Saarland University
Faculty of Natural Sciences and Technology I
Department of Computer Science

Master's Thesis

Unification Modulo Nonnested Recursion Schemes via Anchored Semi-Unification

submitted by

Tobias Tebbi

on 2013-05-14

Supervisor, Advisor

Prof. Dr. Gert Smolka

Reviewers

Prof. Dr. Gert Smolka

Prof. Dr. Christoph Weidenbach

Abstract

A recursion scheme is an orthogonal rewriting system with rules of the form $f(x_1, \ldots, x_n) \to s$. We consider terms to be equivalent if they rewrite to the same redex-free possibly infinite term after infinitary rewriting. For the restriction to the nonnested case, where nested redexes are forbidden, we prove the existence of principal unifiers modulo scheme equivalence. We give an algorithm computing principal unifiers by reducing the problem to a novel fragment of semi-unification we call anchored semi-unification. For anchored semi-unification, we develop a decision algorithm that returns a principal semi-unifier in the positive case and has a time complexity of $\mathcal{O}(n^3\alpha(n))$ where $\alpha(n)$ is the inverse of the Ackermann function.

Acknowledgement

I would like to express my deep gratitude to my advisor Gert Smolka. His encouragement and advice in countless discussions helped me profoundly to write this thesis. Gert taught me how to turn complex ideas into simple parts. Furthermore, I could profit a lot from his profound knowledge. For example, without him, I would not have known that what I was working on is called recursion schemes and semi-unification.

I would like to thank my colleagues at the Programming Systems Lab, Chad E. Brown, Christian Doczkal, Ute Hornung, Jonas Kaiser, Steven Schäfer and Sigurd Schneider, for the great atmosphere and interesting discussions.

I especially want to thank Steven for the many fruitful discussions. It was his idea that tree equivalence might have something to do with unification.

Also, I would like to thank my friends and family for their constant support, motivation and a thousand other things there are no words for.

Finally, I would like to thank Gert and Christoph Weidenbach for reviewing this thesis.

Contents

1 Introduction

Recursion schemes (in the following shortly called schemes) describe mutually recursive function definitions and go back to the 1970s [19, 2]. They can be understood as rewriting systems with rules of the form $f(x_1, \dots, x_n) \to s$ such that there is exactly one rule per function symbol f and s is not a redex. Starting with a finite term, the limit of repeatedly rewriting with these rules is a possibly infinite term containing no redexes, see Dershowitz et al. [3].

We consider two terms to be equivalent if they rewrite to the same redex-free term. This equivalence is known as tree equivalence (introduced by Rosen [19]). Given a scheme $\mathcal{S}$, we will speak of $\mathcal{S}$-equivalence in the following.

As shown by Courcelle [2], $\mathcal{S}$-equivalence is interreducible to the equivalence of deterministic pushdown automata (DPDA). DPDA equivalence was an open problem for 20 years and was finally shown to be decidable by Sénizergues [22]. Sénizergues' proof yields a non-elementary decision procedure. The best known upper bound for the complexity of the problem is primitive recursive (see Stirling [24]).

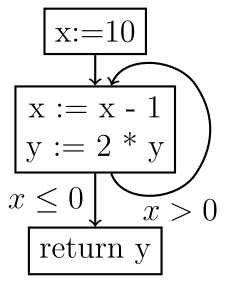
In this thesis, we will consider schemes without nested redexes, which we call nonnested schemes following Courcelle [2]. Sabelfeld [20] gives a polynomial decision procedure for $\mathcal{S}$-equivalence where $\mathcal{S}$ is a nonnested scheme.

Our motivation to investigate nonnested schemes is compiler verification. Nonnested schemes suffice to encode control flow graphs (CFGs) where everything but register updates and jumps is left uninterpreted. See Figure 1.1 for an example of a CFG together with a corresponding scheme. If two CFGs result in equivalent recursion schemes, then they are observationally equivalent. So if a compiler optimization produces a transformed CFG whose recursion scheme is equivalent to the recursion scheme of the original CFG, then this optimization is sound. Thus in a verified compiler, a compilation phase that produces a scheme-equivalent CFG can be validated by running a recursion scheme equivalence checker. We expect that compilation phases like global value numbering, code motion for assignments and register allocation (except for spilling) can be validated with this method.

We solve a more general problem than $\mathcal{S}$-equivalence, namely unification modulo $\mathcal{S}$-equivalence. This relies on our result that $\mathcal{S}$-unification is unitary (i.e., principal $\mathcal{S}$-unifiers exist). Given two terms s and t, we show how to compute a principal $\mathcal{S}$-unifier of s and t if one exists. For our results, we restrict substitutions such that variables are replaced with redex-free terms.

Our method is a certifying algorithm [16] in the following sense. Whenever we compute a principal substitution σ such that σs and σt are $\mathcal{S}$-equivalent, we also have a certificate proving that σs and σt are $\mathcal{S}$-equivalent.

Our method works by reducing the unification problem to a new decidable



$$\begin{array}{lcl} f_1(x,y) & \to & f_2(10,y) \\ f_2(x,y) & \to & \text{if}(x-1>0, f_2(x-1, 2*y), f_3(x-1, 2*y)) \\ f_3(x,y) & \to & \text{return}(y) \end{array}$$

Figure 1.1: CFG with corresponding scheme

fragment of semi-unification we call *anchored semi-unification*. Semi-unification is a generalization of ordinary syntactic unification. Given a set of inequalities $s_1 \dot{\preceq} t_1, \ldots, s_n \dot{\preceq} t_n$, a solution consists of substitutions $\sigma, \tau_1, \ldots, \tau_n$ such that $\tau_i(\sigma s_i) = \sigma t_i$ for $i \in \{1, \ldots, n\}$. Semi-unification was first identified by Lankford and Musser [11] in 1978 and rediscovered in different areas ten years later [6, 8, 18]. In its general form, semi-unification was shown to be undecidable by Kfoury et al. [9]. Several decidable fragments of semi-unification are known, but all of them differ significantly from our new fragment (see Chapter 5).

We present two decision-procedures for anchored semi-unification. First, we give a naive algorithm based on equational rules, which allows for a simple proof of correctness. These rules can be seen as a restriction of the diverging semi-unification rules of Leiß [13]. However, the naive algorithm has exponential run-time. Second, we present an efficient algorithm that is based on the unification-closure method [1]. This algorithm has almost cubic complexity, that is, $\mathcal{O}(n^3\alpha(n))$ where $\alpha(n)$ is the inverse of the Ackermann function.

This thesis is based on a publication at RTA 2013 [23].

Overview

In **Chapter 2**, we define unification modulo nonnested recursion schemes and show how to reduce this problem to the semi-unification problem.

We employ a direct coinductive definition of $\mathcal{S}$-equivalence for nonnested schemes and seem to be the first to do so. According to our definition, two terms s and t are $\mathcal{S}$-equivalent (i.e., $s \equiv t$) if there is a compliant relation relating s and t. Compared with process equivalence, compliant relations play the role of bisimulations.

Our first main result is the existence of principal $\mathcal{S}$-unifiers. A substitution σ is a principal $\mathcal{S}$-unifier of two terms s and t if it is a principal (aka most general) substitution such that $\sigma s \equiv \sigma t$. We aim at solving the $\mathcal{S}$-unification problem (ScUP), asking for a principal $\mathcal{S}$-unifier of two terms $f\bar{s}$ and $g\bar{t}$.

We solve ScUP by reducing it to the anchored semi-unification problem (AnSUP). This is done using three reduction steps.

$$\text{ScUP} \to \text{FIP} \to \text{SUP}^* \to \text{AnSUP}$$

From the existence of principal $\mathcal{S}$-unifiers, it follows that there are principal pairs. A pair of terms $(f\overline{s}, g\overline{t})$ is a principal pair for the function symbols f and g if for all tuples of terms $\overline{u}$ and $\overline{v}$, we have $f\overline{u} \equiv g\overline{v}$ iff $(f\overline{u}, g\overline{v})$ is a substitution instance of $(f\overline{s}, g\overline{t})$.

To find an $\mathcal{S}$-unifier of $f\overline{s}$ and $g\overline{t}$, it suffices to determine a principal pair for f and g. If there is no principal pair for f and g, then $f\overline{s}$ and $g\overline{t}$ are not $\mathcal{S}$-unifiable.

In analogy to up-to techniques for bisimulations, we weaken the conditions on compliant relations and obtain a decidable criterion for finite relations that implies $\mathcal{S}$-equivalence for the pairs in the relation (and their substitution instances). We call finite relations that satisfy this criterion certificates. It turns out that every finite set of principal pairs can be extended to a certificate by adding more principal pairs. Thus there are certificates for all $\mathcal{S}$-equivalences between redexes. On the other hand, a principal certificate contains only principal pairs. A principal certificate is a relation σF where F is a frame and σ is a principal substitution such that σF is a certificate. A frame is roughly a relation on terms of the form $f\overline{x}$ that does not contain a variable twice. We will show that if there is a principal pair for f and g, then it is possible to compute a frame F such that there is a substitution σ with σF being a principal certificate containing a principal pair for f and g. Finding such a substitution given a frame is the frame instantiation problem (FIP).

We reduce FIP to the standard semi-unification problem SUP. Since SUP is undecidable in general, we need a further reduction to the anchored semi-unification problem AnSUP.

In **Chapter 3**, we define AnSUP and show that it is decidable using a naive algorithm. Furthermore, we show that every instance of SUP obtained by our reduction from FIP translates to an instance of AnSUP. In the diagram above, SUP* indicates the fragment of SUP reachable by our reduction from FIP.

While we use inequalities $s \mathrel{\dot{\preceq}} t$ to define SUP, our definition of AnSUP employs equations $s \doteq t$ where s and t can contain instance variables αx consisting of a simple variable x and a substitution variable α that represents a substitution. The anchoredness constraint ensures that for every relevant instance variable αx, there is an equation $\alpha x \doteq s$ where s contains no instance variables.

We present a naive decision procedure that employs terminating semi-unification rules that are a restriction of the rules in [13]. The anchoredness constraint is preserved by applications of the semi-unification rules and allows us to always eliminate instance variables. However, the naive algorithm has exponential complexity.

In **Chapter 4**, we give a fast algorithm for anchored semi-unification and show that its complexity is almost cubic. The algorithm is based on an alternative definition of the semi-unification problem that makes it easier to adapt the unification-closure method to our problem.

2 Nonnested Linear Recursion Schemes

In this chapter, we define the unification problem for nonnested recursion schemes and show how to reduce it to the semi-unification problem.

2.1 Equivalence Modulo Nonnested Schemes

We assume an alphabet of **constants** (ranged over by a, b, c), an alphabet of **function symbols** (ranged over by f, g, h) and an alphabet of **variables** (ranged over by x, y, z). We assume that there are infinitely many variables and finitely many function symbols. We define **terms** (ranged over by s, t, u, v) using the grammar

$$s, t ::= a \mid x \mid s \cdot t \mid f\overline{s}$$

where $\overline{s}$ is a tuple of terms not containing a term of the form $f\overline{t}$.

