, we will identify eigenvariables in the original proof π with the eigenvariables that replaced them in the end-sequent of $\Phi\pi$. Eigenvariables are then only introduced by strong quantifier rules on the left side of their corresponding cuts, i.e.for any occurrence of an eigenvariable α, the cut $C(\alpha)$ is to the right.

Put differently, a formula can only contain eigenvariables whose cut is to the right; in particular a cut formula $\forall x\, \varphi[x]$ for the cut with eigenvariable α can only contain variables β with $\beta > \alpha$. Hence for $\varphi[\alpha] \to \bigwedge_j \varphi[t_j] \in \Pi$, the formula $\varphi[\alpha]$ can only contain the eigenvariable α or any variable β with $\beta > \alpha$.

A term t_j can only be introduced in a weak quantifier inference on the right side of $C(\alpha)$, and since it can only contain variables β whose cut is even further right, we have $\beta > \alpha$ again. □

Theorem 2.19. *Let $H = \Gamma', \varphi_1 \to \bigwedge_j \varphi_1[\alpha_1 \backslash t_{1,j}], \ldots, \varphi_n \to \bigwedge_j \varphi_n[\alpha_n \backslash t_{n,j}] \vdash \Delta'$ be an extended Herbrand sequent of the Σ_1-sequent $\Gamma \vdash \Delta$.*

Then there is a simple proof π of $\Gamma \vdash \Delta$ with n cuts and the same quantifier instantiations as given in H.

Proof. We cannot directly invert the construction in Theorem 2.18, since a cut-free proof π^* of H might not have a linear structure, i.e. it could be the case that the $\to_l$ rule is applied twice to the same implication $\varphi_i \to \bigwedge_j \varphi_i[\alpha_i \backslash t_{i,j}]$ in different subtrees, giving two cuts.

Using Craig's interpolation theorem it is possible to find stronger formulas φ_i' for which such a cut-free proof with linear structure exists. The details are given in [Het+14b]. □

One application of extended Herbrand sequents is cut-introduction: given a cut-free proof, we would like to find a proof of the same end-sequent, but with cuts. By Theorem 2.19 we can reduce this problem to finding an extended Herbrand sequent. We will accomplish this by extracting the Herbrand sequent of the cut-free proof and then finding a grammar (of a suitable form) that covers this Herbrand sequent. But this will only tell us about the variables α_i (corresponding to the non-terminals), and the terms $t_{i,j}$ (corresponding to the productions); there will be no information about the cut-formulas φ_i in this grammar. Hence we will define the notion of a *schematic* extended Herbrand sequent, which is basically an extended Herbrand sequent without cut-formulas, and then prove that we can always find such cut-formulas.

Definition 2.20. Let $\Gamma \vdash \Delta$ be a Σ_1-sequent. A schematic extended Herbrand-sequent for $\Gamma \vdash \Delta$ is a sequent $\Gamma', \Pi \vdash \Delta'$ of the following form:

$$\underbrace{\Gamma', X_1(\alpha_1) \to \bigwedge_j X_1(t_{1,j}), \ldots, X_n(\alpha_n) \to \bigwedge_j X_n(t_{n,j})}_{\Pi} \vdash \Delta'$$

- $\alpha_0, \ldots, \alpha_{n-1}$ are variables.
- Γ' consists of instances of formulas in Γ, possibly containing the new variables α_i.
- Each X_i is a unary second order predicate variable.
- Each term $t_{i,j}$ only contains variables α_k for $k > i$.
- Δ' consists of instances of formulas in Δ, possibly containing the new variables α_i.
- The sequent $\bigwedge_{j_1,\ldots,j_n}(\Gamma' \wedge \neg\Delta')[\alpha_1 \backslash t_{1,j_1}] \ldots [\alpha_n \backslash t_{n,j_n}] \vdash$ is a quasi-tautology.

The last condition implies that the following sequent is a Herbrand sequent for $\Gamma \vdash \Delta$:

$$\bigcup_{j_1,\ldots,j_n} \Gamma'[\alpha_1 \backslash t_{1,j_1}] \ldots [\alpha_n \backslash t_{n,j_n}] \vdash \bigcup_{j_1,\ldots,j_n} \Delta'[\alpha_1 \backslash t_{1,j_1}] \ldots [\alpha_n \backslash t_{n,j_n}]$$

Theorem 2.21. *Let $\Gamma', \varphi_1 \to \bigwedge_j \varphi_1[\alpha_1 \backslash t_{1,j}], \ldots, \varphi_n \to \bigwedge_j \varphi_n[\alpha_n \backslash t_{n,j}] \vdash \Delta'$ be an extended Herbrand sequent for $\Gamma \vdash \Delta$, then $\Gamma', X_1(\alpha_1) \to \bigwedge_j X_1(t_{1,j}), \ldots, X_n(\alpha_n) \to \bigwedge_j X_n(t_{n,j}) \vdash \Delta'$ is a schematic extended Herbrand sequent.*

Proof. The crucial part of this proof is to show that $\bigcup_{j_1,\ldots,j_n} \Gamma'[\alpha_1 \backslash t_{1,j_1}] \ldots [\alpha_n \backslash t_{n,j_n}] \vdash \bigcup_{j_1,\ldots,j_n} \Delta'[\alpha_1 \backslash t_{1,j_1}] \ldots [\alpha_n \backslash t_{n,j_n}]$ is a Herbrand sequent for $\Gamma \vdash \Delta$. This can be shown by following the cut-elimination process, for details see [Het12]. □

Definition 2.22. A solution for a schematic extended Herbrand sequent H is a substitution σ that assigns to each $X_i(\alpha_i)$ a quantifier-free formula in such a way that only the variables $\alpha_{i+1}, \ldots, \alpha_n$ may occur free in $\sigma(X_i)$, and $H\sigma$ is a quasi-tautological sequent.

These two conditions imply that whenever σ is a solution of a schematic extended Herbrand sequent H, then $H\sigma$ is an extended Herbrand sequent.

Lemma 2.23. *Let $H = \Gamma', \varphi_1 \to \bigwedge_j \varphi_1[\alpha_1 \backslash t_{1,j}], \ldots, \varphi_n \to \bigwedge_j \varphi_n[\alpha_n \backslash t_{n,j}] \vdash \Delta'$ be an extended Herbrand sequent of the Σ_1-sequent $\Gamma \vdash \Delta$.*

Then $\sigma\colon X_i(\alpha_i) \mapsto \varphi_i$ is a solution of the schematic extended Herbrand sequent $H' = \Gamma', X_1(\alpha_1) \to \bigwedge_j X_1(t_{1,j}), \ldots, X_n(\alpha_n) \to \bigwedge_j X_n(t_{n,j}) \vdash \Delta'$ of $\Gamma \vdash \Delta$, and $H'\sigma = H$.

Proof. The sequent H' is indeed a schematic extended Herbrand sequent, and valid since clearly $H'\sigma = H$. □

Definition 2.24. Let $H = \Gamma', X_1(\alpha_1) \to \bigwedge_j X_1(t_{1,j}), \dots X_n(\alpha_n) \to \bigwedge_j X_n(t_{n,j}) \vdash \Delta'$ be a schematic extended Herbrand sequent for $\Gamma \vdash \Delta$.

The canonical solution for H is the substitution given by $X_i(\alpha_i) \mapsto C_i$ where C_i is given recursively as:

$$C_1 = \Gamma' \wedge \neg\Delta'$$
$$C_{i+1} = \bigwedge_j C_i[\alpha_i \backslash t_{i,j}]$$

Theorem 2.25. *Let H be a schematic extended Herbrand sequent. Then the canonical solution for H is indeed a solution.*

Proof. Let σ be the canonical solution for H. Consider the formula $C_i \to \bigwedge_j C_i[\alpha_i \backslash t_{i,j}]$ in $H\sigma$. By the definition of the canonical solution, this formula is equal to $C_i \to C_{i+1}$.

Then $H\sigma$ simplifies to the following sequent:

$$\Gamma', \Gamma' \wedge \neg\Delta' \to C_1, C_1 \to C_2, \dots, C_n \to C_{n+1} \vdash \Delta'$$

By Definition 2.20 of the schematic extended Herbrand sequent, C_{n+1} is a quasi-tautology, and hence $H\sigma$ is one as well. □

2.3 Grammars

For this section we will always consider the following fixed Σ_1 sequent $\Gamma \vdash \Delta$:

$$\forall x_1 \dots \forall x_{k_1} \varphi_1, \dots, \forall x_1 \dots \forall x_{k_l} \varphi_l \vdash \exists x_1 \dots \exists x_{k_{l+1}} \varphi_{l+1}, \dots, \exists x_1 \dots \exists x_{k_p} \varphi_p$$

Herbrand sequents capture the instances of the formulas in the end-sequent that are contained in a proof. In addition of viewing them as a sequent, we can collect them in a term language in an extended signature $\Sigma' \cup \{r_1/k_1, \dots, r_p/k_p\}$ that includes a new k_i-ary function symbol r_i for each formula $\forall \overline{x} \varphi_i$. For example, the Herbrand sequent $\varphi_3[x_1 \backslash c, x_2 \backslash f(d), x_3 \backslash g(f(d))] \vdash$ corresponds to the term language $\{r_3(c, f(d), g(f(d))\}$.

Definition 2.26. Let $\varphi_i[\overline{x} \backslash \overline{t}]$ be an instance of $\forall \overline{x}\, \varphi_i$, then the corresponding term is $r_i(\overline{t})$.

Vice versa, let $r_i(\overline{t})$ be a term, then its corresponding instance is $\varphi_i[\overline{x} \backslash \overline{t}]$.

Definition 2.27. A proof grammar G is a trat grammar $G = (\tau, N, \Sigma \cup \{r_1, \dots, r_p\}, P)$ such that:

- All productions starting with the axiom τ are of the form $\tau \to r_i(\overline{s})$, where $s \in \mathcal{T}(\Sigma)$.
- There are no other occurrences of the symbols r_i.

Under this correspondence, schematic extended Herbrand sequents directly correspond to proof grammars:

Definition 2.28. Let $H = \Gamma', \varphi_1 \to \bigwedge_j \varphi_1[\alpha_1 \backslash t_{1,j}], \ldots, \varphi_n \to \bigwedge_j \varphi_n[\alpha_n \backslash t_{n,j}] \vdash \Delta'$ be a schematic extended Herbrand sequent of $\Gamma \vdash \Delta$, then the proof grammar $G(H) = (\alpha_1, N, \Sigma', P)$ of H is given by:

- $N = \{\alpha_1, \ldots, \alpha_n\}$.
- $P = \{\tau \to r_i(\bar{t})\colon \varphi_i[\overline{x} \backslash \bar{t}] \in \Gamma' \cup \Delta'\} \cup \{\alpha_i \to t_{i,j}\}$.

Definition 2.29. Let $G = (\tau, N, \Sigma', P)$ be a proof grammar. Its schematic extended sequent $H(G)$ is given by:

$$H(G) = \begin{cases} \varphi_i[\overline{x} \backslash \bar{t}], \text{ for every } \tau \to r_i(\bar{t}) \in P \text{ and } i \leq l \\ X_\alpha(\alpha) \to \bigwedge_{\alpha \to t \in P} X_\alpha(t) \text{ for every } \alpha \in N \\ \vdash \\ \varphi_i[\overline{x} \backslash \bar{t}] \text{ for every } \tau \to r_i(\bar{t}) \in P \text{ and } i > l \end{cases}$$

Depending on the grammar G, the schematic extended sequent $H(G)$ may not be a schematic extended *Herbrand* sequent; this is the case if the language generated by the grammar does not correspond to a Herbrand sequent, i.e. if it is not a quasi-tautology.

Lemma 2.30. *Let G be a proof grammar, then $G(H(G)) = G$ (up to a renaming of the non-terminals).*

Let H be a schematic extended Herbrand sequent, then $H(G(H)) = H$ (up to a renaming of the second-order variables X_i, and as sets of formulas, i.e. ignoring duplicate instances).

Definition 2.31. The proof grammar $G(\pi)$ of a proof π is the proof grammar of its extended Herbrand sequent H:

$$G(\pi) := G(H)$$

Theorem 2.32. *Let π be a simple proof of $\Gamma \vdash \Delta$, and $G(\pi)$ its proof grammar.*

Then $\{\varphi_i[\overline{x} \backslash \bar{t}]\colon r_i(\bar{t}) \in \mathcal{L}(G(\pi)), i \leq l\} \vdash \{\varphi_i[\overline{x} \backslash \bar{t}]\colon r_i(\bar{t}) \in \mathcal{L}(G(\pi)), i > l\}$ is an Herbrand sequent for $\Gamma \vdash \Delta$.

Proof. See [Het+14b]. □

3 Inductive proofs

In this section we look at proofs in LK with the following additional rule allowing induction:

$$\frac{\Gamma \vdash \varphi[0], \Delta \qquad \Gamma, \varphi[\nu] \vdash \varphi[s(\nu)], \Delta}{\Gamma \vdash \varphi[t], \Delta}\ \mathrm{Ind}$$

Here, t is an arbitrary term, ν is the eigenvariable of the inference, and must not occur in Γ and Δ.

The constant symbol 0 and the unary function symbol s denote zero and successor, respectively. Numerals are defined by iterated application of the successor, i.e. $k = s(k-1)$ and $0 = 0$ (we identify numbers and numerals for ease of readability).

Definition 3.1. An sip (short for simple induction proof) is an LK-proof with induction π of the following form:

$$\dfrac{\dfrac{\dfrac{\dfrac{[\pi_{\mathrm{base}}]}{\Gamma_0[\alpha,\beta] \vdash F[\alpha,0,\beta]}}{\Gamma[\alpha] \vdash \forall y\, F[\alpha,0,y]} \qquad \dfrac{\dfrac{[\pi_{\mathrm{step}}]}{\Gamma_1[\alpha,\nu,\gamma], \bigwedge_i F[\alpha,\nu,t_i[\alpha,\nu,\gamma]] \vdash F[\alpha,s(\nu),\gamma]}}{\Gamma[\alpha], \forall y\, F[\alpha,\nu,y] \vdash \forall y\, F[\alpha,s(\nu),y]}}{\Gamma[\alpha] \vdash \forall y\, F[\alpha,\alpha,y]} \qquad \dfrac{\dfrac{[\pi_{\mathrm{end}}]}{\Gamma_2[\alpha], \bigwedge_i F[\alpha,\alpha,u_i[\alpha]] \vdash B[\alpha]}}{\Gamma[\alpha], \forall y\, F[\alpha,\alpha,y] \vdash B[\alpha]}}{\Gamma[\alpha] \vdash B[\alpha]}$$

- The background theory $\Gamma[\alpha]$ is a set of Π_1 formulas in prenex form, i.e. each formula is of the form $\forall x_1 \ldots \forall x_{k_i}\, \varphi_i[\alpha, x_1, \ldots, x_{k_i}]$ where only the indicated variables may occur as free variables and i is an index identifying the formula.
- $\Gamma_0[\alpha,\beta]$, $\Gamma_1[\alpha,\nu,\gamma]$ and $\Gamma_2[\alpha]$ are sets of instances of formulas in $\Gamma[\alpha]$ that may only contain the indicated variables as free variables; e.g. an instance in $\Gamma_0[\alpha,\beta]$ is of the form $\varphi_i[\alpha, s_1[\alpha,\beta], s_2[\alpha,\beta], \ldots, s_{k_i}[\alpha,\beta]]$.
- $t_i[\alpha,\nu,\gamma]$ and $u_i[\alpha]$ are terms containing only the indicated free variables.
- The subproofs π_{base}, π_{step}, and π_{end} are cut-free and do not contain induction rules.

Remark 3.2. This definition of simple induction proof is slightly more general than the one presented in [EH15]. There, in the end-sequent α is only allowed to occur in $B[\alpha]$, and not in the background theory Γ. The reason we include it here is because it will simplify the encoding of simple loop problems into sips. This generalization does not incur any significant changes to the theory; the notions of sip grammars, schematic sips and their solutions stay the same.

Substituting a concrete number n for the eigenvariable α in the end-sequent allows us to produce a proof with n cuts instead of the induction rule:

Definition 3.3. The instance proof π_n of an sip π for n is an LK-proof without induction of the following form:

$$\dfrac{\dfrac{\dfrac{[\pi_{\text{base}}]}{\Gamma_0[n,\gamma_0] \vdash F[n,0,\gamma_0]}}{\Gamma[n] \vdash \forall y\, F[n,0,y]} \qquad \dfrac{\dfrac{[\pi_{\text{step}}]}{\Gamma_1[n,0,\gamma_1], \bigwedge_i F[n,0,t_i[n,0,\gamma_1]] \vdash F[n,1,\gamma_1]}}{\Gamma[n], \forall y\, F[n,0,y] \vdash \forall y\, F[n,1,y]}}{\Gamma[n] \vdash \forall y\, F[n,1,y]}$$

$$\ddots$$

$$\dfrac{\Gamma[n] \vdash \forall y\, F[n,n,y] \qquad \dfrac{\dfrac{[\pi_{\text{end}}]}{\Gamma_2[n], \bigwedge_i F[n,n,u_i[n]] \vdash B[n]}}{\Gamma[n], \forall y\, F[n,n,y] \vdash B[n]}}{\Gamma[n] \vdash B[n]}$$

The grammar $G(\pi_n)$ of the instance proof π_n produces a language that corresponds to an Herbrand sequent H of the end-sequent $\Gamma[n] \vdash B[n]$. We encode the instances in H as terms in $\mathcal{L}(G(\pi_n))$ as follows:

- The term $r_i(s_1, \ldots, s_{k_i})$ corresponds to the instance $\varphi_i[\alpha, s_1, \ldots, s_{k_i}]$.
- The constant r_B corresponds to the "instance" of $B[\alpha]$. (r_B is nullary because $B[\alpha]$ has zero quantifiers)

We can now read off the productions of $G(\pi_n)$ from the instance proof π_n:

- $\tau \to r_B$.
- $\tau \to r_j(\overline{s})$ for every instance $\varphi_j[n, \overline{s}] \in \Gamma_0[n, \gamma_0]$.
- $\tau \to r_j(\overline{s})$ for every instance $\varphi_j[n, \overline{s}] \in \Gamma_1[n, i, \gamma_j]$ and every $i < n$.
- $\gamma_i \to t_j[n, i, \gamma_{i+1}]$ for every term $t_j[n, i, \gamma_{i+1}]$ and every $i < n$.
- $\tau \to r_j(\overline{s})$ for every instance $\varphi_j[n, \overline{s}] \in \Gamma_2[n]$.
- $\gamma_n \to u_j[n]$ for every term $u_j[n]$.

For different n, these grammars follow a clear pattern, since the proofs follow a similar pattern as well; a sip grammar captures this pattern—the productions in the grammar of an instance proof will be (suitable) substitution instances of productions of the sip grammar.

Definition 3.4. A sip grammar $G = (\Sigma, P)$ is a tree grammar with non-terminals τ, β, γ, γ_{end}, α, and ν where only the following forms of productions are allowed:

- $\tau \to r_i(\overline{s}[\alpha, \beta])$

- $\tau \to r_i(\overline{s}[\alpha, \nu, \gamma])$
- $\gamma \to s[\alpha, \nu, \gamma]$
- $\gamma_{\text{end}} \to s[\alpha]$

The terms s are arbitrary terms in the language Σ only containing the indicated non-terminals.

Definition 3.5. The sip grammar for the sip π contains the following productions:

- $\tau \to r_B$
- $\tau \to r_j(\overline{s})$ for every instance $\varphi_j[\alpha, \overline{s}] \in \Gamma_0[\alpha, \beta]$.
- $\tau \to r_j(\overline{s})$ for every instance $\varphi_j[\alpha, \overline{s}] \in \Gamma_1[\alpha, \nu, \gamma]$.
- $\gamma \to t_j[\alpha, \nu, \gamma]$ for every term t_j.
- $\tau \to r_j(\overline{s})$ for every instance $\varphi_j[\alpha, \overline{s}] \in \Gamma_2[\alpha]$.
- $\gamma_{\text{end}} \to u_j[\alpha]$ for every term u_j.

Definition 3.6. Let n be a natural number and p a production of a sip grammar. A production p' is an n-instantiation of p, written $p \overset{n}{\rightsquigarrow} p'$, if it is obtained in one of the following ways:

$$
\begin{array}{rcll}
\tau \to r[\alpha, \beta] & \overset{n}{\rightsquigarrow} & \tau \to r[n, \gamma_0] & \\
\tau \to r[\alpha, \nu, \gamma] & \overset{n}{\rightsquigarrow} & \tau \to r[n, i, \gamma_{i+1}] & \text{for any } 0 \leq i < n \\
\gamma \to r[\alpha, \nu, \gamma] & \overset{n}{\rightsquigarrow} & \gamma_i \to r[n, i, \gamma_{i+1}] & \text{for any } 0 \leq i < n \\
\gamma_{\text{end}} \to r[\alpha] & \overset{n}{\rightsquigarrow} & \gamma_n \to r[n] &
\end{array}
$$

Definition 3.7. Let n be a natural number. The n-th instance grammar $G_n = (\tau, N_n, \Sigma, P_n)$ of a sip grammar $G = (\Sigma, P)$ is a trat grammar with non-terminals $N_n = \{\tau\} \cup \{\gamma_i \colon 0 \leq i \leq n\}$, and all n-instantiations of P as productions:

$$P_n = \{p' \colon \exists p \in P \; p \overset{n}{\rightsquigarrow} p'\}$$

Lemma 3.8. *The grammar of an instance proof is the same as the instance grammar of the sip grammar:*

$$G(\pi_n) = G(\pi)_n$$

Proof. Both grammars have the same non-terminals, $\tau, \gamma_0, \gamma_1, \ldots, \gamma_n$—we only have to check that the productions are the same as well.

Each of the productions in $G(\pi_n)$ is a production of $G(\pi)_n$ as well—we just have to go through the list of productions in $G(\pi_n)$ and check that each is an n-instantiation of a production in $G(\pi)$.

The other direction is a bit more complicated; while Definition 3.5 and Definition 3.6 have corresponding cases, a production starting from the non-terminal τ can arise in one case in Definition 3.5, but be n-instantiated in a different case in Definition 3.6—we have to verify all combinations.

For example, let $p = \tau \to r_j(\overline{s})$ be a production in $G(\pi)$ because $\varphi_j[\alpha, \overline{s}]$ is in $\Gamma_0[\alpha, \beta]$. Depending on whether $r_j(\overline{s})$ actually contains the non-terminals α and β or not, an n-instantiation can be obtained in different ways:

- Without any restrictions on the non-terminals in p, we can have $p \overset{n}{\rightsquigarrow} p' = \tau \to r_j(\overline{s}[\alpha \backslash n, \beta \backslash n])$. This is the corresponding case, and we clearly have $p' \in G(\pi_n)$.
- If β does not occur in p, then $p \overset{n}{\rightsquigarrow} p' = \tau \to r_j(\overline{s}[\alpha \backslash n, \nu \backslash i, \gamma \backslash \gamma_{i+1}])$ is possible. But since ν and γ did not occur either, this produces the same production p as above and is again in $G(\pi_n)$. □

3.1 Schematic sips

Definition 3.9. For an end-sequent $\Gamma[\alpha] \vdash B[\alpha]$, sets of instances $\Gamma_0[\alpha, \beta]$, $\Gamma_1[\alpha, \nu, \gamma]$, and $\Gamma_2[\alpha]$ of Γ, and finite sets of terms $t_i[\alpha, \nu, \gamma]$ and $u_i[\alpha]$, the schematic sip S is the triple of the following sequents, where X is a ternary second-order predicate variable:

$$\Gamma_0[\alpha, \beta] \vdash X[\alpha, 0, \beta]$$
$$\Gamma_1[\alpha, \nu, \gamma], \bigwedge_i X[\alpha, \nu, t_i[\alpha, \nu, \gamma]] \vdash X[\alpha, s(\nu), \gamma]$$
$$\Gamma_2[\alpha], \bigwedge_i X[\alpha, \alpha, u_i[\alpha]] \vdash B[\alpha]$$

Definition 3.10. Let G be a sip grammar for the end-sequent $\Gamma[\alpha] \vdash B[\alpha]$ with set of productions P. Its schematic sip is then given as follows:

- $\Gamma_0[\alpha, \beta] = \{\varphi_j[\alpha, \overline{s}[\alpha, \beta]] \colon \tau \to r_j(\overline{s}[\alpha, \beta]) \in P, j \neq B\}$
- $\Gamma_1[\alpha, \nu, \gamma] = \{\varphi_j[\alpha, \overline{s}[\alpha, \nu, \gamma]] \colon \tau \to r_j(\overline{s}[\alpha, \nu, \gamma]) \in P, j \neq B\}$
- $\Gamma_2[\alpha] = \{\varphi_j[\alpha, \overline{s}[\alpha]] \colon \tau \to r_j(\overline{s}[\alpha]) \in P, j \neq B\}$
- The terms $t_i[\alpha, \nu, \gamma]$ are all right sides of productions starting with the non-terminal γ; if there are no such productions, the term $t_1 = 0$ is included instead.
- The terms $u_i[\alpha]$ are all right sides of productions starting with the non-terminal γ_{end}, if there are no such productions, the term $u_1 = 0$ is included instead.

(The sets of instances Γ_0, Γ_1, and Γ_2 are in general not disjoint.)

Definition 3.11. A solution for the schematic sip S is given by a quantifier-free formula $F[\alpha, \nu, \gamma]$ such that each of the sequents in S is a quasi-tautology in $S[X \backslash F]$.

Lemma 3.12. *Let π be a sip and S its schematic sip. The induction formula F from π is then a solution for the schematic sip.*

Proof. Note that the sets of instances Γ_i in S are supersets of the respective sets in π.

The cut-free proofs π_{base}, π_{step}, and π_{end} are proofs of the sequents in $S[X\backslash F]$. Hence $S[X\backslash F]$ is quasi-tautological and F a solution of S. □

Lemma 3.13. *Let S be a schematic sip with solution F, then there is a sip π which has S as schematic sip and F as induction formula.*

Proof. Let π_{base}, π_{step}, and π_{end} be cut-free proofs of the respective sequents in $S[X\backslash F]$. These exist since $S[X\backslash F]$ is a quasi-tautology.

The sets of instances Γ_i and step-terms t_j and u_j together with the proofs π_{base}, π_{step}, and π_{end} then give the desired sip π. □

The schematic sip gives conditions for an induction formula F; in an instance proof this induction formula turns into n cut formulas. The corresponding condition for the cut formulas is the schematic extended Herbrand sequent H; for an instance proof π_n this is given as follows:

$$\begin{aligned}\Gamma_0[n,\gamma_0], \Gamma_1[n,0,\gamma_1], \ldots, \Gamma_1[n,n-1,\gamma_n], \Gamma_2[n]& \\ X_0(\gamma_0) \to \bigwedge_i X_0(t_i[n,0,\gamma_1])&, \\ \vdots& \\ X_{n-1}(\gamma_{n_1}) \to \bigwedge_i X_{n-1}(t_i[n,0,\gamma_n])&, \\ X_n(\gamma_n) \to \bigwedge_i X_n(u_i[n]) &\vdash B[n]\end{aligned} \tag{3.1}$$

If we have a sip and its induction formula F, then the (instantiated) cut formulas $F_i := F[n,i,\gamma_i]$ of the instance proof π_n give a solution of the schematic extended Herbrand sequent. However, for fixed n, we can explicitly give a solution without a sip, the canonical substitution of its schematic extended Herbrand sequent as in Definition 2.24:

$$\begin{aligned}C_0 &= \bigwedge \Gamma_0[n,\gamma_0] \wedge \bigwedge \Gamma_1[n,0,\gamma_1] \wedge \cdots \wedge \bigwedge \Gamma_1[n,n-1,\gamma_n] \wedge \bigwedge \Gamma_2[n] \wedge \neg B[n] \\ C_1 &= \bigwedge_i C_0[\gamma_0 \backslash t_i[n,0,\gamma_1]] \\ &\vdots \\ C_n &= \bigwedge_i C_{n-1}[\gamma_{n-1} \backslash t_i[n,n-1,\gamma_n]]\end{aligned}$$

This canonical substitution σ is different for each n, not only because the numeral n can occur in C_0 (this is something we cannot avoid), but also because even the propositional

structure of C_0 depends on the value of n: If $\varphi_j[\alpha, \overline{s}] \in \Gamma_1$ contains ν, then C_0 will contain n different instances of φ_j.

But for sips, we can give a more regular solution:

Definition 3.14. The canonical solution of a schematic sip S is given by the collection of formulas $C'_i[\alpha, \gamma]$ for $i \geq 0$:

$$C'_0[\alpha, \gamma] = \bigwedge \Gamma_0[\alpha, \gamma]$$
$$C'_{i+1}[\alpha, \gamma] = \bigwedge \Gamma_1[\alpha, i, \gamma] \wedge \bigwedge_j C'_i[\alpha, t_j[\alpha, i, \gamma]]$$

The canonical solution C'_i of a sip will induce a solution of the schematic extended Herbrand sequent by substituting $X_i(\gamma_i) \mapsto C'_i[n, \gamma_i]$. This solution differs from the generic canonical solution only by the index at which the formulas first appear.