Note that although we restrict ourselves to a single binary operator $(\cdot)$, we do not loose expressive power compared to full first-order terms since they can be encoded, for example with $a(s_1, s_2, \dots, s_n) \rightsquigarrow (\cdots((a \cdot s_1) \cdot s_2) \cdot \cdots \cdot s_n)$. We impose the usual discipline that every function symbol has a **fixed arity** and that for $f\overline{s}$, the length of $\overline{s}$ is the arity of f. We omit parentheses such that $s \cdot t \cdot u = s \cdot (t \cdot u)$.

Since we will have a rewrite rule for every function symbol, we call every term of the form $f\overline{s}$ a **redex**. A term is **simple** if it contains no redexes. A term is **plain** if it is simple or a redex.

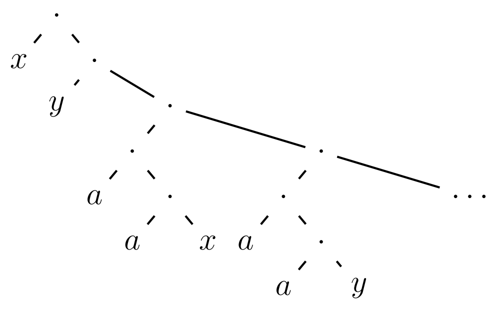
A **substitution** (ranged over by σ, τ) is a function from variables to simple terms. Note that this is a nonstandard restriction and affects our results for unification modulo nonnested schemes. We lift substitutions to range over terms, tuples and sets in the usual way.

A **declaration** of f is a rewrite rule $f\overline{x} \to s_1 \cdot s_2$ with distinct variables $\overline{x}$ and terms s_1, s_2 containing only variables from $\overline{x}$. A **nonnested scheme** (in the following shortly **scheme**) $\mathcal{S}$ is a set of declarations that contains exactly one declaration for every function symbol. We assume that a scheme $\mathcal{S}$ is given. For technical reasons, we require that $\mathcal{S}$ is **reduced**, that is, for every declaration $f\overline{x} \to s_1 \cdot s_2$, both s_1 and s_2 are plain and at least one of them is a redex. Sabelfeld [20] uses a similar restriction of the same name. Given a redex $f\overline{s}$, its **unfolding** $\mathcal{S}(f\overline{s})$ is the term σt where $f\overline{x} \to t$ is the unique declaration of f in $\mathcal{S}$ and σ is a substitution with $\sigma\overline{x} = \overline{s}$. We write $\mathcal{S}_i s$ for t_i where $\mathcal{S}s = t_1 \cdot t_2$ and $i \in \{1, 2\}$.

Example 1 For the reduced scheme

$$\begin{aligned} f(x,y) &\to x \cdot g(a \cdot x, y) \\ g(x,y) &\to y \cdot f(a \cdot x, a \cdot a \cdot y) \end{aligned}$$

infinitary rewriting of the redex $f(x,y)$ results in the infinite tree



□

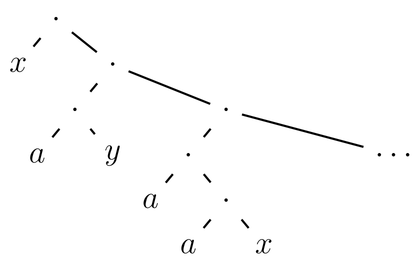
General Convention. From now on until Section 2.5, s, t, u and v will always denote plain terms and R will always denote a binary relation on plain terms.

We now give a coinductive definition of $\mathcal{S}$-equivalence. We call a relation R on plain terms **closed** if $R(\mathcal{S}_i s)(\mathcal{S}_i t)$ for all pairs of redexes $(s,t) \in R$ and $i \in \{1,2\}$. The **kernel** of a relation R is $\mathcal{K}R := \{(s,t) \in R \mid s \text{ or } t \text{ simple}\}$. A relation R is **coreflexive** if $s = t$ whenever Rst. We call a relation R **compliant** if it is closed and $\mathcal{K}R$ is coreflexive. Two plain terms s and t are **$\mathcal{S}$-equivalent** iff there is a compliant relation R with Rst. Since $\mathcal{S}$ is fixed, we simply write $s \equiv t$ to say that s and t are $\mathcal{S}$-equivalent and use $(\equiv)$ to denote the set of all $\mathcal{S}$-equivalent pairs. Note that a compliant relation is essentially a bisimulation between the trees generated by infinitary rewriting.

Example 2 For the scheme from Example 1,

$$\begin{aligned} \{&(f(x, a\cdot y), g(y,x)), \\ &(x,x),\ (g(a\cdot x, a\cdot y), f(a\cdot y, a\cdot a\cdot x)), \\ &(a\cdot y, a\cdot y),\ (f(a\cdot a\cdot x, a\cdot a\cdot a\cdot y), g(a\cdot a\cdot y, a\cdot a\cdot x)), \ldots\} \end{aligned}$$

is a compliant relation containing $(f(x, a\cdot y), g(y,x))$. Observe how this relation corresponds to a walk through the following tree, which can be obtained by infinitary rewriting of either $f(x, a \cdot y)$ or $g(y, x)$.



□

Proposition 3 *$\mathcal{S}$-equivalence ($\equiv$) is an equivalence relation on plain terms. Moreover, it is the largest compliant relation.*

Proof Reflexivity follows from the fact that $\{(s, s) \mid s \text{ plain}\}$ is compliant. The other properties follow from the fact that the inverse, union, intersection and composition of compliant relations are compliant. ■

Our coinductive definition of $\mathcal{S}$-equivalence is equivalent to the usual approach using infinite trees. For every term s, infinitary rewriting with the rules of the scheme yields a possibly infinite redex-free term that can be seen as a possibly infinite binary tree $\mathcal{T}s$ whose internal nodes are labeled with $\cdot$ and whose leaves are labeled with constants and variables. Two terms s and t are tree-equivalent if $\mathcal{T}s = \mathcal{T}t$. We write $(\mathcal{T}s) \cdot (\mathcal{T}t)$ for the tree whose root has exactly $\mathcal{T}s$ and $\mathcal{T}t$ as children. For a formal definition of tree equivalence, see e.g. Courcelle [2]. We will not define $\mathcal{T}s$, but instead we use the following two properties.

1. $\mathcal{T}(s \cdot t) = (\mathcal{T}s) \cdot (\mathcal{T}t)$
2. $\mathcal{T}s = (\mathcal{T}(\mathcal{S}_1 s)) \cdot (\mathcal{T}(\mathcal{S}_2 s))$ if s is a redex

In the following, we write $(\sim_\mathcal{T})$ for the relation on plain terms such that $s \sim_\mathcal{T} t$ iff $\mathcal{T}s = \mathcal{T}t$. Using property (1.), it is clear that for simple terms s and t, we have $s \sim_\mathcal{T} t$ iff $s = t$. For a simple term s, $\mathcal{T}s$ is finite. In contrast to this, $\mathcal{T}s$ is infinite if s is a redex because $\mathcal{S}$ is reduced. Taking these properties together, we obtain that $\mathcal{K}(\sim_\mathcal{T})$ is coreflexive. We also have that $(\sim_\mathcal{T})$ is closed because if $s \sim_\mathcal{T} t$ for redexes s and t, then, by property (2.), $(\mathcal{T}(\mathcal{S}_1 s)) \cdot (\mathcal{T}(\mathcal{S}_2 s)) = (\mathcal{T}(\mathcal{S}_1 t)) \cdot (\mathcal{T}(\mathcal{S}_2 t))$ and hence $\mathcal{S}_i s \sim_\mathcal{T} \mathcal{S}_i t$ for $i \in \{1, 2\}$. Thus $(\sim_\mathcal{T})$ is a compliant relation and therefore $(\sim_\mathcal{T}) \subseteq (\equiv)$.

It remains to show that $(\sim_\mathcal{T}) \supseteq (\equiv)$. Assume, for contradiction, that there are plain terms s and t with $s \equiv t$ that are not tree-equivalent. When two trees are different, then they differ at some finite level. Select s and t such that the level l at which $\mathcal{T}s$ and $\mathcal{T}t$ differ is minimal. Since $s \equiv t$, we have that s and t are both redexes because otherwise $s = t$. Since $\mathcal{T}s = (\mathcal{T}(\mathcal{S}_1 s)) \cdot (\mathcal{T}(\mathcal{S}_2 s))$ and $\mathcal{T}t = (\mathcal{T}(\mathcal{S}_1 t)) \cdot (\mathcal{T}(\mathcal{S}_2 t))$ differ at level l, there is $i \in \{1, 2\}$ such that $\mathcal{T}(\mathcal{S}_i s)$ and $\mathcal{T}(\mathcal{S}_i t)$ differ at level $l - 1$. So we have a contradiction because $\mathcal{S}_i s \equiv \mathcal{S}_i t$.

Remark 4 We can restrict ourselves to plain terms because other terms can be transformed into plain terms by adding additional declarations to the scheme.

Also, the restriction to reduced schemes is inessential because every scheme can be transformed into an equivalent reduced one.

- A declaration $f\overline{x} \to s$ with s simple can be eliminated by replacing every redex $f\overline{t}$ in $\mathcal{S}$ with the simple term $\mathcal{S}(f\overline{t})$.
- A declaration with deep redexes can be split into several declarations, e.g.

$$f(x) \to a \cdot g() \cdot x \qquad \rightsquigarrow \qquad \begin{array}{l} f(x) \to a \cdot f'(x) \\ f'(x) \to g() \cdot x \end{array}$$

□

2.2 Scheme Unification Problem (ScUP)

A substitution σ is an **$\mathcal{S}$-unifier** of two terms s and t if $\sigma s \equiv \sigma t$. We define the usual **instantiation pre-order** ($\preceq$) on tuples of terms and substitutions. For two tuples of terms $\overline{s}$ and $\overline{t}$, we have $\overline{s} \preceq \overline{t}$ if $\sigma\overline{s} = \overline{t}$ for some substitution σ. For two substitutions σ and τ, we have $\sigma \preceq \tau$ if $\tau = \tau' \circ \sigma$ for some substitution τ' (as usual, $(\tau' \circ \sigma)(x) := \tau'(\sigma x)$). We call a set Σ of substitutions **quasi-principal** if Σ is either empty or contains a substitution σ such that $\Sigma = \{\tau \mid \sigma \preceq \tau\}$. In this case we call σ a **principal element** of Σ. A **principal $\mathcal{S}$-unifier** of s and t is a principal element of $\{\sigma \mid \sigma s \equiv \sigma t\}$.

Problem 1 (ScUP) Given two plain terms s and t, find a principal $\mathcal{S}$-unifier of s and t if one exists. □

In the remainder of this section, we prove that two terms have a principal $\mathcal{S}$-unifier if they are $\mathcal{S}$-unifiable. The principal $\mathcal{S}$-unifiers will turn out to be essential for our reduction, because they allow us to characterize all $\mathcal{S}$-equivalences between terms with the finite set of principal $\mathcal{S}$-unifiers between terms of the form $f\overline{x}$.

The **closure** $\mathcal{C}(s,t)$ of s and t is the smallest closed relation containing (s,t).

Proposition 5 *$s \equiv t$ iff $\mathcal{K}(\mathcal{C}(s,t))$ is coreflexive.*

Proof If $s \equiv t$, then there is a compliant relation R containing (s,t). Thus $\mathcal{C}(s,t) \subseteq R$. Hence $\mathcal{K}(\mathcal{C}(s,t)) \subseteq \mathcal{K}R$ is coreflexive.