Lemma 3.15. *Let $H = \Gamma', X_1(\alpha_1) \to \bigwedge_j X_1(t_{1,j}), \ldots X_n(\alpha_n) \to \bigwedge_j X_n(t_{n,j}) \vdash \Delta'$ be a schematic extended Herbrand sequent for $\Gamma \vdash \Delta$, and let $f \colon \Gamma' \cup \Delta' \to \{1, \ldots, n+1\}$ be a function such that $f(\varphi_j[\overline{x} \backslash \overline{t}]) \leq i$ if α_i occurs in $\overline{t}$ for any i.*

The f-modified canonical solution for H given by $X_i(\gamma_i) \mapsto D_i$ then solves H, where D_i is given recursively as:

$$D_1 = \bigwedge f^{-1}(1)$$
$$D_{i+1} = \bigwedge f^{-1}(i+1) \wedge \bigwedge_j C_i[\alpha_i \backslash t_{i,j}]$$

Proof. Let us first show by induction on i that $\vdash D_{i+1} \leftrightarrow \bigwedge_{j_1,\ldots,j_i} f^{-1}(\{1, \ldots, i + 1\})[\alpha_1 \backslash t_{1,j_1}] \cdots [\alpha_i \backslash t_{i,j_i}]$. For D_1 this is immediate; for D_{i+1} we use the fact that $f^{-1}(i+1) = f^{-1}[\alpha_1 \backslash t_{1,j_1}] \cdots [\alpha_i \backslash t_{i,j_i}]$ since $f^{-1}(i+1)$ does not contain any of the non-terminals $\alpha_1, \ldots, \alpha_i$ by assumption.

We can now proceed in a similar manner as in Theorem 2.25, and prove that the following sequent is quasi-tautological:

$$\Gamma', D_1 \to \bigwedge_j D_1[\alpha_1 \backslash t_{1,j}], \ldots, D_n \to \bigwedge_j D_n[\alpha_n \backslash t_{n,j}] \to \Delta'$$

But $\Gamma', \neg\Delta' \vdash \left(\bigwedge_j D_i[\alpha_1 \backslash t_{1,j}]\right) \to D_{i+1}$ for all i, and D_{n+1} is again a quasi-tautology. □

Corollary 3.16. *For a schematic sip S and schematic extended Herbrand sequent H for n, the assignment $\sigma' \colon X_i(\gamma_i) \mapsto C'_i[n, \gamma_i]$ is a solution of the schematic extended Herbrand sequent.*

Proof. The sequence $C'_i[n, \gamma_i]$ is an f-modified canonical solution, given by the following f:

$$\begin{aligned} f(\varphi) &= 0 && \text{for } \varphi \in \Gamma_0[n, \gamma_0] \\ f(\varphi) &= i + 1 && \text{for } \varphi \in \Gamma_1[n, i, \gamma_{i+1}] \text{ and } 0 \leq i < n \\ f(\varphi) &= n + 1 && \text{for } \varphi \in \Gamma_2[n] \\ f(B[n]) &= n + 1 \end{aligned}$$

□

3.2 Finding minimal sip grammars

First of all, we need to adapt some definitions from trat grammars for sip grammars:

Definition 3.17. Let $G = (\Sigma, P)$ be a sip grammar. Its size $|G|$ is the number $|P|$ of its productions.

Definition 3.18. Let $G' = (\Sigma, P')$ and $G = (\Sigma, P)$ be sip grammars in the same signature. G' is a sub-grammar of G if $P' \subseteq P$.

Definition 3.19. Let G be a sip grammar, and (L_i) a family of languages. The sip grammar G covers the languages (L_i) if $\mathcal{L}(G_i) \supseteq L_i$ for all i.

Given a finite family of languages $(L_i)_{i \in I}$, we want to find a sip grammar G that covers them. This task poses two new problems compared to the case for trat grammars:

- There are several different languages, not just a single one.
- The productions in the generated grammar need to satisfy certain restrictions, in particular the right-hand sides may only contain some non-terminals.

But as with trat grammars, this task can again be divided into two problems:

1. Find a polynomially-sized sip grammar H containing a sip grammar of minimal size covering (L_i).
2. Find a sip grammar $G \subseteq H$ covering (L_i) of minimal size.

The following definitions and theorems closely follow the approach taken for the case of trat grammars; they are natural generalizations of Definition 1.35, Lemmas 1.34 and 1.54, Theorem 1.36, and Corollary 1.53 for sip grammars.

In order to construct the grammar H, we first have to define normal forms for sip grammars.

Definition 3.20. A production $\delta \to k$ of is in sip-normal form relative to the family of languages (L_i) if it is of one of the forms in Definition 3.4 and $\delta \to k$ is in trat-normal form relative to the language $\bigcup_i L_i$.

Remark 3.21. The normal forms here are much less restrictive than those in [EH15]; however, this does not change the main result of this section: that finding a sip grammar covering (L_i) of minimal size is polynomial-time reducible to Max-SAT.

Lemma 3.22. *Let G be a sip grammar, $\delta \to k$ one of its productions, L_i an instance language for the number i, and k' a term such that $k \in \mathrm{st}(L_i) \models k = k'$.*

Then $\mathcal{L}(G_i) \cap L_i \subseteq \mathcal{L}(G'_i) \cap L_i$ for the sip grammar G' which is obtained from G by replacing the production $\delta \to k$ by $\delta \to k'$.

Note that k' has at most the non-terminals occurring in k, hence G' is indeed a sip grammar.

Proof. Consider the changes between G_i and G'_i; these are replacements of n-instantiations of a production $\delta \to k$ by n-instantiations of $\delta \to k'$. We will handle replacing each of these n-instantiations one by one using Lemma 1.34.

The crucial property of these n-instantiations is that they are substitution instances of $\delta \to k$, i.e. there is a substitution σ that does not depend on k such that $\epsilon \to k\sigma \in P_i$ is the n-instantiation (for some non-terminal $\epsilon \in N_i$ which does not interest us as we do not change it).

For example consider the n-instantiations of $\gamma \to k$: these are precisely $\gamma_j \to k[\alpha\backslash i, \nu\backslash j, \gamma\backslash\gamma_{j+1}]$ for $j < i$—the substitution σ is given by $\sigma = [\alpha\backslash i, \nu\backslash j, \gamma\backslash\gamma_{j+1}]$, and $\epsilon = \gamma_j$.

Let us now prove that we can replace $\epsilon \to k\sigma$ with $\epsilon \to k'\sigma$ in G_i. We only need to verify that $k\sigma \in L_i \models k\sigma = k'\sigma$—but this is true since $k \in L_i \models k = k'$ by assumption. □

Theorem 3.23. *Let (L_i) be a family of languages, and $G = (\Sigma, P)$ a sip grammar covering (L_i). Then there is a sip grammar $G' = (\Sigma, P')$ with $|P'| \leq |P|$ where all productions are in sip-normal form relative to (L_i) that still covers (L_i).*

Proof. For each production $p = \delta \to k \in P$ take a k' in normal form relative to $\mathrm{st}(\bigcup L_i)$ such that $k \in \mathrm{st}(\bigcup L_i) \models k = k'$. The production $p' = \delta \to k'$ is then in sip-normal form, let P' be the set of such productions $\delta \to k'$.

Successively applying Lemma 3.22 shows that $\mathcal{L}(G_i) \cap L_i \subseteq \mathcal{L}(G'_i) \cap L_i$ and hence $L_i \subseteq \mathcal{L}(G'_i)$. □

Lemma 3.24. *Every finite family of languages (L_i) can be covered by a sip grammar.*

Proof. Take the sip grammar with the productions $P = \{\tau \to t \colon t \in \bigcup_i L_i\}$. □

Lemma 3.25. *There exists an infinite family of languages (L_i) such that there is no sip grammar covering it.*

Proof by asymptotics. By Lemma 1.12, $|\mathcal{L}(G_n)| \leq |P_n|^{|N_n|+1} \leq (n|P|)^{n+3}$, and any (L_i) whose size grows sufficiently fast is a counterexample. □

Proof by concrete counterexample. Take the family of languages $L_n = \{r(i,j,k)\colon i,j,k \leq n\}$ and assume there is a sip grammar G covering (L_n).

Let us first show that there is a C_G that only depends on the sip grammar G such that for all numbers i,j,k, if $r(i,j,k) \in \mathcal{L}(G_n)$, then $\min\{|a-b|\colon a,b \in \{0,i,j,k,n\} \wedge a \neq b\} \leq C_G$.

For each production $p = \delta \to r(p_1,p_2,p_3)$ in G, we define a number C_p such that $\min\{|a-b|\colon a,b \in \{0,p_1\sigma,p_2\sigma,p_3\sigma,n\} \wedge a \neq b\} \leq C_p$ if $\sigma \models \varphi_{G_n} \wedge \delta = r(p_1,p_2,p_3)$.

- If any p_i contains a function symbol other than s or 0, then set $C_p = 0$.
- If a p_i is ground (i.e. a numeral), then set $C_p = p_i$.
- If a p_i contains the non-terminal α, i.e. $p_i = s^k(\alpha)$, then set $C_p = k$.
- If at least two arguments contain the non-terminal ν, i.e. $p_i = s^k(\nu)$ and $p_j = s^l(\nu)$ with $i \neq j$, then set $C_p = |k-l|$.
- If none of the above applies, there are at least two arguments containing γ, i.e. $p_i = s^k(\gamma)$ and $p_j = s^l(\gamma)$ with $i \neq j$, and we can set $C_p = |k-l|$ again.

Now we can define $C_G = \max_p C_p$; using C_G we can give a concrete term that is not generated by G_n: namely $r(n - C_G - 1, n - 2C_G - 2, n - 3C_G - 3) \notin \mathcal{L}(G_n)$ if $n \geq 4C_G + 4$. $\square$

Lemma 3.26. *Let (L_i) be a family of languages. There is a sip grammar $H = (\Sigma, Q)$ that contains a grammar covering (L_i) of minimal size as a sub-grammar.*

This grammar can be computed by an algorithm whose runtime is polynomially bounded in $|\mathrm{st}(\bigcup L_i)|$. The number of productions in H and the number of subterms of right-hand sides of productions in H is then also polynomially bounded.

Proof. First compute the set K of normal forms with non-terminals α,ν,γ relative to the language $\mathrm{st}(\bigcup L_i)$ using Theorem 1.48. This is polynomial-time computable as in Lemma 1.54.

We need to handle productions of different forms separately:

- To get productions of the form $\tau \to s[\alpha,\beta]$, add for each $k \in K$ with $\mathrm{Var}(k) \subseteq \{\alpha,\gamma\}$ the production $\tau \to k[\gamma\backslash\beta]$ to P.
- To get productions of the form $\tau \to s[\alpha,\nu,\gamma]$, add for each $k \in K$ with $\mathrm{Var}(k) \subseteq \{\alpha,\nu,\gamma\}$ the production $\tau \to k$ to P.
- Similarly for productions of the form $\gamma \to s[\alpha,\nu,\gamma]$, and $\gamma_{\mathrm{end}} \to s[\alpha]$.

The resulting grammar H consists of all sip-normal forms and contains a grammar covering (L_i) of minimal size by Theorem 3.23. $\square$

Lemma 3.27. *Let G be a sip grammar, and (L_i) a finite family of languages. There exists a propositional formula $\psi_{G,(L_i)}$ such that the following are equivalent for a sub-grammar $G' \subseteq G$:*

- *There is an assignment I satisfying $\psi_{G,(L_i)}$ such that $P' = \{p \in P \colon I \models \delta \to k \in P'\}$.*
- *G' covers (L_i).*

Proof.

$$\psi_{G,(L_i)} \equiv \bigwedge_i \psi_{G_i,L_i} \wedge \bigwedge_{p \overset{i}{\leadsto} p_i} p_i \in P'_i \to p \in P'$$

□

Corollary 3.28. *Let G be a sip grammar, and (L_i) a family of languages. There exists a CNF formula $\tilde{\psi}_{G,(L_i)}$ with hard and soft clauses such that the following are equivalent for a sub-grammar $G' \subseteq G$:*

- *There is a solution I to the Max-SAT problem given by $\tilde{\psi}_{G,(L_i)}$ such that $P' = \{p \in P \colon I \models \text{“}\delta \to k \in P'\text{”}\}$.*
- *G' is a sub-grammar of G covering (L_i) of minimal size.*

Proof. Convert the formula $\psi_{G,(L_i)}$ into CNF. The formula $\tilde{\psi}_{G,(L_i)}$ consists of the clauses of $\psi_{G,(L_i)}$ (which are hard), and the soft clauses $\neg\delta \to k \in P'$ for each $\delta \to k \in P$. □

Theorem 3.29. *Let (L_i) be a finite family of languages. There is a polynomial-time computable Max-SAT problem such that any solution I gives a sip grammar $G = (\Sigma, P)$ of minimal size covering (L_i) by setting $P := \{p \in Q \colon I \models \text{“}p \in P\text{”}\}$ where “$p \in P$” is a different propositional variable for each $p \in Q$.*

The runtime is polynomially bounded in $|\mathrm{st}(L)|$ for fixed n.

Proof. Compute the grammar H as in Lemma 3.26 and then use Corollary 3.28 to produce the Max-SAT problem. □

3.3 Finding induction formulas

Having found a sip grammar, we have almost all data needed for a sip. What is still missing, however, is an induction formula, and the sub-proofs ensuring that it is indeed an induction formula.

By Lemmas 3.12 and 3.13, this amounts to finding a solution of the schematic sip. We will now give a further condition for such a solution that will constrain the search.

Intuitively, the canonical solution $C'_i[n, \gamma_i]$ of a schematic sip captures the information available in the instance proof of a sip at step i because it includes all instances of the background theory used up to that point. A possible induction formula for this sip needs to be provable from these instances as well, so we can conjecture that it is a consequence of this canonical solution, which indeed it is:

Lemma 3.30. *Let S be a schematic sip and $F[\alpha, \nu, \gamma]$ a solution, then $C'_i[\alpha, \gamma] \vdash F[\alpha, i, \gamma]$ for all i.*

Proof. If F is a solution, then by Definition 3.11 the following sequents are quasi-tautologies:

$$\Gamma_0[\alpha,\gamma] \vdash F[\alpha,0,\gamma]$$
$$\Gamma_1[\alpha,i,\gamma], \bigwedge_j F[\alpha,i,t_j[\alpha,i,\gamma]] \vdash F[\alpha,i+1,\gamma]$$

Proceed by induction on i:

The sequent $\Gamma_0[\alpha,\gamma] \vdash F[\alpha,0,\gamma]$ just states that $C_0'[\alpha,\gamma] \vdash F[\alpha,0,\gamma]$.

Consider now $C_{i+1}'[\alpha,\gamma] = \bigwedge \Gamma_1[\alpha,i] \wedge \bigwedge_j C_i'[\alpha,t_j[\alpha,i,\gamma]]$; by applying substitution instances of the induction hypothesis, we get $C_{i+1}'[\alpha,\gamma] \vdash \bigwedge \Gamma_1[\alpha,i] \wedge \bigwedge_j F[\alpha,i,t_j[\alpha,i,\gamma]]$. Using the solution definition again, this yields $C_{i+1}'[\alpha,\gamma] \vdash F[\alpha,i+1,\gamma]$. □

Therefore the task of finding an induction formula amounts to finding a particular consequence D of $C_i'[\alpha,\gamma]$, and then viewing it is a substitution instance $D = F[\alpha,i,\gamma]$, i.e. replacing some occurrences of i by ν.

Definition 3.31. Propositional resolution is the following inference rule on clauses:

$$\frac{\Gamma,\varphi \qquad \Delta,\neg\varphi}{\Gamma,\Delta}$$

Propositional paramodulation is the following inference rule on clauses:

$$\frac{\Gamma, t_1 = t_2 \qquad \Delta[t_i]}{\Gamma, \Delta[t_j]}$$

Forgetful propositional resolution is an operation on clause sets and replaces two clauses Γ,φ and $\Gamma,\neg\varphi$ in a clause set by the clause Γ. Forgetful paramodulation similarly replaces two clauses by their paramodulant.

Eberhard and Hetzl give two algorithms for the problem of finding an induction formula in [EH15]:

Complete algorithm: Enumerate all logical consequences of $C_0'[\alpha.\gamma]$. For each consequence D, enumerate all possible F that yields D as a substitution instance—these only depend on which occurrences of 0 are replaced with ν or remain a numeral. Since all formulas are quantifier-free, we can efficiently check that an F is indeed a solution for the schematic form, and if so, return it as a solution.

Heuristic algorithm: Given a number n, convert the formula $C_n'[\alpha,\gamma]$ into its CNF $A[\alpha,\gamma]$. Generate consequences $D[\alpha,\gamma]$ of this CNF recursively using the following two operations:

1. Forgetful resolution.
2. Forgetful propositional paramodulation.

Note that each consequence $D[\alpha, \gamma]$ needs to satisfy $\Gamma_2[n], \bigwedge D[n, u_i[n]] \vdash B[n]$ in order to be the substitution instance of a solution; we prune all other $D[\alpha, \gamma]$ from the search tree.

For each generated consequence $D[\alpha, \gamma]$, again check for each possible F whether it is a solution, and if so return it.

4 While programs

We will consider a simple programming language containing loops, conditionals and assignments, with standard operational semantics and Hoare calculus, as presented in [ABO10]. The programs are generated by the following grammar:

$$p ::= \textbf{skip} \mid \mathtt{x} := t \mid p_1;\ p_2 \mid \textbf{if } \varphi \textbf{ then } p_1 \textbf{ else } p_2 \textbf{ fi} \mid \textbf{while } \varphi \textbf{ do } p \textbf{ od}$$

Here, t is a term and φ a quantifier-free formula. These may contain function and relation symbols from a signature Σ.

4.1 Operational semantics

The language is untyped; semantically we will consider all values as natural numbers.

Definition 4.1. A state σ is a map from program variables to values.

$\sigma \models \varphi$ means that the state σ satisfies a formula φ, $\sigma(t)$ is the value of the term t in the state σ. The state $\sigma[\mathtt{y} \mapsto a]$ is the state σ where the variable $\mathtt{y}$ is updated by the value a.

The small-step semantics are given by the following transition rules; pairs $\langle p, \sigma\rangle$ of a program p or the empty program e and a state σ are called configurations; the transition $\langle p, \sigma\rangle \to \langle q, \tau\rangle$ means that running the program p in state σ yields the state τ after a single step with the remaining program q.

1. $\langle \textbf{skip}, \sigma\rangle \to \langle e, \sigma\rangle$
2. $\langle \mathtt{x} := t, \sigma\rangle \to \langle e, \sigma[\mathtt{x} \mapsto \sigma(t)]\rangle$
3. $\dfrac{\langle p_1, \sigma\rangle \to \langle p_2, \tau\rangle}{\langle p_1;\ q, \sigma\rangle \to \langle p_2;\ q, \tau\rangle}$
   (We need to be careful in the case that $p_2 = e$: we identify $e;\ q$ with q, so that $\langle p_2;\ q, \tau\rangle$ is a valid state.)
4. $\langle \textbf{if } \varphi \textbf{ then } p \textbf{ else } q \textbf{ fi}, \sigma\rangle \to \langle p, \sigma\rangle$, where $\sigma \models \varphi$
5. $\langle \textbf{if } \varphi \textbf{ then } p \textbf{ else } q \textbf{ fi}, \sigma\rangle \to \langle q, \sigma\rangle$, where $\sigma \models \neg\varphi$
6. $\langle \textbf{while } \varphi \textbf{ do } p \textbf{ od}, \sigma\rangle \to \langle p;\ \textbf{while } \varphi \textbf{ do } p \textbf{ od}, \sigma\rangle$, where $\sigma \models \varphi$
7. $\langle \textbf{while } \varphi \textbf{ do } p \textbf{ od}, \sigma\rangle \to \langle e, \sigma\rangle$, where $\sigma \models \neg\varphi$

Definition 4.2. Let p be a program and σ a state.

1. A transition sequence is a (finite or infinite) sequence of configurations $\langle p_i, \sigma_i \rangle$ such that:
$$\langle p, \sigma \rangle = \langle p_0, \sigma_0 \rangle \to \langle p_1, \sigma_1 \rangle \to \dots$$
2. A computation is a maximal transition sequence.

Lemma 4.3 (Determinism)**.** *For any program p and state σ, there is exactly one computation starting with the configuration $\langle p, \sigma \rangle$.*

Lemma 4.4 (Progress)**.** *For a program $p \neq e$ and state σ, there is always a program p' and state σ' such that*
$$\langle p, \sigma \rangle \to \langle p', \sigma' \rangle$$

The (partial correctness) semantics $\mathcal{M}[\![p]\!]$ of a program p is now given by the map assigning to each program the set of its final states:

Definition 4.5. $\mathcal{M}[\![p]\!](\sigma) = \{\tau \colon \langle p, \sigma \rangle \to^* \langle e, \tau \rangle\}$

For any p and σ, the semantics $\mathcal{M}[\![p]\!](\sigma)$ is either the empty set, if p does not halt on σ; or the singleton set containing the unique output state otherwise.

Lemma 4.6 (Input/Output)**.** *The single step reductions extend to $\mathcal{M}$:*

1. $\mathcal{M}[\![\mathbf{skip}]\!](X) = X$
2. $\mathcal{M}[\![p;\ q]\!](X) = \mathcal{M}[\![q]\!](\mathcal{M}[\![p]\!](X))$
3. $\mathcal{M}[\![\mathbf{if}\ \varphi\ \mathbf{then}\ p\ \mathbf{else}\ q\ \mathbf{fi}]\!](X) =$
$\mathcal{M}[\![p]\!](\{\sigma \in X \colon \sigma \models \varphi\}) \cup \mathcal{M}[\![q]\!](\{\sigma \in X \colon \sigma \models \neg\varphi\})$
4. $\mathcal{M}[\![\mathbf{while}\ \varphi\ \mathbf{do}\ p\ \mathbf{od}]\!](X) =$
$\mathcal{M}[\![\mathbf{while}\ \varphi\ \mathbf{do}\ p\ \mathbf{od}]\!](\mathcal{M}[\![p]\!](\{\sigma \in X \colon \sigma \models \varphi\})) \cup \{\sigma \in X \colon \sigma \models \neg\varphi\}$

Definition 4.7. The weakest liberal precondition $\mathrm{wlp}(p, \varphi)$ of a formula φ (maybe containing program variables) under a program p is a formula such that for any state σ:
$$\sigma \models \mathrm{wlp}(p, \varphi) \ \leftrightarrow\ \forall \tau \in \mathcal{M}[\![p]\!](\sigma) \colon \tau \models \varphi$$

Lemma 4.8. *The weakest liberal precondition exists.*

Proof. For loop-free programs, we can easily give a canonical weakest liberal precondition:
$$\begin{aligned}
\mathrm{wlp}(\mathbf{skip}, \varphi) &\equiv \varphi \\
\mathrm{wlp}(p;\ q, \varphi) &\equiv \mathrm{wlp}(p, \mathrm{wlp}(q, \varphi)) \\
\mathrm{wlp}(\mathtt{x} := t, \varphi) &\equiv \varphi[\mathtt{x} \backslash t] \\
\mathrm{wlp}(\mathbf{if}\ \psi\ \mathbf{then}\ p\ \mathbf{else}\ q\ \mathbf{fi}, \varphi) &\equiv (\psi \to \mathrm{wlp}(p, \varphi)) \wedge (\neg\psi \to \mathrm{wlp}(q, \varphi))
\end{aligned}$$

That this is indeed a weakest liberal precondition is seen by induction on the program and applying Lemma 4.6. The general case for programs with loops requires encoding the finite sequences of program states at each iteration of the loop into a number, for details see [ABO10]. □

Note that this translation does not introduce quantifiers; this allows us to write the program semantics (for loop-free programs) as a quantifier-free formula:

$$\overline{\tau} \in \mathcal{M}[\![p]\!](\overline{\sigma}) \quad \equiv \quad \mathrm{wlp}(p, \overline{\mathtt{x}} = \overline{\tau})[\overline{\mathtt{x}} \backslash \overline{\sigma}]$$

Let $\mathtt{x}_1, \ldots, \mathtt{x}_n$ be the program variables, then $\sigma_1, \ldots, \sigma_n$ and $\tau_1, \ldots, \tau_n$ are constant symbols and the formula above is short for:

$$\mathrm{wlp}(p, \mathtt{x}_1 = \tau_1 \wedge \cdots \wedge \mathtt{x}_n = \tau_n)[\mathtt{x}_1 \backslash \sigma_1, \ldots, \mathtt{x}_n \backslash \sigma_n]$$

Example 4.9. $\tau \in \mathcal{M}[\![\mathtt{x} := f(\mathtt{x})]\!](\sigma) \quad \equiv \quad \tau = f(\sigma)$

4.2 Hoare logic

The expression $\{\varphi\}\ p\ \{\psi\}$ is called a triple; intuitively this expresses the notion that if φ is true before running the program p, then ψ will be true after the program p has finished.

We write $\vdash \{\varphi\}\ p\ \{\psi\}$ if this triple is derivable using the rules in Section 4.2 and $\models \{\varphi\}\ p\ \{\psi\}$ if it is true in the sense of Definition 4.11.

$$\{\varphi\}\ \textbf{skip}\ \{\varphi\} \quad \text{skip}$$

$$\{\varphi[\mathtt{x}\backslash t]\}\ \mathtt{x} := t\ \{\varphi\} \quad \text{assignment}$$

$$\frac{\{\varphi\}\ p\ \{\psi\} \qquad \{\psi\}\ q\ \{\theta\}}{\{\varphi\}\ p;\ q\ \{\theta\}}\ \text{composition}$$

$$\frac{\{\varphi \wedge \theta\}\ p\ \{\psi\} \qquad \{\varphi \wedge \neg\theta\}\ q\ \{\psi\}}{\{\varphi\}\ \textbf{if}\ \theta\ \textbf{then}\ p\ \textbf{else}\ q\ \textbf{fi}\ \{\psi\}}\ \text{conditional}$$

$$\frac{\{\varphi \wedge \psi\}\ p\ \{\varphi\}}{\{\varphi\}\ \textbf{while}\ \psi\ \textbf{do}\ p\ \textbf{od}\ \{\varphi \wedge \neg\psi\}}\ \text{loop}$$

$$\frac{\models \varphi \to \varphi' \qquad \{\varphi'\}\ p\ \{\psi'\} \qquad \models \psi' \to \psi}{\{\varphi\}\ p\ \{\psi\}}\ \text{consequence}$$

Figure 4.1: Hoare logic for while programs

In Section 4.2 there is one rule for each program construct, and a general consequence rule. The proofs composed of these rules are almost completely determined by the program structure—given a triple $\{\varphi\}\ p\ \{\psi\}$, there are only two rules that can be applied: either the rule for the outermost program construct in p, or the consequence rule.

If we rewrite a proof using Figure 4.2, then we can even get more regular proofs. For example, if p is a conditional or sequence, then the proof will never apply the consequence rule, just the conditional or sequence rule, respectively.

Lemma 4.10. *The rewrite system in Figure 4.2 is terminating, and preserves validity of proofs.*

Definition 4.11. A triple is true, written $\models \{\varphi\}\ p\ \{\psi\}$, if $\sigma \models \varphi$ and $\tau \in \mathcal{M}[\![p]\!](\sigma)$ implies $\tau \models \psi$ for any σ.

Theorem 4.12 (Soundness). $\vdash \{\varphi\}\ p\ \{\psi\}$ *implies* $\models \{\varphi\}\ p\ \{\psi\}$.

Proof. By induction on the proof of $\vdash \{\varphi\}\ p\ \{\psi\}$:
Skip: If $\sigma \models \varphi$ and $\tau \in \mathcal{M}[\![\mathbf{skip}]\!](\sigma) = \{\sigma\}$, then clearly $\tau \models \varphi$ as well.
Assignment: $\mathcal{M}[\![\mathtt{x} := t]\!](\sigma) = \{\sigma[\mathtt{x} \mapsto \sigma(t)]\}$, and $\sigma[\mathtt{x} \mapsto \sigma(t)] \models \varphi$ if $\sigma \models \varphi[x \backslash t]$.
And similarly for the other rules. ☐

Theorem 4.13 (Loop-free completeness). *Let p be a program not containing the while statement, then* $\models \{\varphi\}\ p\ \{\psi\}$ *implies* $\vdash \{\varphi\}\ p\ \{\psi\}$.

Definition 4.14. For a loop $p = \mathbf{while}\ \varphi\ \mathbf{do}\ p_1\ \mathbf{od}$, consider the extended loop $p_x = \mathtt{x} := 0;\ \mathbf{while}\ \varphi\ \mathbf{do}\ p_1;\ \mathtt{x} := \mathtt{x} + 1\ \mathbf{od}$ that counts the number of iterations (where $\mathtt{x}$ is a variable not occurring in p).

If σ is a state such that p_x (or equivalently, p) terminates, i.e. $\mathcal{M}[\![p_x]\!](\sigma) = \{\tau\}$, then we define $\mathrm{iter}(p, \sigma) = \tau(\mathtt{x})$ to be the number of its iterations starting from state σ.

Definition 4.15. A language Σ is called expressive if for every loop p there is a term t such that $\mathrm{iter}(p, \sigma) = \sigma(t)$ whenever defined.