If $\mathcal{K}(\mathcal{C}(s,t))$ is coreflexive, then $\mathcal{C}(s,t)$ is a compliant relation containing (s,t). Thus $s \equiv t$. ■

Proposition 6 *For every redex s, we have $\mathcal{S}(\sigma s) = \sigma(\mathcal{S}s)$.*

This means that the closure is invariant under substitutions, as can be seen with the following lemma.

Lemma 7 $\mathcal{C}(\sigma s, \sigma t) = \sigma\,\mathcal{C}(s,t)$

Proof
Using Proposition 6, we obtain that $\sigma\,\mathcal{C}(s,t)$ is closed. Since also $(\sigma\,\mathcal{C}(s,t))(\sigma s)(\sigma t)$, we have $\mathcal{C}(\sigma s,\sigma t) \subseteq \sigma\,\mathcal{C}(s,t)$. It remains to show that $\mathcal{C}(\sigma s,\sigma t) \supseteq \sigma\,\mathcal{C}(s,t)$. This follows from a simple induction on the derivations of the elements in $\mathcal{C}(s,t)$. ■

The following well-known result generalizes the existence of principal unifiers for ordinary unification to infinite systems of equations. Note that σR is coreflexive iff σ is an (ordinary) unifier of R.

Proposition 8 *For a relation R on terms with finitely many variables, $\{\sigma \mid \sigma R$ coreflexive$\}$ is quasi-principal.*

Proof See Proposition 4.10 in Eder [4]. ■

Theorem 9 *$\{\sigma \mid \sigma s \equiv \sigma t\}$ is quasi-principal.*

Proof We have

$$\begin{aligned}
&\{\sigma \mid \sigma s \equiv \sigma t\} \\
&= \{\sigma \mid \mathcal{K}(\mathcal{C}(\sigma s,\sigma t)) \text{ coreflexive}\} && \text{by Proposition 5} \\
&= \{\sigma \mid \mathcal{K}(\sigma\,\mathcal{C}(s,t)) \text{ coreflexive}\} && \text{by Lemma 7} \\
&= \{\sigma \mid \sigma(\mathcal{K}(\mathcal{C}(s,t))) \text{ coreflexive}\}
\end{aligned}$$

and $\{\sigma \mid \sigma(\mathcal{K}(\mathcal{C}(s,t)))$ coreflexive$\}$ is quasi-principal by Proposition 8. ■

For a relation R on plain terms, we define

$$\uparrow R := \{(s',t') \mid \exists (s,t) \in R.\ (s,t) \preceq (s',t')\}$$

Given function symbols f and g, we call $(f\overline{s}, g\overline{t})$ a **principal pair** for (f,g) if $\uparrow\{(f\overline{s}, g\overline{t})\} = \{(f\overline{u}, g\overline{v}) \mid f\overline{u} \equiv g\overline{v}\}$. Note that a principal pair for f and g completely characterizes $\mathcal{S}$-equivalence for terms of the form $f\overline{s}$ and $g\overline{t}$.

Corollary 10 *If σ is a principal $\mathcal{S}$-unifier of $f\overline{x}$ and $g\overline{y}$, and $\overline{x},\overline{y}$ are pairwise distinct variables, then $\sigma(f\overline{x}, g\overline{y})$ is a principal pair.*

2.3 Frame Instantiation Problem (FIP)

In this section, we introduce the frame instantiation problem, which will allow us to compute principal pairs. In Section 2.4, we will then show how to construct a frame that can be instantiated to a certificate consisting of principal pairs only.

For a relation R on redexes, we define $\Uparrow R := \uparrow R \cup \{(s,s) \mid s \text{ simple}\}$. A finite relation R on redexes is a **certificate** if $\Uparrow R(\mathcal{S}_i s)(\mathcal{S}_i t)$ for all redexes $(s,t) \in R$ and $i \in \{1,2\}$. Note that a certificate is essentially a bisimulation up-to instantiation in the sense of up-to techniques for bisimulations [21].

Example 11 For the scheme from Example 1,

$$R := \{(f(x, a \cdot y), g(y, x)),\ (g(y, x), f(x, a \cdot y))\}$$

is a certificate. Observe that $\Uparrow R$ is a superset of the compliant relation from Example 2. Also note that R consists of two principal pairs. □

Lemma 12 *Let R be a certificate. Then $\Uparrow R \subseteq (\equiv)$.*

Proof It suffices to show that $\Uparrow R$ is compliant. We have that $\mathcal{K} \Uparrow R$ is coreflexive because $\mathcal{K} \Uparrow R = \Uparrow(\mathcal{K} R) = \Uparrow\{\}$. It remains to show that $\Uparrow R$ is closed. Consider a pair of redexes $(s, t) \in \Uparrow R$. We have to show that $\Uparrow R(\mathcal{S}_i s)(\mathcal{S}_i t)$ for $i \in \{1, 2\}$. By the definition of $\Uparrow R$, there are redexes $(u, v) \in R$ with $(u, v) \preceq (s, t)$. Since R is a certificate, we have $\Uparrow R(\mathcal{S}_i u)(\mathcal{S}_i v)$. As $(\mathcal{S}_i u, \mathcal{S}_i v) \preceq (\mathcal{S}_i s, \mathcal{S}_i t)$ by Proposition 6, it follows that $\Uparrow R(\mathcal{S}_i s)(\mathcal{S}_i t)$. ■

We write $s \approx t$ if s and t are redexes with the same function symbol, and $(s, t) \approx (u, v)$ if $s \approx u$ and $t \approx v$. We call a relation R on redexes **unitary** if there are no distinct pairs $(s_1, s_2), (t_1, t_2) \in R$ with $(s_1, s_2) \approx (t_1, t_2)$. Note that every unitary relation is finite because there are only finitely many function symbols. We call a unitary relation F on redexes of the form $f\overline{x}$ a **frame** if no variable occurs twice in F.

Example 13 Consider the frame

$$F := \{(f(x_1, x_2), g(x_3, x_4)),\ (g(x_5, x_6), f(x_7, x_8))\}$$

There is a substitution σ such that σF is the certificate from Example 11. Moreover, σ is principal with this property if it is the identity on all variables not occurring in F. □

Problem 2 (FIP) Given a frame F, find a principal element of

$$\{\sigma \mid \sigma F \text{ is a certificate}\}$$

if it exists. □

It is easy to decide if a finite relation is a certificate. Thus a certificate proves that its elements are $\mathcal{S}$-equivalent. So for every solution σ of a frame F, we obtain a certificate (in the sense of a certifying algorithm [16]) for the fact that all pairs of terms in σF are $\mathcal{S}$-equivalent.

2.4 ScUP to FIP

To solve ScUP, it suffices to compute principal pairs. Suppose we want to compute a principal $\mathcal{S}$-unifier of two plain terms s and t. We distinguish three cases.

1. If s and t are both redexes, then consider a principal pair (s', t') with $(s', t') \approx (s, t)$ and fresh variables. We obtain an ordinary unification problem because

$$\{\sigma \mid \sigma s \equiv \sigma t\} = \{\sigma \mid (s', t') \preceq \sigma(s, t)\} = \{\sigma \mid \exists\tau.\ \tau(s', t') = \sigma(s, t)\}$$

2. If s and t are both simple, then we are about to solve an ordinary unification problem since for simple terms, $\mathcal{S}$-equivalence is (syntactic) equality.
3. Otherwise, one of them is a redex and the other one is a simple term and hence they cannot be $\mathcal{S}$-unifiable.

So in the following, we only consider the first case and construct a frame F such that for its principal solution σ, a suitable principle pair is contained in σF.

We call a relation R on redexes **pointwise $\mathcal{S}$-unifiable** if s and t are $\mathcal{S}$-unifiable whenever Rst. We call a relation R on redexes **function-closed** if $\Uparrow R(\mathcal{S}_i s)(\mathcal{S}_i t)$ whenever Rst, $i \in \{1, 2\}$ and $\mathcal{S}_i s$ and $\mathcal{S}_i t$ are both redexes. For example, the frame from Example 13 is pointwise $\mathcal{S}$-unifiable and function-closed.

Lemma 14 *For every function-closed and pointwise $\mathcal{S}$-unifiable frame F, there is a substitution σ such that σF is a certificate containing only principal pairs.*

Proof For $(s, t) \in F$, let $\sigma_{s,t}$ be a principal $\mathcal{S}$-unifier of s and t. These $\mathcal{S}$-unifiers exist because F is pointwise $\mathcal{S}$-unifiable. Since the elements of F have disjoint variables, the following substitution is well-defined.

$$\tau x := \begin{cases} \sigma_{s,t} x & \text{if there is } (s,t) \in F \text{ s.t. } x \text{ occurs in } (s,t) \\ x & \text{otherwise} \end{cases}$$

For all $(s, t) \in F$, we have that $\sigma_{s,t}(s, t) = \tau(s, t)$ and hence $\tau(s, t)$ is a principal pair by Corollary 10. Thus τF consists only of principal pairs and $\Uparrow(\tau F) = (\equiv) \cap \Uparrow F$.

It remains to show that τF is a certificate. Consider a pair of redexes $(s, t) \in \tau F$. We have to show that $\Uparrow(\tau F)(\mathcal{S}_i s)(\mathcal{S}_i t)$ for $i \in \{1, 2\}$. There are $(s', t') \in F$ with $\tau(s', t') = (s, t)$. We have that $s \equiv t$ and hence $\mathcal{S}_i s \equiv \mathcal{S}_i t$. By Proposition 6, $\tau(\mathcal{S}_i s') = \mathcal{S}_i s \equiv \mathcal{S}_i t = \tau(\mathcal{S}_i t')$. Thus either both $\mathcal{S}_i s'$ and $\mathcal{S}_i t'$ are redexes or both are simple. Hence and because F is function-closed, we have $\Uparrow F(\mathcal{S}_i s')(\mathcal{S}_i t')$. Thus also $\Uparrow F(\tau(\mathcal{S}_i s'))(\tau(\mathcal{S}_i t'))$ and equivalently $\Uparrow F(\mathcal{S}_i s)(\mathcal{S}_i t)$. Since $\Uparrow(\tau F) = (\equiv) \cap \Uparrow F$, we conclude that $\Uparrow(\tau F)(\mathcal{S}_i s)(\mathcal{S}_i t)$. ∎

However, we are still missing a way to construct an appropriate function-closed and pointwise $\mathcal{S}$-unifiable frame. It turns out that we get pointwise $\mathcal{S}$-unifiability for free if we just make the frame as small as we can.

Given function symbols f and g, we write $\mathcal{F}(f, g)$ for a fixed function-closed frame with minimum cardinality containing a pair $(f\overline{x}, g\overline{y})$ for some pairwise distinct variables $\overline{x}, \overline{y}$. Since every maximum cardinality frame is function-closed, $\mathcal{F}(f, g)$ always exists. For example, we can take the frame from Example 13 for

$\mathcal{F}(f,g)$ where f and g are the function symbols from Example 1. One can easily compute $\mathcal{F}(f,g)$ by incrementally adding elements $(h\overline{x}, h'\overline{y})$ with fresh and distinct variables $\overline{x}, \overline{y}$ as required by the function-closedness constraint.