Theorem 4.16 (Completeness). *If Σ is expressive, then* $\models \{\varphi\}\ p\ \{\psi\}$ *implies* $\vdash \{\varphi\}\ p\ \{\psi\}$ *for all programs p.*

Proof. See [ABO10]. ☐

$$\frac{\models \varphi \to \varphi' \qquad \dfrac{\models \varphi' \to \varphi'' \qquad \{\varphi''\}\ p\ \{\psi''\} \qquad \models \psi'' \to \psi'}{\{\varphi'\}\ p\ \{\psi'\}} \qquad \models \psi' \to \psi}{\{\varphi\}\ p\ \{\psi\}} \quad \rightsquigarrow \quad \frac{\models \varphi \to \varphi'' \qquad \{\varphi''\}\ p\ \{\psi''\} \qquad \models \psi'' \to \psi}{\{\varphi\}\ p\ \{\psi\}}$$

$$\frac{\models \varphi \to \varphi' \qquad \dfrac{\{\varphi' \wedge \theta\}\ p\ \{\psi'\} \qquad \{\varphi' \wedge \neg\theta\}\ q\ \{\psi'\}}{\{\varphi'\}\ \textbf{if}\ \theta\ \textbf{then}\ p\ \textbf{else}\ q\ \textbf{fi}\ \{\psi'\}} \qquad \models \psi' \to \psi}{\{\varphi\}\ \textbf{if}\ \theta\ \textbf{then}\ p\ \textbf{else}\ q\ \textbf{fi}\ \{\psi\}} \quad \rightsquigarrow \quad \frac{\dfrac{\models \varphi \wedge \theta \to \varphi' \wedge \theta \qquad \{\varphi' \wedge \theta\}\ p\ \{\psi'\} \qquad \models \psi' \to \psi}{\{\varphi \wedge \theta\}\ p\ \{\psi\}} \qquad \dfrac{\dots}{\{\varphi \wedge \neg\theta\}\ q\ \{\psi\}}}{\{\varphi\}\ \textbf{if}\ \theta\ \textbf{then}\ p\ \textbf{else}\ q\ \textbf{fi}\ \{\psi\}}$$

$$\frac{\models \varphi \to \varphi' \qquad \dfrac{\{\varphi'\}\ p\ \{\psi\} \qquad \{\psi\}\ q\ \{\theta'\}}{\{\varphi\}\ p;\ q\ \{\theta\}} \qquad \models \theta' \to \theta}{\{\varphi\}\ p;\ q\ \{\theta\}} \quad \rightsquigarrow \quad \frac{\dfrac{\models \varphi \to \varphi' \qquad \{\varphi'\}\ p\ \{\psi\}}{\{\varphi\}\ p\ \{\psi\}} \qquad \dfrac{\{\psi\}\ q\ \{\theta'\} \qquad \models \theta' \to \theta}{\{\psi\}\ q\ \{\theta\}}}{\{\varphi\}\ p;\ q\ \{\theta\}}$$

$$\frac{\{\varphi[\mathtt{x}\backslash t]\}\ \mathtt{x} := t\ \{\varphi\} \qquad \models \varphi \to \psi}{\{\varphi[\mathtt{x}\backslash t]\}\ \mathtt{x} := t\ \{\psi\}} \quad \rightsquigarrow \quad \frac{\models \varphi[\mathtt{x}\backslash t] \to \psi[\mathtt{x}\backslash t] \qquad \{\psi[\mathtt{x}\backslash t]\}\ \mathtt{x} := t\ \{\psi\}}{\{\varphi[\mathtt{x}\backslash t]\}\ \mathtt{x} := t\ \{\psi\}}$$

$$\frac{\{\varphi\}\ p\ \{\psi\} \qquad \dfrac{\models \psi \to \psi' \qquad \{\psi'\}\ q\ \{\theta\}}{\{\psi\}\ q\ \{\theta\}}}{\{\varphi\}\ p;\ q\ \{\theta\}} \quad \rightsquigarrow \quad \frac{\dfrac{\{\varphi\}\ p\ \{\psi\} \qquad \models \psi \to \psi'}{\{\varphi\}\ p\ \{\psi'\}} \qquad \{\psi'\}\ q\ \{\theta\}}{\{\varphi\}\ p;\ q\ \{\theta\}}$$

Figure 4.2: Hoare proof normalization

5 Loop verification

From now on, we will always consider a fixed simple program with l variables $\mathtt{x}_1, \ldots, \mathtt{x}_l$ (in addition to the variables $\mathtt{n}$ and $\mathtt{i}$ for the for-loop) containing a single loop with a loop-free body S that does not contain assignments to $\mathtt{i}$ or $\mathtt{n}$, but only to $\overline{\mathtt{x}}$.

$$\mathbf{for}\ \mathtt{i} < \mathtt{n}\ \mathbf{do}\ S\ \mathbf{od}$$

For loops are just an abbreviation for a while loop:

$$\mathbf{for}\ \mathtt{i} < \mathtt{n}\ \mathbf{do}\ S\ \mathbf{od} \quad \equiv \quad \mathtt{i} := 0;\ \mathbf{while}\ \mathtt{i} \neq \mathtt{n}\ \mathbf{do}\ S;\ \mathtt{i} := s(\mathtt{i})\ \mathbf{od}$$

We will not impose any condition on the signature Σ except that we assume Σ to include a constant symbol 0 for zero and a unary function symbol s for successor. But this condition is enough to guarantee expressivity for the class of loops we are interested in, since $\mathrm{iter}(\mathbf{for}\ \mathtt{i} < \mathtt{n}\ \mathbf{do}\ \ldots\ \mathbf{od}, \sigma) = \sigma(\mathtt{n})$.

Definition 5.1. Let Γ be a finite Π_1 theory. A simple loop proof π, short slp, is a proof of the following form, together with p, Γ_i, t_i, and u_i as described below:

$$\dfrac{\dfrac{\models A[\overline{\mathtt{x}},\mathtt{n}] \to \forall y\, p[\mathtt{n},0,\overline{\mathtt{x}},y]}{\{A[\overline{\mathtt{x}},\mathtt{n}]\}\ \mathtt{i} := 0\ \{\forall y\, p[\mathtt{n},\mathtt{i},\overline{\mathtt{x}},y]\}} \quad \dfrac{\dfrac{\dfrac{\{\bigwedge_j p[\mathtt{n},\mathtt{i},\overline{\mathtt{x}},t_j]\}\ S\ \{p[\mathtt{n},s(\mathtt{i}),\overline{\mathtt{x}},\gamma]\}}{\{\forall y\, p[\mathtt{n},\mathtt{i},\overline{\mathtt{x}},y] \wedge \mathtt{i} \neq \mathtt{n}\}\ S\ \{\forall y\, p[\mathtt{n},s(\mathtt{i}),\overline{\mathtt{x}},y]\}}}{\{\forall y\, p[\mathtt{n},\mathtt{i},\overline{\mathtt{x}},y]\}\ \ldots\ \{\forall y\, p[\mathtt{n},\mathtt{i},\overline{\mathtt{x}},y] \wedge \mathtt{i} = \mathtt{n}\}} \quad \dfrac{\models \bigwedge_j p[\mathtt{n},\mathtt{n},\overline{\mathtt{x}},u_j] \to B[\overline{\mathtt{x}},\mathtt{n}]}{\models \cdots \to B[\overline{\mathtt{x}},\mathtt{n}]}}{\{\forall y\, p[\mathtt{n},\mathtt{i},\overline{\mathtt{x}},y]\}\ \ldots\ \{B[\overline{\mathtt{x}},\mathtt{n}]\}}}{\{A[\overline{\mathtt{x}},\mathtt{n}]\}\ \mathbf{for}\ \mathtt{i} < \mathtt{n}\ \mathbf{do}\ S\ \mathbf{od}\ \{B[\overline{\mathtt{x}},\mathtt{n}]\}}$$

- $p[\mathtt{n},\mathtt{i},\overline{\mathtt{x}},\gamma]$ is a quantifier-free formula with only the indicated variables (and program variables).
- $\Gamma_0 = \Gamma_0[\overline{\mathtt{x}},\mathtt{n},\beta]$, $\Gamma_1 = \Gamma_1[\mathtt{n},\mathtt{i},\overline{\mathtt{x}},\gamma]$ and $\Gamma_2 = \Gamma_2[\mathtt{n},\overline{\mathtt{x}}]$ are sets of quantifier-free instances of the background theory Γ restricting the deductions in the consequence rules of the proof:
  - The deduction $\models A[\overline{\mathtt{x}},\mathtt{n}] \to \forall y\, p[\mathtt{n},0,\overline{\mathtt{x}},y]$ only requires assumptions from Γ_0, i.e. the following sequent is quasi-tautological:

    $$\Gamma_0, A[\overline{\mathtt{x}},\mathtt{n}] \vdash p[\mathtt{n},0,\overline{\mathtt{x}},\beta]$$

  - The proof of the triple $\{\bigwedge_j p[\mathtt{n},\mathtt{i},\overline{\mathtt{x}},t_j]\}\ S\ \{p[\mathtt{n},s(\mathtt{i}),\overline{\mathtt{x}},\gamma]\}$ only requires assumptions from Γ_1, i.e. the following sequent is quasi-tautological:

    $$\Gamma_1, \bigwedge_j p[\mathtt{n},\mathtt{i},\overline{\mathtt{x}},t_j] \vdash \mathrm{wlp}(S, p[\mathtt{n},s(\mathtt{i}),\overline{\mathtt{x}},\gamma])$$

  - The deduction $\models \bigwedge_j p[\mathtt{n}, \mathtt{i}, \overline{\mathtt{x}}, u_j] \to B[\overline{\mathtt{x}}, \mathtt{n}]$ only requires assumptions from Γ_2, i.e. the following sequent is quasi-tautological:

$$\Gamma_2, \bigwedge_j p[\mathtt{n}, \mathtt{i}, \overline{\mathtt{x}}, u_j] \vdash B[\overline{\mathtt{x}}, \mathtt{n}]$$

- $t_j = t_j[\mathtt{n}, \mathtt{i}, \overline{\mathtt{x}}, \gamma]$ and $u_j = u_j[\mathtt{n}, \overline{\mathtt{x}}]$ are terms in only the indicated variables.

Theorem 5.2. *Let* π' *be a proof of* $\{A\}$ **for** $\mathtt{i} < \mathtt{n}$ **do** S **od** $\{B\}$ *such that:*

- *Any deduction in* π' *requires only assumptions from* Γ*, i.e. whenever* $\models \psi$ *occurs in the proof, then we actually have* $\Gamma \vdash \psi$.
- *The loop rule is only used once ending in* $\{\varphi\}$ **while** $\mathtt{i} \neq \mathtt{n}$ **do** $S;\ \mathtt{i} := s(\mathtt{i})$ **od** $\{\varphi \wedge \mathtt{i} = \mathtt{n}\}$ *where* φ *is a* Π_1 *formula.*

Then there is a slp π *of* $\{A\}$ **for** $\mathtt{i} < \mathtt{n}$ **do** S **od** $\{B\}$.

Proof. Normalize π' using the rules in Figure 4.2, and then apply Herbrand's theorem to obtain the sets of terms t_i and u_i. □

Because of the completeness of Hoare logic (as our language is expressive), Theorem 5.2 implies a certain form of completeness for slps as well. If $\models \{A\}$ **for** $\mathtt{i} < \mathtt{n}$ **do** S **od** $\{B\}$, Γ is large enough, and contains enough Skolem symbols, then both conditions in Theorem 5.2 are satisfied and there will be a slp of this triple.

Similar to the case of induction proofs, where we instantiated the proof for a concrete value of α, we now want to consider proofs for concrete values of $\mathtt{n}$. With that goal in mind, we will first adapt the notion of instantiation of formulas to loop unrolling of programs:

Definition 5.3. For any natural number n, the loop unrolling $(\textbf{for } \mathtt{i} < \mathtt{n} \textbf{ do } p \textbf{ od})_n$ of **for** $\mathtt{i} < \mathtt{n}$ **do** p **od** is the program $p[\mathtt{i}\backslash 0, \mathtt{n}\backslash n];\ \ldots;\ p[\mathtt{i}\backslash n, \mathtt{n}\backslash n];\ \mathtt{i} := n$.

Lemma 5.4. *Let* p *be a program that does not assign to* $\mathtt{i}$ *or* $\mathtt{n}$*, then*

$$\mathcal{M}[\![(\textbf{for } \mathtt{i} < \mathtt{n} \textbf{ do } p \textbf{ od})_{\sigma(\mathtt{n})}]\!](\sigma) = \mathcal{M}[\![\textbf{for } \mathtt{i} < \mathtt{n} \textbf{ do } p \textbf{ od}]\!](\sigma)$$

Proof. Define the program state $\sigma(i)$ after i iterations of the loop as follows ($p;\ \mathtt{i} := s(\mathtt{i})$ is the loop body of the abbreviated while loop of **for** $\mathtt{i} < \mathtt{n}$ **do** p **od**):

$$\sigma(0) = \sigma[\mathtt{i} \mapsto 0]$$
$$\sigma(i+1) = \mathcal{M}[\![p;\ \mathtt{i} := s(\mathtt{i})]\!](\sigma(i))$$

Because p does not assign to $\mathtt{i}$ or $\mathtt{n}$, we get $\sigma(i)(\mathtt{i}) = i$ and $\sigma(i)(\mathtt{n}) = \sigma(\mathtt{n})$ for any "time" $i \leq n$ and hence for any $i < n$:

$$\sigma(i+1) = (\mathcal{M}[\![p]\!](\sigma(i)))[\mathtt{i} \mapsto i] = (\mathcal{M}[\![p[\mathtt{i}\backslash 0, \mathtt{n}\backslash\sigma(\mathtt{n})]]\!](\sigma(i)))[\mathtt{i} \mapsto i]$$

Since at every "time" $i < n$ the loop condition holds, i.e. $\sigma(i) \models \mathtt{i} \neq \mathtt{n}$, and fails at the end, i.e. $\sigma(n) \models \mathtt{i} = \mathtt{n}$, we have that $\mathcal{M}[\![\textbf{for } \mathtt{i} < \mathtt{n} \textbf{ do } p \textbf{ od}]\!](\sigma) = \sigma(n)$.

The state update $[\mathtt{i} \mapsto i]$ commutes with the program semantics of $p[\mathtt{i}\backslash 0, \mathtt{n}\backslash\sigma(\mathtt{n})]$, therefore $\mathcal{M}[\![(\textbf{for } \mathtt{i} < \mathtt{n} \textbf{ do } p \textbf{ od})_{\sigma(\mathtt{n})}]\!](\sigma) = \sigma(n) = \mathcal{M}[\![\textbf{for } \mathtt{i} < \mathtt{n} \textbf{ do } p \textbf{ od}]\!](\sigma)$. □

By extending the language of our theory by symbols for the state $\sigma(i)$ of the loop after i iterations, we can encode the triple we are interested in into a sequent.

Definition 5.5. The extended language $\Sigma_\sigma = \Sigma \cup \{\sigma_{\mathtt{x}_1}/1, \ldots, \sigma_{\mathtt{x}_l}/1\}$ consists of the language Σ and new unary function symbols $\sigma_{\mathtt{x}_j}$ such that $\sigma_{\mathtt{x}_j}(i)$ denotes the state of the program variable $\mathtt{x}_j$ after i iterations.

Definition 5.6. Given a background theory Γ and a triple $\{A\}$ **for** $\mathtt{i} < \mathtt{n}$ **do** S **od** $\{B\}$ with program variables $\{\mathtt{i}, \mathtt{n}, \mathtt{x}_1, \ldots, \mathtt{x}_l\}$ where the loop body S does not assign to $\mathtt{i}$ or $\mathtt{n}$, the associated end-sequent of $\{A\}$ **for** $\mathtt{i} < \mathtt{n}$ **do** S **od** $\{B\}$ is the following sequent in the extended language Σ_σ:

$$\underbrace{\Gamma, \Pi[\alpha], A[\alpha, \overline{\sigma}(0)]}_{\forall\tilde{\Gamma}[\alpha]} \vdash \underbrace{B[\alpha, \overline{\sigma}(\alpha)]}_{\tilde{B}[\alpha]}$$

where $\Pi[\alpha] \equiv \forall z\, \pi[\alpha, z]$, and $\pi[\alpha, z] \equiv \overline{\sigma}(s(z)) = \mathcal{M}[\![S[\mathtt{i}\backslash z, \mathtt{n}\backslash\alpha]]\!](\overline{\sigma}(z))$.

Definition 5.7. Let π be a slp; its associated sip $\tilde{\pi}$ is the LK-proof with induction of its associated end-sequent with the induction formula $F[\alpha, \nu, \gamma] \equiv p[\alpha, \nu, \overline{\sigma}(\nu), \gamma]$.

Not all sips arise from slps, as not all end-sequents arise from Hoare triples in the sense of Definition 5.6. Not even all sips of an associated end-sequent arise from slps, as those that are associated sips of an slp have the following restrictions:

Lemma 5.8. *Let π be any sip of the associated end-sequent of the triple $\{A\}\ p\ \{B\}$. If π satisfies the following properties, then it is the associated sip of some slp ψ, i.e. $\pi = \tilde{\psi}$.*

- *There is only one instance of $\Pi[\alpha]$, and it is $\pi[\alpha, \nu] \in \Gamma_1$.*
- *The arguments of the function symbols $\sigma_{\mathtt{x}_i}$ in the instance terms are restricted:*
  - *In Γ_0 they are always 0.*
  - *In $\Gamma_1 \setminus \{\pi[\alpha, \nu]\}$ and the terms t_i they are always ν.*
  - *In the induction formula, they are always ν.*
  - *In Γ_2 and the terms u_i they are always α.*
- *The formula $A[\alpha, \overline{\sigma}(0)]$ occurs in Γ_0, but not in Γ_1 or Γ_2.*

These restrictions will matter when we will try to find slps of a given triple, i.e. invert the following the following constructions. However the “forward” case is identical to the case of sips:

Definition 5.9. The instance proof of an slp π for a number n is the instance proof $\tilde{\pi}_n$ the sip $\tilde{\pi}$.

Definition 5.10. An slp grammar is an sip grammar in the language Σ_σ where each production is additionally required to be in one of the following forms (where t is an arbitrary term in just Σ):

- $\tau \to t[\alpha, \beta, \overline{\sigma}(0)]$
- $\tau \to t[\alpha, \nu, \gamma, \overline{\sigma}(\nu)]$
- $\tau \to t[\alpha, \overline{\sigma}(\alpha)]$
- $\gamma \to t[\alpha, \nu, \gamma, \overline{\sigma}(\nu)]$
- $\gamma_{\text{end}} \to t[\alpha, \overline{\sigma}(\alpha)]$

The instance grammar of an slp grammar is just the instance grammar of an sip grammar.

Definition 5.11. The slp grammar $G(\pi)$ of an slp π is the sip grammar grammar $G(\tilde{\pi})$ of its associated sip, i.e. $G(\pi) = G(\tilde{\pi})$.

Lemma 5.12. *Let π be an slp, then $G(\pi)$ is an slp grammar.*

Theorem 5.13. *Let π be an slp, then $G(\pi_n) = G(\pi)_n$*

Proof. This is a special case of Lemma 3.8. □

5.1 Finding slp grammars

Note that per Definition 5.10, an slp grammar is just a sip grammar in the signature Σ_σ where the arguments of the function symbols σ_k are further restricted—hence we can find minimal slp grammars in essentially the same way as sip grammars; we take a polynomially-sized slp grammar H and minimize it.

Definition 5.14. A production $\delta \to k[\alpha, \nu, \beta, \gamma, \overline{\sigma}(s)]$ is in slp-normal form relative to a family of languages (L_i) if it is of one of the forms in Definition 5.10 and there is a term r such that $\delta \to k[\alpha, \nu, \beta, \gamma, \overline{\sigma}(r)]$ is in sip-normal form relative to (L_i).

Lemma 5.15. *Let G be a slp grammar, $\delta \to k$ one of its productions, L_i an instance language for the number i.*

Then there exists a production $\delta \to k''$ in slp-normal form such that $\mathcal{L}(G_i) \cap L_i \subseteq \mathcal{L}(G'_i) \cap L_i$ for the slp grammar G' which is obtained from G by replacing the production $\delta \to k$ by $\delta \to k''$.

Proof. Assume for the moment that k contains an occurrence of a $\overline{\sigma}(\nu)$; and let k' be as in Lemma 1.28. Then in k', all of these occurrences still have the same argument, call it t_ν. Then we can apply Lemma 3.22 to the term $k'' = k'[\overline{\sigma}(t_\nu) \backslash \overline{\sigma}(\nu)]$ where all occurrences of t_ν as arguments of $\overline{\sigma}$ have been replaced by ν.

The same argument also works if k contains $\overline{\sigma}(0)$, $\overline{\sigma}(\alpha)$, or no $\overline{\sigma}$ at all. □

Lemma 5.16. *Let (L_i) be a family of languages, such that there is slp grammar covering it. Then there is a slp grammar $H = (\Sigma, Q)$ that contains a slp grammar covering (L_i) of minimal size as a sub-grammar.*

This grammar can be computed by an algorithm whose runtime is polynomially bounded in $|\mathrm{st}(\bigcup L_i)|$. The number of productions in H and the number of subterms of right-hand sides of productions in H is then also polynomially bounded.

Proof. The slp grammar $\tilde{H}$ consists of all productions in slp-normal form.

If G is a slp grammar of minimal size covering (L_i), then by iteratively applying Lemma 5.15 there is a grammar G' with $|G'| \leq |G|$ where all productions are in slp-normal form, i.e. a sub-grammar of H, and which still covers (L_i). □

However, here it can be the case that there is no slp grammar covering a given finite family of languages (L_i); in this case the algorithm fails—for example, there is no slp grammar that covers $L_2 = \{r_1(\sigma_1(0), \sigma_1(1), \sigma_1(2))\}$:

Lemma 5.17. *There is no slp grammar G such that $r_1(\sigma_1(0), \sigma_1(1), \sigma_1(2)) \in G_2$.*

This result is not too astonishing—it corresponds to a proof referencing the program variables at three different points in time.

Proof. Let G be a slp grammar, and d a derivation of $r_1(\sigma_1(0), \sigma_1(1), \sigma_1(2))$ in G_2.

Consider the first step in d which is not a non-terminal—this is either $\tau \to^1 r_1(t_1, t_2, t_3)$ or $\gamma_i \to^1 r_1(t_1, t_2, t_3)$ for some terms t_1, t_2, t_3. Clearly each t_j subsumes $\sigma_1(j)$ and by case analysis on the possible productions in an instance grammar of a slp grammar, either t_j is a non-terminal or equal to $\sigma(i)$. The terms t_j cannot be any non-terminal either, they have to be γ, as the other non-terminals are instantiated with numerals. Due to the rigidity of G_2, there are now only two possible values for the arguments of r_1, which is a contradiction. □

5.2 Finding loop invariants

If we look at how we encoded slps as sips in Definition 5.7, the loop invariant p turned into the induction formula.

Definition 5.18. A schematic slp S is a schematic sip with the same restrictions as in Lemma 5.8.

Definition 5.19. A formula F is a solution for the schematic slp S if:

- F is solution for the schematic sip S.
- F does not contain any occurrences of the function symbols $\sigma_{\mathtt{x}}$, except for $\sigma_{\mathtt{x}}(\nu)$, i.e. F is of the form $F[\alpha, \nu, \overline{\sigma}(\nu), \gamma]$.

Lemma 5.20. *Let π be a slp and S its schematic slp. The loop invariant $\forall y\, p[\mathtt{n}, \mathtt{i}, \overline{\mathtt{x}}, y]$ then gives the solution $p[\alpha, \nu, \overline{\sigma}(\nu), \gamma]$ for the schematic slp.*

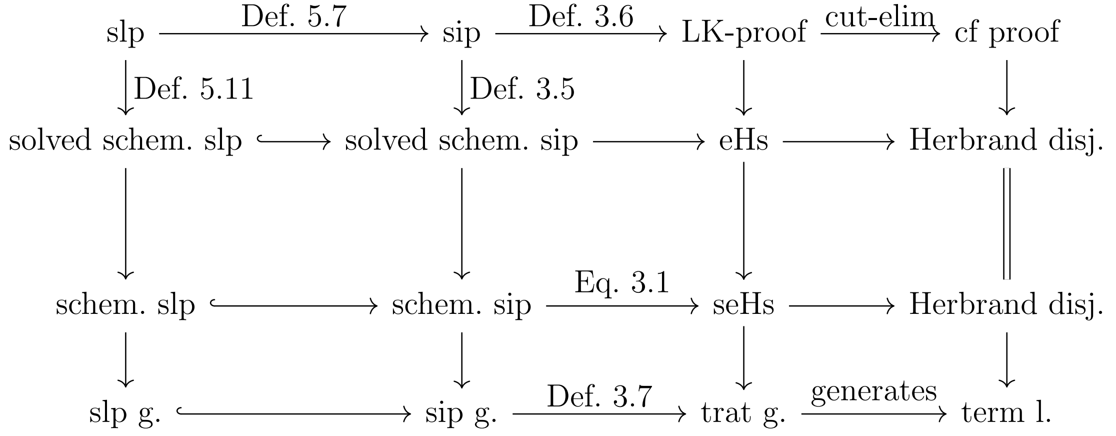


Figure 5.1: Relation of slps and their grammars to sips, proofs with cut and cut-free proofs.

Proof. This directly follows from the analogous result for sips, Lemma 3.12. □

Lemma 5.21. *Let S be a schematic slp with solution $F[\alpha, \nu, \overline{\sigma}(\nu), \gamma]$, then there is a slp π which has S as schematic slp and $\forall y\, F[\mathtt{n}, \mathtt{i}, \overline{\mathtt{x}}, y]$ as loop invariant.*

Proof. Use Lemma 3.13 to obtain a sip for the associated end-sequent, and then convert it into a slp with Lemma 5.8. □

We can then use the same algorithm that we used to find loop invariants for loop invariants of slps; it is however necessary to filter out generated solutions which do not satisfy the restriction in Definition 5.19.

5.3 Overview

Given a triple $\{A\}$ **for** $\mathtt{i} < \mathtt{n}$ **do** S **od** $\{B\}$, our algorithm tries to find a simple loop proof using the following steps:

1. For a suitable set of values for n, automatically generate proofs of the sequents $\Gamma, \Pi[n], A[\overline{\mathtt{x}}\backslash\overline{\sigma}(0), \mathtt{i}\backslash 0, \mathtt{n}\backslash n] \vdash B[\overline{\mathtt{x}}\backslash\overline{\sigma}(n), \mathtt{i}\backslash n, \mathtt{n}\backslash n]$, and extract the Herbrand language L_n.
2. Find a slp grammar G with such that $\mathcal{L}(G_n) \supseteq L_n$ for each of the languages L_n.
3. Find a solution F for the schematic form using one of the algorithms presented above.
4. This gives a slp.

5.4 Examples

In the following, we will present a few examples illustrating the theory of slps.

5.4.1 Addition

As a very simple first example, consider the following program which implements addition using only the successor operation:

$$\{\mathtt{x} = x_0\}\ \textbf{for}\ \mathtt{i} < \mathtt{n}\ \textbf{do}\ \mathtt{x} := s(\mathtt{x})\ \textbf{od}\ \{\mathtt{x} = x_0 + \mathtt{n}\}$$

The background theory necessary to prove the correctness of this loop is as basic as the loop itself; we only need two axioms describing the relation between addition and successor:

$$\Gamma = \{\forall x\, x + 0 = x, \tag{p0}$$
$$\forall xy\, x + s(y) = s(x + y)\} \tag{ps}$$

The labels on the right hand side are the indices we will use in the corresponding term languages, e.g. the instance $c + 0 = c$ of the formula φ_{p0} corresponds to the term $r_{\mathrm{p0}}(c)$.

The associated end-sequent of the triple is the following:

$$\sigma(0) = x_0, \forall i\, \sigma(s(i)) = s(\sigma(i)), \forall x\, \varphi_{\mathrm{p0}}, \forall xy\, \varphi_{\mathrm{ps}} \vdash \sigma(\alpha) = x_0 + \alpha$$

What we are now interested in are languages of instance proofs, these are just another way to look at the Herbrand sequents of the above associated end-sequent. We can easily guess a few of those:

$$\begin{aligned}
L_0 &= \{r_{\mathrm{p0}}(x_0), r_{\mathrm{A}}, r_{\mathrm{B}}\} \\
L_1 &= \{r_{\mathrm{p0}}(x_0), r_{\mathrm{ps}}(x_0, 0), r_{\mathrm{A}}, r_{\mathrm{B}}, r_\pi(0)\} \\
L_2 &= \{r_{\mathrm{p0}}(x_0), r_{\mathrm{ps}}(x_0, 0), r_{\mathrm{ps}}(x_0, 1), r_{\mathrm{A}}, r_{\mathrm{B}}, r_\pi(0), r_\pi(1)\} \\
&\vdots \\
L_{i+1} &= \{r_{\mathrm{p0}}(x_0), r_{\mathrm{ps}}(x_0, 0), \ldots, r_{\mathrm{ps}}(x_0, i), r_{\mathrm{A}}, r_{\mathrm{B}}, r_\pi(0), \ldots, r_\pi(i)\} \\
&\vdots
\end{aligned}$$

For example for L_2, this intuitively means that in order to prove $\{\mathtt{x} = x_0\}\ \mathtt{x} := s(\mathtt{x});\ \mathtt{x} := s(\mathtt{x})\ \{\mathtt{x} = x_0 + 2\}$ we do not need the full theory Γ, we can prove this triple just from $x_0 + 0 = x_0$ and $x_0 + s(0) = s(x_0 + 0)$.

In this case, we can easily give a nice and uniform description of L_i; but in practice we only need a few finite languages.

The following slp grammar covers all of the languages L_i, since we can obtain any term in those languages as the right hand side of a τ-production (after substituting i for ν):

$$\begin{aligned}
\tau &\to r_{\mathrm{p0}}(x_0) \\
\tau &\to r_{\mathrm{ps}}(x_0, \nu) \\
\tau &\to r_{\mathrm{A}} \mid r_{\mathrm{B}} \mid r_\pi(\nu)
\end{aligned}$$

If this slp grammar was extracted from a slp, then we would immediately know what instances were in Γ_0, Γ_1, and Γ_2 in the slp—that would be $\Gamma_0 = \{x_0 + 0 = x_0\}$,

$\Gamma_1 = \{x_0 + 0 = x_0, x_0 + s(\nu) = s(x_0 + \nu)\}$, and $\Gamma_2 = \{x_0 + 0 = x_0\}$. We could also recover the sets of terms t_i and u_i; but in this case there are none, since we have no γ- or γ_{end}-productions (this implies that the loop invariant is quantifier-free). Putting all this information together, we can give the schematic slp for this problem:

$$x_0 + 0 = x_0, \sigma(0) = x_0 \vdash X[\alpha, 0, \sigma(0), \beta]$$
$$X[\alpha, \nu, \sigma(\nu), 0],$$
$$x_0 + 0 = x_0, x_0 + s(\nu) = s(x_0 + \nu),$$
$$\sigma(s(\nu)) = s(\sigma(\nu)) \vdash X[\alpha, s(\nu), \sigma(s(\nu))\gamma]$$
$$X[\alpha, \alpha, \sigma(\alpha), 0], x_0 + 0 = x_0 \vdash \sigma(\alpha) = x_0 + \alpha$$

The only missing piece in order to reconstruct a slp is now a valid substitution for X, i.e. a solution for the schematic slp, which is nothing more than a loop invariant.