Lemma 15 *If $f\overline{s}$ and $g\overline{t}$ are $\mathcal{S}$-unifiable, then $\mathcal{F}(f,g)$ is pointwise $\mathcal{S}$-unifiable.*

Proof Suppose, for contradiction, that $F := \{(u,v) \in \mathcal{F}(f,g) \mid u,v\ \mathcal{S}\text{-unifiable}\} \neq \mathcal{F}(f,g)$. Then F has strictly smaller cardinality. We have $\Uparrow F(f\overline{s})(g\overline{t})$ since $f\overline{s}$ and $g\overline{t}$ are $\mathcal{S}$-unifiable. Thus we can conclude a contradiction by showing that F is also function-closed. Let $(u,v) \in F$ and $i \in \{1,2\}$ such that $\mathcal{S}_i u$ and $\mathcal{S}_i v$ are both redexes. We need to show that $\Uparrow F(\mathcal{S}_i u)(\mathcal{S}_i v)$. Since $\mathcal{F}(f,g)$ is function-closed, we have $(\Uparrow \mathcal{F}(f,g))(\mathcal{S}_i u)(\mathcal{S}_i v)$. Select a substitution σ with $\sigma u \equiv \sigma v$. By Proposition 6, we have $\sigma(\mathcal{S}_i u) = \mathcal{S}_i(\sigma u) \equiv \mathcal{S}_i(\sigma v) = \sigma(\mathcal{S}_i v)$. Thus $\mathcal{S}_i u$ and $\mathcal{S}_i v$ are $\mathcal{S}$-unifiable and hence $\Uparrow F(\mathcal{S}_i u)(\mathcal{S}_i v)$. ■

Corollary 16 *If $f\overline{s}$ and $g\overline{t}$ are $\mathcal{S}$-unifiable, then $\{\sigma \mid \sigma\,\mathcal{F}(f,g)$ is certificate$\}$ is nonempty and for every principal element σ, we have that $\sigma\,\mathcal{F}(f,g)$ contains a principal pair for (f,g).*

We will reduce FIP to AnSUP in a way that a frame F is mapped to an anchored system of equations E with $\{\sigma \mid \sigma F \text{ is certificate}\} = \{\sigma \mid \sigma \text{ semi-unifies } E\}$ and we will prove that $\{\sigma \mid \sigma \text{ semi-unifies } E\}$ is quasi-principal. Thus if $f\overline{s}$ and $g\overline{t}$ are $\mathcal{S}$-unifiable, then $\{\sigma \mid \sigma\,\mathcal{F}(f,g) \text{ is certificate}\}$ contains a principal element σ and $\sigma\,\mathcal{F}(f,g)$ contains a principal pair for (f,g). Otherwise $\{\sigma \mid \sigma\,\mathcal{F}(f,g) \text{ is certificate}\}$ is empty. This concludes the reduction from ScUP to FIP.

2.5 FIP to SUP

In this section, we define the semi-unification problem (SUP) using systems of inequalities and give a reduction from FIP to SUP.

Let D be a relation on tuples of plain terms. We write the elements $(\overline{s}, \overline{t}) \in D$ as **inequalities** $\overline{s} \mathrel{\dot{\preceq}} \overline{t}$ and call D a **system of inequalities**. We call D **directed** if $\overline{s} \preceq \overline{t}$ for all $(\overline{s} \mathrel{\dot{\preceq}} \overline{t}) \in D$. If σD is directed, then σ is a **semi-unifier** of D.

Problem 3 (SUP) Given a system of inequalities D, find a principal semi-unifier of D. □

In this section, we will show how to construct a system of inequalities with the same principal solution as a given instance of FIP.

Note that our definition of SUP corresponds roughly to the usual definition of the semi-unification problem (e.g. in Henglein [6]). Since semi-unification is undecidable, we cannot hope to solve the problem in general. Instead, we only consider the image of the reduction from FIP. In Section 3.2, we will show how to reduce this image to the anchored fragment.

For a unitary relation R, we define the **projection** of a pair (s, t) of plain terms as follows.

$$\pi_R(s,t) := \begin{cases} (s',t') & \text{if } Rs't' \text{ and } (s',t') \approx (s,t) \\ (s,s) & \text{if } s,t \text{ simple} \\ () & \text{otherwise} \end{cases}$$

Note that $\pi_R(s, t)$ is well-defined because R is unitary. The following proposition holds due to the fact that $(s, s) \preceq (s, t)$ iff $s = t$.

Proposition 17 *Let R be unitary. Then $\Uparrow Rst$ iff $\pi_R(s,t) \preceq (s,t)$.*

For a unitary relation R, we construct the system of inequalities

$$\mathcal{D}\,R := \{\pi_R(\mathcal{S}_i s, \mathcal{S}_i t) \mathrel{\dot{\preceq}} (\mathcal{S}_i s, \mathcal{S}_i t) \mid Rst, i \in \{1,2\}\}$$

Example 18 For the frame F from Example 13, we construct the system of inequalities

$$\begin{aligned} \mathcal{D}\,F = \{ \quad & (x_1, x_1) \mathrel{\dot{\preceq}} (x_1, x_4), \\ & (g(x_5,x_6), f(x_7,x_8)) \mathrel{\dot{\preceq}} (g(a \cdot x_1, x_2), f(a \cdot x_3, a \cdot a \cdot x_4)), \\ & (x_6, x_6) \mathrel{\dot{\preceq}} (x_6, x_7), \\ & (f(x_1,x_2), g(x_3,x_4)) \mathrel{\dot{\preceq}} (f(a \cdot x_5, a \cdot a \cdot x_6), g(a \cdot x_7, x_8)) \quad \} \end{aligned}$$ □

Proposition 19 *Let R be unitary. Then $\mathcal{D}\,R$ is directed iff R is a certificate.*

Proof Follows from the construction of $\mathcal{D}\,R$ using Proposition 17. ■

Proposition 20 *Let R be unitary. Then $\sigma(\pi_R(s,t)) = \pi_{(\sigma R)}(\sigma s, \sigma t)$ and $\sigma(\mathcal{D}\,R) = \mathcal{D}(\sigma R)$.*

Lemma 21 *Let R be unitary. Then σ semi-unifies $\mathcal{D}\,R$ iff σR is a certificate.*

Proof Follows immediately from Proposition 19 and Proposition 20. ■

Thus $\mathcal{D}$ is a reduction from FIP to SUP.

3 Anchored Semi-Unification

In this chapter, we define the anchored semi-unification problem (AnSUP), show that the reduction from FIP can be extended to AnSUP and give a naive algorithm proving that AnSUP is decidable.

3.1 Anchored Semi-Unification Problem (AnSUP)

The definition as well as the algorithm for AnSUP rely on a formulation of semi-unification that uses systems of equations between terms with explicit substitution variables instead of inequalities. A similar formulation has been used by Leiß [13]. For example, the inequalities $(x_1, x_2) \dot{\preceq} (x_3, x_4)$ and $(x_5) \dot{\preceq} (x_6)$ corresponds to the equations $\alpha x_1 \doteq x_3$, $\alpha x_2 \doteq x_4$ and $\beta x_5 \doteq x_6$ with the substitution variables α and β. The substitution variables make the additional substitution on the left-hand side of inequalities explicit.

We call the variables we have used so far **simple variables** and assume an additional alphabet of **substitution variables** (ranged over by α, β, γ). An **instance variable** αx is a pair of a substitution variable and a simple variable. We define a new kind of **terms** that extend the simple terms we defined before with instance variables.

$$s, t ::= a \mid x \mid s \cdot t \mid \alpha x$$

Given a simple term s and a substitution variable α, we write $\hat{\alpha} s$ for the term obtained from s by replacing every variable x by αx. As before, substitutions are functions from variables to simple terms. In the semi-unification context [13, 6, 8], this is the standard notion of substitution. We lift substitutions to terms according to the equations $\sigma(\alpha x) = \hat{\alpha}(\sigma x)$, $\sigma(s \cdot t) = \sigma s \cdot \sigma t$ and $\sigma a = a$.

We say a term s **occurs** in a term t if s is a subterm of t. We do not consider x to be a subterm of αx.

An **assignment** A is a function from substitution variables to substitutions. Assignments are lifted to terms such that As is the term obtained from s by replacing every occurrence of an instance variable αx with the simple term $(A\alpha)x$.

A **system of equations** E is a finite set of pairs of terms. We take the freedom to write $s \doteq t$ for a pair (s, t). A substitution σ is a **semi-unifier** of E if there is an assignment A such that for all $s \doteq t \in E$, we have $A(\sigma s) = A(\sigma t)$.

The formulation of semi-unification as just presented has the same expressivity as the formulation with inequalities. We will now restrict the systems of equations we consider to obtain the anchored fragment.

An **atom** (ranged over by X, Y, Z) is either a simple variable or an instance variable. A **partial equivalence relation** (**PER**) is a symmetric and transitive relation. Note that a PER ($\sim$) is an equivalence relation on $\{X \mid X \sim X\}$.

We call an equation $\alpha x \doteq s$ an **anchor** for αx if s is simple. The anchoredness condition ensures that there is an anchor for every instance variable αx that might occur when the semi-unification rules to be defined later are applied. A system of equations E is **anchored** with a PER ($\sim$) on atoms if the following conditions hold.

1. If $s \doteq t \in E$ with X and Y occurring in $s \doteq t$, then $X \sim Y$.
2. If $x \sim y$ and $\alpha x \sim \alpha x$, then $\alpha x \sim \alpha y$.
3. If $\alpha x \sim \alpha x$, then there is a simple term s with $\alpha x \doteq s \in E$ or $x \doteq s \in E$.

For example, $\{x \doteq z \cdot \beta y,\ \alpha x = a \cdot b\}$ is not anchored, but $\{x \doteq z \cdot \beta y,\ \alpha x = a \cdot b,$ $\beta y = z,\ \alpha z = z\}$ is anchored with the symmetric and transitive closure of $\{(x, z),$ $(z, \beta y), (\alpha x, \alpha z), (\alpha z, z)\}$.

Problem 4 (AnSUP) Given an anchored system of equations E, find a principal semi-unifier. □

For comparison to other fragments of semi-unification, we now give a definition of the anchoredness condition for systems of inequalities. A system of inequalities is anchored if there is a partition X of the variables such that for every inequality $\overline{s} \mathrel{\dot{\preceq}} \overline{t}$, the following two conditions hold.

1. All variables in $\overline{t}$ are in a single class of the partition X.
2. All variables in $\overline{s}$ are in a single class A of the partition X and every variable in A occurs at least once at a position in $\overline{s}$ that also exists in $\overline{t}$. Expressed formally, there is a tuple $\overline{u}$ and substitutions σ, τ such that $\sigma\overline{u} = \overline{s}$, $\tau\overline{u} = \overline{t}$ and for all variables $x \in A$, we have that $\sigma x = x$ and x occurs in u.

3.2 FIP to AnSUP

Let F be a frame. We will translate the system of inequalities $\mathcal{D}\,F$ into an equivalent anchored system of equations E such that σ semi-unifies E iff σ semi-unifies $\mathcal{D}\,F$.

The case where $\mathcal{D}\,F$ contains an element of the form $() \mathrel{\dot{\preceq}} (s, t)$ is trivial, since in this case, there are no substitutions that semi-unify $\mathcal{D}\,F$. So in the following, we assume that $\mathcal{D}\,F$ contains only inequalities between pairs.