As we have seen in Lemma 3.30, any loop invariant is a generalization of a consequence of the canonical solution, which is given as follows:

$$C'_0 = x_0 + 0 = x_0 \land \sigma(0) = x_0$$
$$C'_1 = x_0 + 0 = x_0 \land \sigma(0) = x_0 \land x_0 + s(0) = s(x_0 + 0) \land \sigma(1) = s(\sigma(0))$$
$$C'_2 = x_0 + 0 = x_0 \land \sigma(0) = x_0 \land x_0 + s(0) = s(x_0 + 0) \land \sigma(1) = s(\sigma(0))$$
$$\land\, x_0 + s(1) = s(x_0 + 1) \land \sigma(2) = s(\sigma(1))$$

Luckily, these formulas are already in CNF. Paramodulating the two clauses of C'_0, we can obtain the formula $x_0 + 0 = \sigma(0)$; if we replace some occurrences of 0 by ν, we get a candidate for a loop invariant:

$$X[\alpha, \nu, \sigma(\nu), \gamma] \quad \mapsto \quad x_0 + \nu = \sigma(\nu)$$

Applying this substitution in the schematic slp, we can verify that it is indeed a loop invariant:

$$x_0 + 0 = x_0, \sigma(0) = x_0 \vdash x_0 + 0 = \sigma(0)$$
$$x_0 + \nu = \sigma(\nu), x_0 + s(\nu) = s(x_0 + \nu),$$
$$\sigma(s(\nu)) = s(\sigma(\nu)) \vdash x_0 + s(\nu) = \sigma(s(\nu))$$
$$x_0 + \alpha = \sigma(\alpha) \vdash \sigma(\alpha) = x_0 + \alpha$$

5.4.2 Array initialization

This problem is the `Shift` example from [HKV11].

$$\{\top\}\ \textbf{for}\ \mathtt{i} < \mathtt{n}\ \textbf{do}\ \mathtt{x} := \mathrm{set}(x, s(i), \mathrm{get}(x, i))\ \textbf{od}\ \{\forall y\, y \leq n \rightarrow \mathrm{get}(x, y) = \mathrm{get}(x, 0)\}$$

We cannot handle this problem directly, as the postcondition is quantified; we can handle this by Skolemization:

$$\{\top\}\ \textbf{for}\ \mathtt{i} < \mathtt{n}\ \textbf{do}\ \mathtt{x} := \mathrm{set}(x, s(i), \mathrm{get}(x, i))\ \textbf{od}\ \{k \leq n \rightarrow \mathrm{get}(x, k) = \mathrm{get}(x, 0)\}$$

This loop initializes an array by setting each element in turn to its predecessor, and we want to verify that the array is then constant, and all elements equal to the first one.

We formalize arrays as individuals in our first order language, and add suitable axioms for two basic array operations:

- $\mathrm{get}(x, y)$ retrieves the element at index y in the array x.
- $\mathrm{set}(x, y, z)$ returns the array x where the element at index y is replaced with the value z.

The background theory also includes some basic axioms for the natural numbers and their ordering $\leq$.

$$\begin{aligned}
\Gamma = \{&\forall x\, s(x) \neq 0, && \text{(s0)}\\
&\forall x\, x \leq x, && \text{(lr)}\\
&\forall x\, 0 \leq x, && \text{(0l)}\\
&\forall x\, x \leq 0 \rightarrow x = 0, && \text{(l0)}\\
&\forall x \forall y\, x \leq y \rightarrow x \leq s(y), && \text{(sl)}\\
&\forall x \forall y\, x \leq s(y) \rightarrow x \leq y \vee x = s(y), && \text{(ls)}\\
&\forall x \forall y \forall z\, \mathrm{get}(\mathrm{set}(x, y, z), y) = z, && \text{(ge)}\\
&\forall x \forall y \forall z \forall w\, y \neq w \rightarrow \mathrm{get}(\mathrm{set}(x, y, z), w) = \mathrm{get}(x, w)\} && \text{(gn)}
\end{aligned}$$

We can find possible languages for instance proofs:

$$\begin{aligned}
L_0 = \{&r_{\mathrm{l0}}(k), r_{\mathrm{A}}, r_{\mathrm{B}}\}\\
L_2 = \{&r_{\mathrm{l0}}(k), r_{\mathrm{ls}}(k, 0), r_{\mathrm{ls}}(k, 1), r_{\mathrm{gn}}(\sigma(0), 1, \mathrm{get}(\sigma(0), 0), 0),\\
&r_{\mathrm{gn}}(\sigma(1), 2, \mathrm{get}(\sigma(1), 0), 0), r_{\mathrm{gn}}(\sigma(1), 2, \mathrm{get}(\sigma(1), 0), k),\\
&r_{\mathrm{ge}}(\sigma(0), 1, \mathrm{get}(\sigma(0), 0)), r_{\mathrm{ge}}(\sigma(1), 2, \mathrm{get}(\sigma(1), 0)),\\
&r_{\pi}(0), r_{\pi}(1), r_{\mathrm{A}}, r_{\mathrm{B}}\}
\end{aligned}$$

And then guess a slp grammar covering these instance languages:

$$\begin{aligned}
\tau &\rightarrow r_{\mathrm{l0}}(k)\\
\tau &\rightarrow r_{\mathrm{ls}}(k, \nu)\\
\tau &\rightarrow r_{\mathrm{ge}}(\sigma(\nu), s(\nu), \mathrm{get}(\sigma(\nu), \nu))\\
\tau &\rightarrow r_{\mathrm{gn}}(\sigma(\nu), s(\nu), \mathrm{get}(\sigma(\nu), \nu), 0)\\
\tau &\rightarrow r_{\mathrm{gn}}(\sigma(\nu), s(\nu), \mathrm{get}(\sigma(\nu), \nu), k)\\
\tau &\rightarrow r_{\pi}(\nu)\\
\tau &\rightarrow r_{\mathrm{A}}\\
\tau &\rightarrow r_{\mathrm{B}}
\end{aligned}$$

As in the last example, the absence of a γ-production implies that the loop invariant will be quantifier-free. From the slp grammar, we can read off the schematic slp–making sure to insert 0 as a "dummy" step-term for the quantifier-free invariant:

$$\top, k \leq 0 \to k = 0 \vdash X[\alpha, 0, \sigma(0), \gamma]$$
$$\sigma(s\nu) = \mathrm{set}(\sigma(\nu), s\nu, \mathrm{get}(\sigma(\nu), \nu)),$$
$$k \leq s(\nu) \to k \leq \nu \lor k = s(\nu),$$
$$\mathrm{get}(\mathrm{set}(\sigma(\nu), s(\nu), \mathrm{get}(\sigma(\nu), \nu)), s(\nu)) = \mathrm{get}(\sigma(\nu), \nu),$$
$$s(\nu) \neq k \to \mathrm{get}(\mathrm{set}(\sigma(\nu), s(\nu), \mathrm{get}(\sigma(\nu), 0)), k) = \mathrm{get}(\sigma(\nu), k),$$
$$s(\nu) \neq 0 \to \mathrm{get}(\mathrm{set}(\sigma(\nu), s(\nu), \mathrm{get}(\sigma(\nu), 0)), 0) = \mathrm{get}(\sigma(\nu), 0),$$
$$\sigma(s(\nu)) = \mathrm{set}(\sigma(\nu), s(\nu), \mathrm{get}(\sigma(\nu), \nu)),$$
$$X[\alpha, \nu, \sigma(\nu), 0] \vdash X[\alpha, s(\nu), \sigma(s\nu), \gamma]$$
$$X[\alpha, \alpha, \sigma(\alpha), k] \vdash k \leq \alpha \to \mathrm{get}(\sigma(\alpha), k) = \mathrm{get}(\sigma(\alpha), 0)$$

And compute the canonical solutions; these are already converted into CNF and the clauses are labelled so that we can refer to them later on:

$$C'_0[\alpha, \gamma] = k \not\leq 0 \lor k = 0$$
$$C'_1[\alpha, \gamma] = (\sigma(1) = \mathrm{set}(\sigma(0), 1, \mathrm{get}(\sigma(0), 0))) \quad (0)$$
$$\land (k \not\leq 1 \lor k \leq 0 \lor k = 1) \quad (1)$$
$$\land (1 = 0 \lor \mathrm{get}(\mathrm{set}(\sigma(0), 1, \mathrm{get}(\sigma(0), 0)), 0) = \mathrm{get}(\sigma(0), 0)) \quad (2)$$
$$\land (1 = k \lor \mathrm{get}(\mathrm{set}(\sigma(0), 1, \mathrm{get}(\sigma(0), 0)), k) = \mathrm{get}(\sigma(0), k)) \quad (3)$$
$$\land \mathrm{get}(\mathrm{set}(\sigma(0), 1, \mathrm{get}(\sigma(0), 0)), 1) = \mathrm{get}(\sigma(0), 0) \quad (4)$$
$$\land (k \not\leq 0 \lor k = 0) \quad (5)$$

A solution F for the schematic slp needs to satisfy $C'_1[\alpha, \gamma] \vdash F[\alpha, 1, \sigma(1), \gamma]$. We will now try to apply the heuristic search algorithm using paramodulation and resolution to find such a consequence $D[\alpha, \gamma]$ of C'_1. The quasi-tautology criterion for pruning branches in this search algorithm is $D[1, \gamma] \vdash k \leq 1 \to \mathrm{get}(\sigma(1), k) = \mathrm{get}(\sigma(1), 0)$

$$k \not\leq 1 \lor k = 0 \lor k = 1 \quad (6, \mathrm{res}(1,5))$$
$$k \not\leq 1 \lor k = 1 \lor \mathrm{get}(\mathrm{set}(\sigma(0), 1, \mathrm{get}(\sigma(0), 0)), 0) = \mathrm{get}(\sigma(0), k) \quad (7, \mathrm{param}(6,2))$$
$$k \not\leq 1 \lor k = 1 \lor \mathrm{get}(\mathrm{set}(\sigma(0), 1, \mathrm{get}(\sigma(0), 0)), k) = \mathrm{get}(\mathrm{set}(\sigma(0), 1, \mathrm{get}(\sigma(0), 0)), 0) \quad (8, \mathrm{param}(7,3))$$
$$1 = 0 \lor \mathrm{get}(\mathrm{set}(\sigma(0), 1, \mathrm{get}(\sigma(0), 0)), 1) = \mathrm{get}(\mathrm{set}(\sigma(0), 1, \mathrm{get}(\sigma(0), 0)), 0) \quad (9, \mathrm{param}(4,2))$$
$$k \not\leq 1 \lor k = 1 \lor \mathrm{get}(\sigma(1), k) = \mathrm{get}(\sigma(1), 0) \quad (10, \mathrm{param}(0,8))$$
$$1 = 0 \lor \mathrm{get}(\sigma(1), 1) = \mathrm{get}(\sigma(1), 0) \quad (11, \mathrm{param}(0,9))$$

Taking the conjunction of clauses 10 and 11 gives the following consequence of $C'_1[\alpha, \gamma]$:

$$(k \leq 1 \land k \neq 1 \to \mathrm{get}(\sigma(1), k) = \mathrm{get}(\sigma(1), 0)) \land (0 \neq 1 \to \mathrm{get}(\sigma(1), 1) = \mathrm{get}(\sigma(1), 0)$$

Generalizing this formula gives a candidate for a loop invariant:

$$(k \leq \mathtt{i} \land k \neq \mathtt{i} \to \mathrm{get}(\mathtt{x}, k) = \mathrm{get}(\mathtt{x}, 0)) \land (0 \neq \mathtt{i} \to \mathrm{get}(\mathtt{x}, \mathtt{i}) = \mathrm{get}(\mathtt{x}, 0)$$

Which is indeed a loop invariant as can be checked by substituting it for X in the schematic slp. We did not quite follow the algorithm though, we did not forget the clauses we paramodulated—we used clauses 2 and 0 twice in order to generate the consequence. It is not immediately apparent how to do this with forgetful paramodulation and resolution.

5.4.3 Bubble sort

Bubble sort is a naive algorithm to sort lists, in each iteration the list is traversed from end to beginning, and adjacent elements are swapped if they are in the wrong order:

```
for i < n do
   for j ∈ {n − 1, . . . , i + 1} do
      if get(x, j) < get(x, j − 1) then
         tmp := get(x, j);
         x := set(x, j, get(x, j − 1));
         x := set(x, j − 1, tmp)
      fi
   od
od
```

Figure 5.2: The bubble sort algorithm

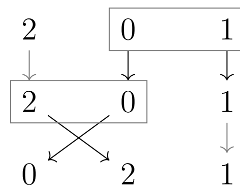


Figure 5.3: Inner loop of bubble sort, “bubbling down” the element 0

Note that this inner loop changes the indices of the elements (which should not be a surprise since sorting needs to permute the list in general). But this change of indices will make a quantified invariant necessary. By contrast, in the previous array initialization example, we could have given a quantified invariant as well, namely $\forall y\, y \leq \mathtt{i} \to \mathrm{get}(\mathtt{x}, y) = \mathrm{get}(\mathtt{x}, 0)$, but after Skolemization we got away with a (slightly more complicated) quantifier-free invariant by instantiating y with k.

Since our approach cannot handle nested loops, we will only consider the outer loop here and axiomatize the inner loop; the inner loop is modeled as a function $f(x,i,n)$ operating on the array x between the indices i and n. We arrive at the following triple for the outer loop:

$$\{\top\}\ \textbf{for}\ \texttt{i} < \texttt{n}\ \textbf{do}\ \texttt{x} := f(\texttt{x}, i, n)\ \textbf{od}\ \{k \le l \land l < n \to \text{get}(\texttt{x}, k) \le \text{get}(\texttt{x}, l)\}$$

In order to keep this example short and readable, we will simplify the background theory a bit. We will ignore the details of the axioms for s, $<$, etc.—in fact, we will ignore these axioms at all. In addition, we will pack all information about the inner loop into a single axiom:

$$\begin{aligned}\Gamma = \{&\Pi_1\text{-theory of } s, <, \le, \text{get, and set},\\ &\forall x \forall i \forall n \forall j\ (i \le j \land j < n \to \text{get}(f(x,i,n),i) \le \text{get}(f(x,i,n),j))\\ &\land (j < i \land i < n \to \text{get}(f(x,i,n),j) = \text{get}(x,j))\\ &\land (j < n \to \text{get}(f(x,i,n),j) = \text{get}(x, g(x,i,n,j)))\\ &\land (i \le j \land j < n \to i \le g(x,i,n,j) \land g(x,i,n,j) < n)\}\end{aligned} \tag{f}$$

In order to axiomatize the fact that the inner loop permutes the elements it would have been nice if we could say that $\forall x\, \exists y\, \text{get}(f(x,i,n),j) = \text{get}(x,y)$. But since we can only handle Π_1 theories, the existential quantifier has been Skolemized, giving the new function $g(x,i,n,j)$ describing the permutation of the j-th element of the list x in the i-th iteration.