We construct E by translating every element of $\mathcal{D}\,F$ into equations according to the rules

$$\begin{array}{ccc}
(f\overline{x}, g\overline{y}) \mathrel{\dot{\preceq}} (f\overline{s}, g\overline{t}) & \rightsquigarrow & \alpha\overline{x} \doteq \overline{s}, \alpha\overline{y} \doteq \overline{t} \\
(s, s) \mathrel{\dot{\preceq}} (s, t) & \rightsquigarrow & s \doteq t
\end{array}$$

where α is a fresh substitution variable for every inequality and $\alpha\overline{x} \doteq \overline{s}$ stands for $\alpha x_1 \doteq s_1, \dots, \alpha x_n \doteq s_n$ with $\overline{x} = (x_1, \dots, x_n)$ being a tuple of variables and $\overline{s} = (s_1, \dots, s_n)$ being a tuple of simple terms of the same length. Note that the rules cover all elements of $\mathcal{D}\,F$ and that in the second rule, s and t are always simple terms.

Example 22 The system of inequalities from Example 18 is translated into the system of equations

$$\begin{aligned} &x_1 \doteq x_4, \\ &\alpha x_5 \doteq a \cdot x_1,\ \alpha x_6 \doteq x_2, \qquad \alpha x_7 \doteq a \cdot x_3,\ \alpha x_8 \doteq a \cdot a \cdot x_4, \\ &x_6 \doteq x_7, \\ &\beta x_1 \doteq a \cdot x_5,\ \beta x_2 \doteq a \cdot a \cdot x_6,\ \beta x_3 \doteq a \cdot x_7,\ \beta x_4 \doteq x_8 \end{aligned}$$

□

Lemma 23 *Let E be a translation of $\mathcal{D}\,F$ for some frame F. Then σ semi-unifies $\mathcal{D}\,F$ iff σ semi-unifies E.*

Proof Since distinct inequalities in $\mathcal{D}\,F$ are translated into equations with disjoint substitution variables, it suffices to consider the translation E' of a singleton subset $D' \subseteq \mathcal{D}\,F$.

Let $D' = \{(f\overline{x}, g\overline{y}) \dot{\preceq} (f\overline{s}, g\overline{t})\}$. Then $E' = \{\alpha\overline{x} \doteq \overline{s}, \alpha\overline{y} \doteq \overline{t}\}$. If σ semi-unifies D', then there is a substitution τ with $\tau(\sigma(f\overline{x}, g\overline{y})) = \sigma(f\overline{s}, g\overline{t})$. Thus σ semi-unifies E' with the assignment A where $A\alpha := \tau$. If σ semi-unifies E', then σ semi-unifies D' because $(A\alpha)(\sigma\overline{x}) = A(\sigma(\alpha\overline{x})) = A(\sigma\overline{s}) = \sigma\overline{s}$ and the same for $\alpha\overline{y}$ and $\overline{t}$.

Let $D' = \{(s, s) \dot{\preceq} (s, t)\}$ with s and t being simple. Then $E' = \{s \doteq t\}$. If σ semi-unifies D', then there is a substitution τ with $\tau(\sigma s) = \sigma s$ and $\tau(\sigma s) = \sigma t$. Thus $\sigma s = \sigma t$ and hence σ semi-unifies E'. If σ semi-unifies E', then $\sigma s = \sigma t$ and hence σ semi-unifies D'. ■

We need the following technical lemma to show that the translation is anchored.

Lemma 24 *Let $(\sim)$ be an equivalence relation on simple variables and let E be a system of equations containing only equations of the form $\alpha x \doteq s$ or $s \doteq t$ where s and t are simple terms. Then E is anchored if the following conditions are satisfied.*

1. *If αx occurs in E and $x \sim y$, then αy occurs in E.*
2. *If $\alpha x_1 \doteq s \in E$ and $\alpha x_2 \doteq t \in E$, then $y_1 \sim y_2$ for all simple variables y_1, y_2 that occur in s or t.*
3. *If $s \doteq t \in E$, then $x \sim y$ for all simple variables x, y that occur in s or t.*

Proof Let $\hat{\sim}$ be the smallest symmetric relation on atoms such that

- If $x \sim y$, then $x \mathrel{\hat{\sim}} y$.
- If αx and αy occur in E, then $\alpha x \mathrel{\hat{\sim}} \alpha y$.

- If $\alpha x \doteq s \in E$ and y occurs in s, then $\alpha x \mathrel{\hat{\sim}} y$.

The transitive closure $\hat{\sim}^*$ of $\hat{\sim}$ is a PER. We will show that E is anchored with $\hat{\sim}^*$. But first, we prove that $x \sim y$ whenever $x \mathrel{\hat{\sim}^*} y$. We do this by induction on the length of the shortest sequence $x \mathrel{\hat{\sim}} X_1 \mathrel{\hat{\sim}} \cdots \mathrel{\hat{\sim}} X_n \mathrel{\hat{\sim}} y$. If $n = 0$, then the claim follows immediately from the definition of $\hat{\sim}$. For the inductive case, we distinguish two cases. If there is some simple variable X_i with $i \in \{1, \ldots, n\}$, then the inductive hypothesis for $x \mathrel{\hat{\sim}} X_1 \mathrel{\hat{\sim}} \cdots \mathrel{\hat{\sim}} X_i$ and $X_i \mathrel{\hat{\sim}} \cdots \mathrel{\hat{\sim}} X_n \mathrel{\hat{\sim}} y$ yield $x \sim X_i$ and $X_i \sim y$ and thus $x \sim y$. Otherwise, by the definition of $\hat{\sim}$, there are simple variables $z_1, \ldots, z_n$ and a single substitution variable α such that $X_i = \alpha z_i$ for all $i \in \{1, \ldots, n\}$. By the definition of $\hat{\sim}$, there are simple terms s, t such that $\alpha z_1 \doteq s \in E$ and $\alpha z_n \doteq t \in E$ where x occurs in s and y occurs in t. Thus $x \sim y$ by requirement (2) on $(\sim)$.

Now, we show that E is anchored with $\hat{\sim}^*$.

1. Let $s \doteq t \in E$ with X and Y occurring in $s \doteq t$. We need to show that $X \mathrel{\hat{\sim}^*} Y$. If s and t are simple, then this follows immediately from requirement (3) on $(\sim)$. Otherwise $s = \alpha x$ for some instance variable αx and the claim follows from the definition of $\hat{\sim}$.
2. Let $x \mathrel{\hat{\sim}^*} y$ and $\alpha x \mathrel{\hat{\sim}^*} \alpha x$. We need to show that $\alpha x \mathrel{\hat{\sim}^*} \alpha y$. Because of requirement (1) on $(\sim)$, it suffices to show that $x \sim y$. As we proved before, this follows from $x \mathrel{\hat{\sim}^*} y$.
3. Let $\alpha x \hat{\sim}^* \alpha x$. It suffices to show that there is a simple term s with $\alpha x \doteq s \in E$. This follows from the conditions on E as $\hat{\sim}^*$ only contains instance variables from E. ■

Lemma 25 *Let E be a translation of $\mathcal{D}\,F$ for some frame F. Then E is anchored.*

Proof Consider the equivalence relation $(\sim)$ on simple variables such that $x \sim y$ iff x and y occur in the same element of F or $x = y$. It is easy to check that $(\sim)$ and E satisfy the conditions of Lemma 24. ■

3.3 Solving AnSUP

In this section, we present naive rules to solve the anchored semi-unification problem. Our rules are a restriction of the rules presented by Leiß [13].

However, the rules presented in this section only yield an algorithm with exponential complexity. In Chapter 4, we will improve upon this by giving a second algorithm, which has polynomial complexity.

We first define the solved form computed by our algorithm. We write $E, s \doteq t$ for the disjoint union $E \mathbin{\dot\cup} \{s \doteq t\}$. Given a system of equations E, we call an atom X **eliminated** if E has the form $E', X \doteq s$ such that neither X nor (in case X is a simple variable) an instance variable αX occur in E' or in s. A system of equations E is in **solved form** if all equations have the form $X \doteq s$ where X is eliminated and s is simple.

Lemma 26 *If a system of equations E is in solved form, then it has a principal semi-unifier.*

Proof Since E is in solved form, we can unambiguously define a substitution σ and an assignment A as follows.

$$\sigma x = \begin{cases} s & \text{if } x \doteq s \in E \\ x & \text{else} \end{cases}$$
$$(A\alpha)x = \begin{cases} s & \text{if } \alpha x \doteq s \in E \\ x & \text{else} \end{cases}$$

For σ to be a semi-unifier of E, we have to show that for every equation $X \doteq s \in E$ we have $A(\sigma X) = A(\sigma s)$. Since E is in solved form, s is simple and cannot contain an eliminated simple variable. Thus $A(\sigma s) = As = s$. So it suffices to show that $A(\sigma X) = s$.

If $X = x$ for some simple variable x, then $\sigma x = s$ by the definition of σ. Since s is simple, it follows that $A(\sigma x) = A(s) = s$.

If $X = \alpha x$ for some instance variable αx, then $\sigma x = x$ because otherwise x would be eliminated and hence αx could not occur in E. So it follows that $A(\sigma(\alpha x)) = A(\alpha x) = s$.

It remains to show that σ is principal. Consider some other semi-unifier τ of E. Since τ unifies all equations of the form $x \doteq s \in E$, we have that $\tau x = \tau(\sigma x)$ for all simple variables x. Thus $\sigma \preceq \tau$. ∎

A system of equations E is **clashed** if one of the following conditions holds.

- $a \doteq s \in E$ or $s \doteq a \in E$ where $a \neq s$ and s is not an atom.
- $x \doteq s \in E$ where x or αx for some α occurs in s and s is not an atom.
- $\alpha x \doteq s \in E$ where αx occurs in $s \neq \alpha x$.

Proposition 27 *A clashed system of equations has no semi-unifiers.*

The rules in Figure 3.1 transform every anchored system of equations into an equivalent system of equations that is clashed or in solved form. To ensure termination, the rules must not be applied to a clashed system of equations. We write $s[t/x]$ for σs where σ is the unique substitution satisfying $\sigma x = t$ and $\sigma y = y$ for all y with $y \neq x$. We write $s[t/\alpha x]$ for the term that is obtained from s by replacing every occurrence of αx with t. We write $E[s/X]$ for $\{t[s/X] \doteq u[s/X] \mid t \doteq u \in E\}$.

$$\begin{array}{lrcl}
R_{\text{bop}} & E, s_1 \cdot s_2 \doteq t_1 \cdot t_2 & \Longrightarrow & E, s_1 \doteq t_1, s_2 \doteq t_2 \\
R_{\text{refl}} & E, s \doteq s & \Longrightarrow & E \\
R_{\text{orient1}} & E, s \doteq \alpha x & \Longrightarrow & E, \alpha x \doteq s \text{ if } s \text{ is not an instance variable} \\
R_{\text{orient2}} & E, s \doteq x & \Longrightarrow & E, x \doteq s \text{ if } s \text{ is not an atom} \\
R_{\text{elim1}} & E, \alpha x \doteq s & \Longrightarrow & E[s/\alpha x], \alpha x \doteq s \text{ if } s \text{ is simple} \\
R_{\text{elim2}} & E, x \doteq s & \Longrightarrow & E[s/x], x \doteq s \text{ if } x \text{ does not occur in } s \\
 & & & \qquad\qquad\text{and } s \text{ is simple}
\end{array}$$

Figure 3.1: Semi-Unification Rules

Example 28 We transform the following anchored system of equations into solved form.

$$\begin{array}{rr}
 & \alpha x_1 \doteq a \cdot x_3,\ \alpha x_2 \doteq a \cdot (a \cdot x_4),\ \alpha x_3 \doteq a \cdot x_1,\ \alpha x_4 \doteq x_2,\ x_1 \doteq x_4 \\
\overset{R_{\text{elim2}}}{\Longrightarrow} & \alpha x_4 \doteq a \cdot x_3,\ \alpha x_2 \doteq a \cdot (a \cdot x_4),\ \alpha x_3 \doteq a \cdot x_4,\ \alpha x_4 \doteq x_2,\ x_1 \doteq x_4 \\
\overset{R_{\text{elim1}}}{\Longrightarrow} & x_2 \doteq a \cdot x_3,\ \alpha x_2 \doteq a \cdot (a \cdot x_4),\ \alpha x_3 \doteq a \cdot x_4,\ \alpha x_4 \doteq x_2,\ x_1 \doteq x_4 \\
\overset{R_{\text{elim2}}}{\Longrightarrow} & x_2 \doteq a \cdot x_3,\ a \cdot \alpha x_3 \doteq a \cdot (a \cdot x_4),\ \alpha x_3 \doteq a \cdot x_4,\ \alpha x_4 \doteq a \cdot x_3,\ x_1 \doteq x_4 \\
\overset{R_{\text{bop}}}{\Longrightarrow} & x_2 \doteq a \cdot x_3,\ a \doteq a,\ \alpha x_3 \doteq a \cdot x_4,\ \alpha x_4 \doteq a \cdot x_3,\ x_1 \doteq x_4 \\
\overset{R_{\text{refl}}}{\Longrightarrow} & x_2 \doteq a \cdot x_3,\ \alpha x_3 \doteq a \cdot x_4,\ \alpha x_4 \doteq a \cdot x_3,\ x_1 \doteq x_4 \quad \square
\end{array}$$

Proposition 29 *If $E \Longrightarrow E'$, then E and E' have the same semi-unifiers.*

Proposition 30 *If $E, X \doteq s$ is anchored and s is simple, then $E[s/X], X \doteq s$ is anchored.*

Lemma 31 *If $E \Longrightarrow E'$ and E is anchored, then E' is anchored.*

Proof For R_{elim1} and R_{elim2}, this follows immediately from Proposition 30. For the other rules, the claim is trivial. ∎

Lemma 32 *There is no infinite reduction sequence $E_1 \Longrightarrow E_2 \Longrightarrow \cdots$ such that E_i is not clashed for all $i \in \{1, 2, \ldots\}$.*

Proof We assign a triple of natural numbers $(n_1, n_2, n_3 + n_4 + n_5)$ to every system of equations $E = \{s_1 \doteq t_1, \ldots, s_n \doteq t_n\}$ where

n_1 is the number of simple variables in E that are not eliminated,

n_2 is the number of instance variables in E that are not eliminated,

n_3 is the sum of the sizes of every term s_i that is not an instance variable for $i \in \{1, 2, \ldots\}$,

n_4 is the sum of the sizes of every term s_i that is not an atom for $i \in \{1, 2, \ldots\}$ and

n_5 is the number of equations in E.

With the usual lexicographical ordering, this yields a well-founded ordering on systems of equations. It is easy to show that every rule application respects the ordering and hence an infinite reduction sequence is impossible. ∎

We call a system of equations E **terminal** if there is no system of equations E' with $E \Longrightarrow E'$.

Proposition 33 *If E is anchored and terminal, then E is either clashed or in solved form.*

Proof Let E be anchored, terminal and not clashed. We show that E is in solved form, that is, all equations have the form $X \doteq s$ where X is eliminated and s is simple.

Let $s \doteq t \in E$. Then s is an atom because otherwise R_{bop}, R_{refl}, R_{orient1} or R_{orient2} would be applicable contradicting the fact that E is terminal. Also, t is simple because if it contained an instance variable αx, then E would also contain an equation $\alpha x \doteq u$ because E is anchored. This is a contradiction because R_{elim1} would be applicable. Since s is an atom and t is simple, s must be eliminated because otherwise R_{elim1} or R_{elim2} would be applicable. ∎

Thus we can transform every anchored system of equations E either into solved form or into a clashed system of equations. In the first case, we have computed a principal semi-unifier of E. In the second case, E has no semi-unifiers.

4 Solving AnSUP Efficiently

The naive rules for anchored semi-unification presented in Section 3.3 suffer from the same problems as the naive rules for ordinary unification (see [1]). The term size can grow exponentially in the number of steps. For example, the rules perform poorly on the ordinary unification problem $x_1 \doteq x_2 \cdot x_2,\ x_2 \doteq x_3 \cdot x_3, \dots,\ x_{n-1} \doteq x_n \cdot x_n$. The repeated substitutions yield a term of exponential size. Processing such exponentially large terms can take an exponential number of steps.

In this chapter, we will show how to solve AnSUP in almost cubic time.

4.1 Extended Substitutions

Our efficient algorithm makes use of an alternative definition of the semi-unification problem, which we will give now. An **extended substitution** θ is a function from atoms to simple terms. Extended substitutions are lifted to terms according to the equations $\theta(s \cdot t) = \theta s \cdot \theta t$ and $\theta a = a$. Note that extended substitutions ignore the intended connection between a simple variable x and its instance variables αx. For example, we can have $\theta x = a \cdot a$ while $\theta(\alpha x) = a$.

To regain the connection between instance variables and simple variables, we define a restriction on extended substitutions. A **compatible substitution** θ is an extended substitution such that $\theta(\alpha x) = \theta(\hat{\alpha}(\theta x))$ for all simple variables x and substitution variables α.

Now, we can define an alternative to semi-unifiers. A **compatible unifier** θ of a system of equations E is a compatible substitution such that $\theta s = \theta t$ for all $s \doteq t \in E$. To show the equivalence to semi-unifiers, we need a standard notion from unification theory. A substitution σ is **idempotent** if $\sigma(\sigma s) = \sigma s$ for all terms s.

Lemma 34 *An anchored system of equations E has a semi-unifier iff it has a compatible unifier.*[1]

Proof We show the two directions of the biconditional separately.

- If E has a semi-unifier, then it also has an idempotent semi-unifier. This follows from the construction in the proof of Lemma 26. Let σ be an

[1]The restriction to anchored systems of equations is not strictly necessary and could be removed by showing that every system of equations that has a semi-unifier also has an idempotent semi-unifier. For systems of inequalities, this has been shown by Henglein [6].

idempotent semi-unifier of E. Then there is an assignment A such that $A(\sigma s) = A(\sigma t)$ for all $s \doteq t \in E$. We define the extended substitution

$$\begin{aligned} \theta x &:= \sigma x \\ \theta(\alpha x) &:= A(\sigma(\alpha x)) \end{aligned}$$

We first prove that $\theta s = A(\sigma s)$ for all terms s by induction on s. If $s = x$, then $\theta x = \sigma x = A(\sigma x)$ because σx is simple. The other cases are trivial.

Using this, we can show that θ is a compatible unifier of E. We have that $\theta s = A(\sigma s) = A(\sigma t) = \theta t$ for all $s \doteq t \in E$. It remains to show that θ is compatible. This is the case because $\theta(\hat{\alpha}(\theta x)) = \theta(\hat{\alpha}(\sigma x)) = \theta(\sigma(\alpha x)) = A(\sigma(\sigma(\alpha x))) = A(\sigma(\alpha x)) = \theta(\alpha x)$ for all simple variables x and substitution variables α.

- Let θ be a compatible unifier of E. We define the substitution σ with $\sigma x := \theta x$ and the assignment A with $(A\alpha)x := \theta(\alpha x)$.

  For σ to be a semi-unifier of E, it suffices to show that $\theta s = A(\sigma s)$ for all terms s. We prove this by induction on s. If $s = x$, then $\theta x = \sigma x = A(\sigma x)$ because σx is simple. If $s = \alpha x$, then

$$\begin{aligned} A(\sigma(\alpha x)) &= A(\hat{\alpha}(\sigma x)) && \text{by the definition of substitutions} \\ &= A(\hat{\alpha}(\theta x)) && \text{by the definition of } \sigma \\ &= \theta(\hat{\alpha}(\theta x)) && \text{by a nested induction on } \hat{\alpha}(\theta x) \\ &= \theta(\alpha x) && \text{because } \theta \text{ is compatible} \end{aligned}$$

  The case that s is not an atom is trivial. ∎

4.2 Fast Semi-Unification Rules

In this section, we present fast rules to solve AnSUP.

Given a system of equations E, we call an extended substitution θ such that $\theta s = \theta t$ for all $s \doteq t \in E$ a **weak unifier** of E. Our algorithm works by transforming a system of equations until weak unifiers and compatible unifiers almost coincide. Since finding a weak unifier is a standard unification problem, this approach allows us to stay very close to the almost linear unification algorithm that uses the unification-closure method (see Section 2.3.3 in Baader and Snyder [1]). Like in the unification-closure method, we use a union-find structure that implements an equivalence relation on terms with fixed representatives for each equivalence class. Leaving out the implementation details, we assume that we can efficiently manipulate idempotent functions ρ from terms to terms. We call ρs the **representative** of s. The corresponding equivalence kernel $(\simeq_\rho)$ is defined by $s \simeq_\rho t$ iff $\rho s = \rho t$. It satisfies that $s \simeq_\rho \rho s$ and thus ρ selects a representative from every equivalence class of $(\simeq_\rho)$.

In the algorithm, we will have to merge equivalence classes and select a common representative. We give precedence to terms without atoms, then terms that are not atoms, then simple variables and finally instance variables. This is achieved with the following operation.

$$(\rho, s \simeq t)u := \begin{cases} \rho u & \text{if } \rho u \neq \rho s \text{ and } \rho u \neq \rho t \\ \rho t & \text{otherwise, if } \rho t \text{ contains no atoms} \\ \rho s & \text{otherwise, if } \rho s \text{ is not an atom or } \rho t = \alpha x \text{ for some } \alpha x \\ \rho t & \text{otherwise} \end{cases}$$

The rules for the fast semi-unification algorithm are shown in Figure 4.1. They operate on a pair $\rho; E$ consisting of an idempotent function ρ from terms to terms and a system of equations E. We call such a pair a **state**. In contrast to the rules from Section 3.3, we treat the equations in E as unoriented pairs, that is, we do not distinguish between $s \doteq t$ and $t \doteq s$.

To solve a system of equations E, we start with the **initial state** $\rho_E; E$ where $\rho_E(\alpha x) = s$ for some $\alpha x \doteq s \in E$ if such an anchor exists and $\rho_E t = t$ for all other terms t. The rules work by removing or replacing equations from E while merging equivalence classes of $(\simeq_\rho)$.

We associate every state $\rho; E$ with the system of equations

$$\mathcal{E}(\rho; E) := \{s \doteq \rho s \mid s \neq \rho s\} \cup E$$

We say that a state $\rho; E$ **implies an equation** $s \doteq t$ if we have $\theta s = \theta t$ for every weak unifier θ of $\mathcal{E}(\rho; E)$.