Just as we have done in the previous two examples, we will find a few instance languages:

$$\begin{aligned}L_0 &= \{\dots\}\\ L_1 &= \{\dots\}\\ L_2 &= \{r_{\text{f}}(\sigma(0),0,2,g(\sigma(1),1,2,1)),\\ &\qquad r_{\text{f}}(\sigma(1),1,2,0),\dots\}\\ L_3 &= \{r_{\text{f}}(\sigma(0),0,3,k),\\ &\qquad r_{\text{f}}(\sigma(0),0,3,l),\\ &\qquad r_{\text{f}}(\sigma(0),0,3,g(\sigma(2),2,3,l)),\\ &\qquad r_{\text{f}}(\sigma(0),0,3,g(\sigma(1),1,3,l)),\\ &\qquad r_{\text{f}}(\sigma(0),0,3,g(\sigma(1),1,3,g(\sigma(2),2,3,l))),\\ &\qquad r_{\text{f}}(\sigma(1),1,3,k),\\ &\qquad r_{\text{f}}(\sigma(1),1,3,l),\\ &\qquad r_{\text{f}}(\sigma(1),1,3,g(\sigma(2),2,3,l)),\\ &\qquad r_{\text{f}}(\sigma(2),2,3,k),\\ &\qquad r_{\text{f}}(\sigma(2),2,3,l),\dots\}\end{aligned}$$

The dots are instances of the axioms of s, $<$, etc., that we are still choosing to ignore. For L_0 and L_1 we did not need any information about the inner loop at all, as empty and

singleton lists are always sorted anyways. In L_2 and L_3 we can see how the Herbrand languages keep track of the change in indices after each iteration—by iterated application of g. This iterated application corresponds to the production $\gamma \to g(\sigma(\nu), \nu, \alpha, \gamma)$ in the following slp grammar which covers $L_0, L_1, L_2, \ldots$:

$$\begin{aligned}
\tau &\to r_{\mathrm{f}}(\sigma(\nu), \nu, \alpha, k) \\
\tau &\to r_{\mathrm{f}}(\sigma(\nu), \nu, \alpha, \gamma) \\
\tau &\to \ldots \\
\gamma &\to \gamma \\
\gamma &\to g(\sigma(\nu), \nu, \alpha, \gamma) \\
\gamma_{\mathrm{end}} &\to l
\end{aligned}$$

Just as before, we can again read off a schematic slp:

$$\begin{aligned}
\top, \ldots &\vdash X[\alpha, 0, \sigma(0), \gamma] \\
\varphi_f(\sigma(\nu), \nu, \alpha, k), \varphi_f(\sigma(\nu), \nu, \alpha, \gamma), \ldots, & \\
\sigma(s(\nu)) = f(\sigma(\nu), \nu, \alpha), & \\
X[\alpha, \nu, \sigma(\nu), \gamma], & \\
X[\alpha, \nu, \sigma(\nu), g(\sigma(\nu), \nu, \alpha, \gamma)] &\vdash X[\alpha, s(\nu), \sigma(s(\nu)), \gamma] \\
X[\alpha, \alpha, \sigma(\alpha), l], \ldots &\vdash k \leq l \land l < n \to \mathrm{get}(\sigma(\alpha), k) = \mathrm{get}(\sigma(\alpha), l)
\end{aligned}$$

Because the slp grammar contained γ-productions, we have more occurrences of X on the antecedent of the second sequent. Each of these X corresponds to a γ-production of the slp grammar, and the fourth argument of each X is the right hand side of the corresponding γ-production—in the slp, these would be the terms t_1 and t_2 instantiating the quantified loop invariant. The canonical solution is a bit more complicated as well due to the γ-productions:

$$\begin{aligned}
C'_0 &= \ldots \\
C'_1 &= \cdots \land \sigma(1) = f(\sigma(0), 0, \alpha) \land \\
&\quad \varphi_f(\sigma(0), 0, \alpha, k) \land \varphi_f(\sigma(0), 0, \alpha, \gamma) \\
C'_2 &= \cdots \land \sigma(1) = f(\sigma(0), 0, \alpha) \land \sigma(2) = f(\sigma(1), 1, \alpha) \land \\
&\quad \varphi_f(\sigma(0), 0, \alpha, k) \land \varphi_f(\sigma(0), 0, \alpha, \gamma) \land \varphi_f(\sigma(0), 0, \alpha, g(\sigma(1), 1, \alpha, \gamma)) \land \\
&\quad \varphi_f(\sigma(1), 1, \alpha, k) \land \varphi_f(\sigma(1), 1, \alpha, \gamma)
\end{aligned}$$

While in the previous two examples with quantifier-free loop invariants, we only had to add new conjuncts in each C'_i, here we actually have to substitute γ in C'_{i-1} by each possible right hand side of a γ-production.

However there is no change in how we can find a loop invariant, we still consider (the right) consequences of C'_i:

$$C'_2 \vdash k < \gamma \land \gamma < \alpha \land k < 2 \land 2 \leq \alpha \to \mathrm{get}(\sigma(2), k) \leq \mathrm{get}(\sigma(2), \gamma)$$

And then generalize them into a loop invariant:

$$\forall y\,(k < y \land y < \mathtt{n} \land k < \mathtt{i} \land \mathtt{i} \leq \mathtt{n} \rightarrow \mathrm{get}(\mathtt{x}, k) \leq \mathrm{get}(\mathtt{x}, y))$$

That this is indeed a loop invariant can be checked by substituting it for X in the schematic slp.

Conclusion

This thesis applies the method for inductive theorem proving proposed in [EH15] to the verification of simple loop programs by encoding the verification conditions as an inductive problem, requiring various extensions to the basic induction case covered there:

The encoding of the verification conditions as an inductive problem yields more general end-sequents than previously considered. In particular, it was necessary to allow α in the antecedent, i.e. allow sequents of the form $\Gamma[\alpha] \vdash B[\alpha]$ instead of just $\Gamma \vdash B[\alpha]$.

Aside from the end-sequent, the notion of grammar corresponding to slps proved challenging. While every slp grammar is a sip grammar, slp grammars are restricted in two significant ways: Some productions are required to be present in every slp grammar; this is a straightforward (although novel) restriction that requires little change in the rest of the theory.

However, slp grammars also have a more fundamental restriction on the form of their productions, namely that only certain terms are allowed as arguments to the function symbols $\sigma_{\mathtt{x}}$; this restriction had major effects:

- In contrast to sip grammars, we lose the theoretical result that every finite family of languages can be covered by a sip grammar.
- The notion of normal forms introduced in [EH14] and used in [EH15] to efficiently find (sip) grammars covering a finite family of languages turned out to be too strict—using that definition, we could no longer find a covering slp grammar that consists only of productions in normal form. This work introduces a new notion of normal form which is more general, while still preserving polynomial-time reducibility to Max-SAT.

While adapting this method to slp grammars, it proved convenient to reformulate the existing work on normal forms and grammar minimization in order to elucidate the points where changes related to slp grammars are necessary. Previously this method was presented as a monolithic process taking a family of languages and returning a formula whose Max-SAT solution encoded a covering grammar of minimal size.

In this work we separated this process into two steps: First we generate a covering grammar consisting of all productions in normal form, and then we encode the minimization as a Max-SAT problem. Each of these steps requires different changes for slp grammars. The covering grammar needs to be enlarged for the restricted arguments to the function symbols $\sigma_{\mathtt{x}}$, and the minimization needs to be restricted in order to guarantee the presence of certain productions.

In order to facilitate the definition of normal forms required for slp grammars, the existing work in [EH14] was recast using the equational logic of term algebras. This

formulation gives a transparent proof of the reducibility of grammar minimization to Max-SAT, and relates covering grammars and normal forms to the anti-unification in [Plo70a; Rey70]. Many of the proofs follow a common pattern by ordering non-terminals by a strict subterm relation, this proof technique could conceivably be generalized to an induction principle for trat grammars.

While this reformulation made the exposition clearer, some details were lost: in [EH15] a class of restricted normal forms for productions of sip grammars was defined. While both restricted normal forms and the sip-normal forms in this work yield covering grammars that can be generated in polynomial time, their relation still merits further investigation.

We also compared the canonical solutions and schematic sips in the work on induction with the canonical substitutions and schematic extended Herbrand sequents in [Het+14b]. Using f-modified canonical solutions, we showed how these two are related; as a corollary we obtain a large class of solutions that are very similar to the canonical substitutions of [Het+14b].

In the end it proved difficult to find good examples that make full use of the theory, i.e. which have Π_1 loop invariants. While universally quantified induction formulas are commonplace in inductive proofs; in program verification, the loop invariants can make use of a much more expressive language containing the program variables, reducing the need for quantified invariants. In some cases, the natural loop invariant is universally quantified (e.g. over all indices of an array), but this quantification usually becomes unnecessary after Skolemizing the postcondition, which is required as our approach only handles quantifier-free postconditions.

From a practical point of view, the main restriction of the approach in this work is that we cannot verify nested loops. It might be possible to relax this restriction by considering more general paths through a control-flow graph than single loops; or maybe by reducing it to nested induction, and developing a theory of nested inductive proofs. In addition, we have also restricted the classes of formulas—but this poses only a minor problem since many verification problems only require quantifier-free formulas: pre- and post-conditions are required to be quantifier-free, the background theory Π_1, and the loop invariant Π_1 with a single quantifier. Pre- and postconditions could be trivially generalized to Π_1 pre- and Σ_1 post-conditions, respectively. Allowing Π_1 invariants with an arbitrary number of quantifiers could be done as in [Het+14a], which handled the corresponding result for Π_1 cut-formulas with an arbitrary number of quantifiers. Relaxing the restriction on loop invariants to Π_2 or Σ_2 invariants is much harder and requires corresponding results for proofs with Π_2 or Σ_2 cut formulas, which are part of ongoing research.

It is not yet clear how well the approach for loop verification presented here will perform in practice. Automatically extracted Herbrand sequents from pathological instance proofs might not be easily generalized to a slp grammar; we are considering to work with grammars modulo an equational theory in this case. The algorithm we are proposing to find loop invariants still awaits implementation in `gapt`[1] and empirical testing as well.

[1] `https://logic.at/gapt`

Bibliography

[ABO10] Krzysztof R. Apt, Frank S. de Boer, and Ernst-Rüdiger Olderog. *Verification of Sequential and Concurrent Programs*. Texts in Computer Science. Springer, 2010.

[Ack62] Wilhelm Ackermann. *Solvable cases of the decision problem*. North-Holland, 1962.

[Arg+08] Josep Argelich, Chu Min Li, Felip Manya, and Jordi Planes. "The First and Second Max-SAT Evaluations." In: *Journal on Satisfiability, Boolean Modeling and Computation* 4.2-4 (2008), pp. 251–278.

[BN98] Franz Baader and Tobias Nipkow. *Term Rewriting and All That*. Cambridge University Press, 1998.

[Com+07] H. Comon, M. Dauchet, R. Gilleron, C. Löding, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. *Tree Automata Techniques and Applications*. 2007. URL: `http://www.grappa.univ-lille3.fr/tata`.

[EH14] Sebastian Eberhard and Stefan Hetzl. "Decomposing languages by rigid acyclic grammars". preprint. 2014. URL: `http://www.logic.at/staff/hetzl/research/zerlegung.pdf`.

[EH15] Sebastian Eberhard and Stefan Hetzl. "Inductive theorem proving based on tree grammars". In: *Annals of Pure and Applied Logic* (2015). URL: `http://dx.doi.org/10.1016/j.apal.2015.01.002`.

[Flo67] Robert W. Floyd. "Assigning meanings to programs". In: *Proceedings of Symposia in Applied Mathematics*. Vol. 19. AMS, 1967, pp. 19–32.

[Gen35] Gerhard Gentzen. "Untersuchungen über das logische Schließen. I". In: *Mathematische Zeitschrift* 39.1 (1935), pp. 176–210.

[Her30] Jacques Herbrand. "Recherches sur la théorie de la démonstration". PhD thesis. 1930.

[Het+14a] Stefan Hetzl, Alexander Leitsch, Giselle Reis, Janos Tapolczai, and Daniel Weller. "Introducing Quantified Cuts in Logic with Equality". In: *International Joint Conference on Automated Reasoning (IJCAR)*. Ed. by Stéphane Demri, Deepak Kapur, and Christoph Weidenbach. Vol. 8562. Lecture Notes in Computer Science. Springer, 2014, pp. 240–254.

[Het+14b] Stefan Hetzl, Alexander Leitsch, Giselle Reis, and Daniel Weller. "Algorithmic introduction of quantified cuts". In: *Theoretical Computer Science* 549 (2014), pp. 1–16.

[Het12] Stefan Hetzl. “Applying Tree Languages in Proof Theory”. In: *Language and Automata Theory and Applications (LATA)*. Ed. by Adrian-Horia Dediu and Carlos Martín-Vide. Vol. 7183. Lecture Notes in Computer Science. Springer, 2012, pp. 301–312.

[HKV11] Krytsof Hoder, Laura Kovács, and Andrei Voronkov. “Case Studies on Invariant Generation Using a Saturation Theorem Prover”. In: *Proceedings of the Mexican International Conference on Artificial Intelligence (MICAI)*. Vol. 7094. Lecture Notes in Artificial Intelligence. 2011, pp. 1–15.

[Hoa69] Charles Antony Richard Hoare. “An axiomatic basis for computer programming”. In: *Communications of the ACM* 12.10 (1969), pp. 576–580.

[JKV11] Florent Jacquemard, Francis Klay, and Camille Vacher. “Rigid Tree Automata and Applications”. In: *Information and Computation* 209.3 (2011), pp. 486–512.

[Mah88] Michael J. Maher. “Complete axiomatizations of the algebras of finite, rational and infinite trees”. In: *Proceedings of the Third Annual Symposium on Logic in Computer Science (LICS)*. IEEE. 1988, pp. 348–357.

[Opp78] Derek C. Oppen. “Reasoning about Recursively Defined Data Structures”. In: *Conference Record of the Fifth Annual ACM Symposium on Principles of Programming Languages (POPL)*. Ed. by Alfred V. Aho, Stephen N. Zilles, and Thomas G. Szymanski. 1978, pp. 151–157.

[Plo70a] Gordon D. Plotkin. “A note on inductive generalization”. In: *Machine Intelligence* 5.1 (1970), pp. 153–163.

[Plo70b] Gordon D. Plotkin. *Lattice theoretic properties of subsumption*. Edinburgh University, Department of Machine Intelligence and Perception, 1970.

[Plo71] Gordon D. Plotkin. “A further note on inductive generalization”. In: *Machine Intelligence* 6 (1971), pp. 101–124.

[Rey70] John C. Reynolds. “Transformational systems and the algebraic structure of atomic formulas”. In: *Machine Intelligence* 5.1 (1970), pp. 135–151.

[TS00] Anne S. Troelstra and Helmut Schwichtenberg. *Basic Proof Theory*. Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 2000.

[Tse83] Grigori S. Tseitin. “On the complexity of derivation in propositional calculus”. In: *Automation of Reasoning: Classical Papers in Computational Logic*. Vol. 2. Springer, 1983, pp. 466–483.