Proposition 35 *A state $\rho; E$ implies every equation $s \doteq t$ with $s \simeq_\rho t$.*

We say that an extended substitution θ is a **weak (compatible) unifier of** $\rho; E$ if θ is a weak (compatible) unifier of $\mathcal{E}(\rho; E)$. The rules maintain the set of compatible unifiers of $\rho; E$. Moreover, as we will show in Lemma 44, if E is empty, then we can construct a principal weak unifier θ of $\rho; E$ that is compatible and hence θ is a principal compatible unifier of the initial system of equations.

The rules R'_{bop} and R'_{refl} correspond to rules in the standard unification-closure algorithm. The rule R'_{elim} is special in that it cares about the relationship between simple variables and their instance variables. It needs to extract a simple term $\lfloor s \rfloor_\rho$ out of ρ.

$$\lfloor s \rfloor_\rho = \begin{cases} \rho s & \text{if } \rho s \text{ contains no atoms} \\ \text{select a smallest simple term } t \text{ with } t \simeq_\rho \alpha x & \text{otherwise, if } s = \alpha x \\ \lfloor s_1 \rfloor_\rho \cdot \lfloor s_2 \rfloor_\rho & \text{otherwise, if } s = s_1 \cdot s_2 \\ s & \text{otherwise, if } s \text{ is simple} \end{cases}$$

Proposition 36 *Every state $\rho; E$ implies the equation $\lfloor s \rfloor_\rho \doteq s$.*

$$
\begin{array}{ll}
R'_{\text{refl}} & \rho; E, s \doteq t \implies \rho; E \\
 & \text{if } \rho s = \rho t \\
R'_{\text{bop}} & \rho; E, s \doteq t \implies \rho, s \simeq t; E, s_1 \doteq t_1, s_2 \doteq t_2 \\
 & \text{if } \rho s \neq \rho t,\ \rho s = s_1 \cdot s_2 \text{ and } \rho t = t_1 \cdot t_2 \\
R'_{\text{elim}} & \rho; E, s \doteq t \implies \rho, s \simeq t; E, \alpha_1 x \doteq \hat{\alpha}_1 \lfloor t \rfloor_\rho, \ldots, \alpha_n x \doteq \hat{\alpha}_n \lfloor t \rfloor_\rho \\
 & \text{if } \rho s \neq \rho t,\ \rho s = x \text{ and } \alpha_1 x, \ldots, \alpha_n x \text{ are all instance variables} \\
 & \text{of } x \text{ in } \mathcal{E}(\rho; E, s \doteq t)
\end{array}
$$

Figure 4.1: Fast Semi-Unification Rules

Note that $\lfloor s \rfloor_\rho$ can be undefined if s contains an instance variable αx such that there is no simple term t with $\alpha x \simeq_\rho t$. This would lead to R'_{elim} being not applicable. However, we will prevent this from happening using the anchoredness condition.

4.3 Correctness

It is easy to see that the rules are sound in the following sense.

Proposition 37 *If $\rho; E \implies \rho'; E'$, then θ is a compatible unifier of $\mathcal{E}(\rho; E)$ iff θ is a compatible unifier of $\mathcal{E}(\rho'; E')$.*

It remains to show that whenever we reach a terminal state that was obtained by applying the rules to an initial state constructed from an anchored system of equations, we can efficiently determine a principal compatible unifier or the fact that there are no compatible unifiers. This proof requires a number of invariants.

We call a state $\rho; E$ **proper** if the following properties are satisfied.

1. If ρs is an instance variable, then all terms in the equivalence class $[s]_{\simeq_\rho}$ are instance variables.
   If ρs is a simple variable, then all terms in the equivalence class $[s]_{\simeq_\rho}$ are atoms.
   If s contains no atoms, then ρs contains no atoms.
2. If $s_1 \cdot s_2 \simeq_\rho t_1 \cdot t_2$, then (s_i, t_i) is in the reflexive transitive closure of $(\simeq_\rho) \cup E$ for $i \in \{1, 2\}$.

Lemma 38 *For every state $\rho; E$, rule application preserves the invariant that $\rho; E$ is proper.*

Proof The first property of a proper state follows from the definition of $\rho, s \simeq t$. It remains to show that the second property is preserved. For the rule R'_{elim}, this follows from the first property. For R'_{bop}, this follows from the fact that we add the necessary equations to E. ■

The initial state $\rho_E; E$ is proper for every system of equations E. Thus all states that will ever occur are proper. In the following, we will assume that all states are proper without stating it again.

To argue that the rules preserve anchoredness, we extract a system of equations from a state and require this system of equations to be anchored. It turns out that $\mathcal{E}(\rho; E)$ is not sufficient for this purpose and we need to add the **additional anchors**

$$\mathcal{A}(\rho; E) := \{\alpha x \doteq s \mid s \text{ simple}, \alpha x \simeq_\rho s \text{ and } \rho(\alpha x) \text{ contains an atom}\}$$

We call a state $\rho; E$ **anchored** with a PER $(\sim)$ if the system of equations $\mathcal{E}(\rho; E) \cup \mathcal{A}(\rho; E)$ is anchored with $(\sim)$ and for every anchor $\alpha x \doteq s \in E$, we have $\alpha x \simeq_\rho s$.

It might seem strange that we remove anchors from $\mathcal{A}(\rho; E)$ once a representative contains no atoms. This is to take care for situations like $E = \{x \doteq a, y \doteq a, \alpha x = a\}$ where x and y must not be related by a PER with which E is anchored, but x and y might happen to share the representative a.

Proposition 39 *If the system of equations E is anchored, then the initial state $\rho_E; E$ is anchored.*

In the following, we write $s \sim t$ if $X \sim Y$ for all atoms X, Y in s or t. Note that if t contains an atom, $s \sim t$ and $t \sim u$, then $s \sim u$.

Lemma 40 *If $\rho; E, s \doteq t$ is anchored with a PER $(\sim)$, then $\rho, s \simeq t; E$ is also anchored with $(\sim)$.*

Proof Let s', t' be the terms such that $\{s', t'\} = \{\rho s, \rho t\}$ and $t' = (\rho, s \simeq t)s$. The system of equations $\mathcal{E}(\rho, s \simeq t; E)$ can be obtained from $\mathcal{E}(\rho; E, s \doteq t)$ by removing $s \doteq t$, replacing some equations $u \doteq s'$ with $u \doteq t'$ and possibly adding the equation $s' \doteq t'$. We have that $u \sim t'$ because either t' contains no atoms or both t' and s' contain an atom since $\rho, s \simeq t; E$ is proper and then $u \sim t'$ follows from $u \sim s'$ and $s' \sim t'$. Thus we have $u_1 \sim u_2$ for all equations $u_1 \doteq u_2 \in \mathcal{E}(\rho, s \simeq t; E)$. It remains to show that this also holds for $\mathcal{A}(\rho, s \simeq t; E)$ and that $\mathcal{E}(\rho, s \simeq t; E) \cup \mathcal{A}(\rho, s \simeq t; E)$ contains sufficiently many anchors. For this, we distinguish two cases.

- Consider the case that t' contains an atom. Then $\mathcal{A}(\rho, s \simeq t; E)$ differs from $\mathcal{A}(\rho; E)$ in that $\mathcal{A}(\rho, s \simeq t; E)$ contains additional anchors $\alpha x \doteq u$ with $\alpha x \simeq_{\rho, s \simeq t} u$. We will show that $\alpha x \sim u$. We can assume that $\alpha x \not\simeq_\rho u$ because the other case is trivial. Now assume w.l.o.g. (for the other case, switch s' and t') that $\alpha x \simeq_\rho s'$ and $u \simeq_\rho t'$. Since $\rho, s \simeq t; E$ is proper, we have that s' contains an atom and thus $\alpha x \sim u$ since $\alpha x \sim s'$, $s' \sim t'$ and $t' \sim u$. It remains to show that there are still anchors for all instance variables in $(\sim)$. This is the case since all anchors removed in $\mathcal{E}(\rho, s \simeq t; E)$ are still contained in $\mathcal{A}(\rho, s \simeq t; E)$.

- Consider the case that t' does not contain an atom. Then $\mathcal{A}(\rho, s \simeq t; E)$ differs from $\mathcal{A}(\rho; E)$ in that $\mathcal{A}(\rho, s \simeq t; E)$ does not contain anchors for the instance variables in the equivalence class $[s]_{\rho, s\simeq t}$. However, we still have enough anchors because for every instance variable $\alpha x \in [s]_{\rho, s\simeq t}$, we have that $(\rho, s \simeq t)(\alpha x) = t'$ and thus the anchor $\alpha x \doteq t'$ is in $\mathcal{E}(\rho, s \simeq t; E)$. ■

Proposition 41 *If $\rho; E$ is anchored with $(\sim)$ and t occurs in $\mathcal{E}(\rho; E)$, then $t \sim \lfloor t \rfloor_\rho$.*

Lemma 42 *For every state $\rho; E$, rule application preserves that $\rho; E$ is anchored with an invariant PER.*

Proof For R'_{refl}, the claim is trivial. For R'_{bop}, the claim follows from Lemma 40.
For R'_{elim}, consider a state $\rho; E, s \doteq t$ with $\rho s = x$ and $\alpha_1 x, \ldots, \alpha_n x$ being the instance variables of x in $\mathcal{E}(\rho; E)$. Apart from the conclusion of Lemma 40, we additionally need to show that $\alpha_i x \sim \hat{\alpha}_i \lfloor t \rfloor_\rho$ for $i \in \{1, \ldots, n\}$. This is the case because

- $x \sim s$ and s is an atom because $\rho; E$ is proper,
- $s \sim t$ and t contains atoms if $\lfloor t \rfloor_\rho$ contains atoms,
- $t \sim \lfloor t \rfloor_\rho$ by Proposition 41 and
- $\alpha_i x \sim \alpha_i x$. ■

We call a simple variable x **bound** in a state $\rho; E$ if $\rho x \neq x$. The rule R'_{elim} makes sure that whenever a simple variable x gets bound, sufficiently many equations are added to the state to express the relationship between x and its instance variables. This is expressed by the following property, which is somewhat involved because we have to prevent these equations from being cyclic. We say that a state $\rho; E$ is **pre-compatible** if there is a well-founded strict partial order $(>)$ on simple variables such that $x > y$ implies $x \simeq_\rho y$ and for every bound simple variable x and for every instance variable αx that occurs in $\mathcal{E}(\rho; E)$, there is a simple term s such that $x > s$ if s is a simple variable and $\rho; E$ implies the equations $x \doteq s$ and $\alpha x \doteq \hat{\alpha} s$.

Lemma 43 *For every anchored state $\rho; E$, rule application preserves the invariant that $\rho; E$ is pre-compatible.*

Proof For every rule application $\rho; E \Longrightarrow \rho'; E'$, we have that a weak unifier of $\mathcal{E}(\rho'; E')$ is also a weak unifier of $\mathcal{E}(\rho; E)$. Thus $\rho'; E'$ implies every equation that is implied by $\rho; E$. Also, by Lemma 42, the set of instance-variables in $\mathcal{E}(\rho; E)$ and $\mathcal{E}(\rho'; E')$ is identical. So we only need to consider R'_{elim}, since all other rules do not change the set of bound simple variables. Consider a state $\rho; E, s \doteq t$ with $\rho s = x$ and an application $\rho; E, s \doteq t \Longrightarrow \rho'; E'$ of R'_{elim} to this state. We need to show that the new equations in E' suffice for $\rho'; E'$ to be pre-compatible. We have that x is the only newly bound variable. Also, $\mathcal{E}(\rho'; E')$ contains the same instance variables as $\mathcal{E}(\rho; E, s \doteq t)$ and $\rho'; E'$ implies $x \doteq \lfloor t \rfloor_\rho$ by Proposition 36. Thus the equations added in E' satisfy all conditions for $\rho'; E'$ to be pre-compatible if we can extend $(>)$ such that $x > \lfloor t \rfloor_\rho$ if $\lfloor t \rfloor_\rho$ is a simple variable. This is possible because $\rho x \neq \rho t$ and hence $x \not\simeq_\rho \lfloor t \rfloor_\rho$ if $\lfloor t \rfloor_\rho$ is a simple variable. ■

If no rule applies to a pre-compatible and anchored state $\rho; E$, then we want to extract a principal compatible unifier from $\rho; E$ if one exists. Given that $\rho; E$ has a weak unifier θ, we can define the following function by induction on the size of θs (this works because $\theta t = \theta(\rho t)$ for all terms t).

$$\lceil s \rceil_\rho = \begin{cases} \lceil s_1 \rceil_\rho \cdot \lceil s_2 \rceil_\rho & \text{if } \rho s = s_1 \cdot s_2 \\ \rho s & \text{otherwise} \end{cases}$$

Since $\rho; E$ is proper and anchored, we have that $\lceil s \rceil_\rho$ is a simple term. Moreover, $\rho; E$ implies $s \doteq \lceil s \rceil_\rho$ and $\lceil s \rceil_\rho$ only contains simple variables x with $\rho x = x$. We obtain a principal compatible unifier θ_ρ by defining $\theta_\rho X := \lceil X \rceil_\rho$ for all atoms X.

Lemma 44 *If no rule applies to a pre-compatible and anchored state $\rho; E$ that has a weak unifier, then θ_ρ is a principal compatible unifier of $\rho; E$.*

Proof Assume $\rho; E$ has a weak unifier θ. First, we show that E is empty. Suppose, for contradiction, that E contains an equation $s \doteq t$. Since $\rho; E$ is anchored, it is impossible that ρs is an instance variable or that $\lfloor t \rfloor_\rho$ is undefined. Thus, the rule conditions cover all cases that are not obviously contradictory and some rule has to apply contradicting our assumptions.

Next, we will show that θ_ρ is a weak unifier of $\mathcal{E}(\rho; E)$. For this, it suffices to show that $\theta_\rho s = \theta_\rho(\rho s)$ for all terms s. Since $\lceil s \rceil_\rho = \lceil \rho s \rceil_\rho$, this follows from the fact that $\theta_\rho s = \lceil s \rceil_\rho$, which we prove now by induction on s.

If s is an atom, then this follows from the definition of θ_ρ. If $s = a$, then $\theta_\rho a = a = \theta a = \theta(\rho a) = \rho a = \lceil a \rceil_\rho$. So let $s = s_1 \cdot s_2$ and $\rho s = s'_1 \cdot s'_2$. Note that ρs cannot be an atom because $\rho; E$ is proper. Also, we have that $\rho s_i = \rho s'_i$ for $i \in \{1, 2\}$ because $s \simeq_\rho \rho s$, $\rho; E$ is proper and E is empty. Thus by induction, we have $\theta_\rho s = \theta_\rho s_1 \cdot \theta_\rho s_2 = \lceil s_1 \rceil_\rho \cdot \lceil s_2 \rceil_\rho = \lceil s'_1 \rceil_\rho \cdot \lceil s'_2 \rceil_\rho = \lceil s \rceil_\rho$.

We have that θ_ρ is a principal weak unifier of $\rho; E$ because $\theta s = \theta \lceil s \rceil_\rho$ and hence $\theta = \theta \circ \theta_\rho$ for every weak unifier θ of $\rho; E$.

It remains to show that θ_ρ is compatible. This will rely on the fact that $\rho; E$ is pre-compatible with a well-founded strict partial order $(>)$ on simple variables. We need to show that $\theta_\rho(\alpha x) = \theta_\rho(\hat{\alpha}(\theta_\rho x))$ for all simple variables x and substitution variables α. We prove this by induction on the well-founded lexicographical ordering on simple variables which, given x and y, first compares the sizes of $\theta_\rho x$ and $\theta_\rho y$ and if they are equal, compares x and y according to $(>)$. Consider an instance variable αx. Since $\rho; E$ is pre-compatible, there is a simple term s such that $x > s$ if s is a simple variable, $\theta_\rho x = \theta_\rho s$ and $\theta_\rho(\alpha x) = \theta_\rho(\hat{\alpha} s)$.

If s is a simple variable y, then the inductive hypothesis for y yields $\theta_\rho(\hat{\alpha}(\theta_\rho x)) = \theta_\rho(\hat{\alpha}(\theta_\rho y)) = \theta_\rho(\alpha y) = \theta_\rho(\alpha x)$.

If s is not an atom, then we can apply the inductive hypothesis to every simple variable y in s since the size of $\theta_\rho y$ is strictly smaller than the size of $\theta_\rho s = \theta_\rho x$. Thus $\theta_\rho(\hat{\alpha}(\theta_\rho x)) = \theta_\rho(\hat{\alpha}(\theta_\rho s)) = \theta_\rho(\hat{\alpha} s) = \theta_\rho(\alpha x)$. This concludes the proof that θ_ρ is compatible.

Since θ_ρ is compatible and a principal weak unifier, it is a principal compatible unifier. ∎

So given an anchored system of equations E, we can compute a principal compatible unifier of E as follows.

1. Start with the initial state $\rho_E; E$.
2. Apply the rules as long as possible, yielding a state $\rho'; E'$.
3. Check if $\mathcal{E}(\rho'; E')$ has a weak unifier using a standard unification algorithm. If it does, then θ_ρ is the principal compatible unifier we searched for. Otherwise, there are no compatible unifiers.

4.4 Complexity

In this section, we will argue that it is possible to implement the algorithm in polynomial time. This needs two standard ingredients.

First, we represent terms by nodes in a directed acyclic graph (DAG). For example, the term $(f \cdot (g \cdot \alpha x)) \cdot (g \cdot \alpha x)$ is represented by the uppermost node in the following DAG

$$\begin{tikzcd}
 & & \cdot \arrow[dl] \arrow[dd, bend left] & \\
 & \cdot \arrow[dl] \arrow[dr] & & \\
f & & \cdot \arrow[dl] \arrow[dr] & \\
 & g & & \alpha x
\end{tikzcd}$$

In the following we assume that there is a global DAG such that at every point during the execution of the algorithm, all terms are represented by nodes in the global DAG. Moreover, we assume that this global DAG is minimal in the sense that two different nodes always represent two different terms. This can be achieved with asymptotically no overhead using hash-consing [5]. Whenever we need a new term $s \doteq t$ for existing terms s and t, we either find a suitable node using a hash-map that maps pairs of nodes to their parent or else we create a single new node and add edges to the nodes for s and t. The fact that the global DAG is minimal allows us to test terms equality in time $\mathcal{O}(1)$. Despite the fact that we only manipulate nodes, we will continue to write a node in the global DAG as the term it represents. In the following, we will call the cardinality of the set of subterms of a term s the **size** of s since this corresponds exactly to the number of nodes in the global DAG that are reachable from the node representing s. Moreover, we say that a set of terms M has **size** $|\{s \mid s \text{ occurs as subterm in } M\}|$ and a system of equations E has **size** $|E| + |\{s \mid s \text{ occurs as subterm in } E\}|$. Again, this corresponds to the size of a representation in memory because the global DAG allows for structure sharing between different terms.

Second, we represent ρ with the usual union-find structure [25], which can perform finding a representative and computing $\rho, s \simeq t$ in an amortized running time of $\mathcal{O}(\alpha(n))$ where $\alpha(n)$ is the inverse of the Ackermann function. In the following, we will write α for $\alpha(\text{size of } E)$ where E is the initial system of equations.

To argue the complexity of the algorithm, we will associate every state $\rho; E$ with a run-time measure $r(\rho; E)$ and show that every rule application $\rho; E \Longrightarrow \rho'; E'$ can be performed in time $\mathcal{O}(r(\rho; E) - r(\rho'; E'))$. Then the running time of the overall algorithm is $\mathcal{O}(r(\rho; E))$ for an initial state $\rho; E$.

We set

$$r(\rho; E) := (n_1 + 8n_2(n_3 + n_4 n_5))\alpha$$

where

- n_1 is $3 \cdot |\{\rho s \mid s \text{ occurs as subterm in } \mathcal{E}(\rho; E)\}| + \text{size of } \mathcal{E}(\rho; E)$
- n_2 is the number of different instance variables αx in $\mathcal{E}(\rho; E)$ such that x is unbound.
- n_3 is the size of the largest term in $\mathcal{E}(\rho; E)$
- n_4 is the number of different unbound simple variables that occur in $\mathcal{E}(\rho; E)$.
- n_5 is the size of $\{s \mid \text{there is } \alpha x \text{ s.t. } s \text{ is a smallest simple term with } s \simeq_\rho \alpha x\}$

Since $r(\rho_E; E) \in \mathcal{O}((\text{size of } E)^3\alpha)$, we only need to check that the individual rules sufficiently decrease $r(\rho; E)$ to prove that every sequence of rule applications needs time $\mathcal{O}((\text{size of } E)^3\alpha)$.

R'_{refl} Rule application needs time $\mathcal{O}(\alpha)$. We have that n_1 is decreased while n_2, n_3, n_4 and n_5 are not changed.

R'_{bop} Rule application needs time $\mathcal{O}(\alpha)$. We have that n_1 is decreased while n_2, n_3, n_4 and n_5 are not changed.

R'_{elim} Let n'_2 be the new value of n_2 after the rule application. Rule application needs time $\mathcal{O}((n_2 - n'_2)n_3\alpha)$. The simple term $\lfloor s \rfloor_\rho$ can be computed efficiently if we permanently associate every representative ρu with a smallest simple term t such that $t \simeq_\rho \rho u$.

We have that n_2 is not increased and n_4 is decreased by 1 while n_1 is increased by at most $7(n_2 - n'_2)n_3$ and n_3 is increased by at most n_5. So taking everything together, $r(\rho; E)$ is decreased by at least $(n_2 - n'_2)n_3\alpha$.

Since the final test if $\mathcal{E}(\rho; E)$ has a weak unifier for a terminal state $\rho; E$ can be performed in time linear to the size of $\mathcal{E}(\rho; E)$ using a standard unification algorithm, the overall performance of our algorithm is $\mathcal{O}(n^3\alpha(n))$ where n is the size of the initial system of equations.

5 Conclusion

This thesis presents the first unification algorithm for nonnested recursion schemes. Based on a novel coinductive definition of $\mathcal{S}$-equivalence, we establish the existence of principal $\mathcal{S}$-unifiers. Our method for solving $\mathcal{S}$-unification problems works by a reduction to a new decidable semi-unification problem we call AnSUP (anchored semi-unification problem). We present an algorithm that computes a principal semi-unifier whenever one exists and prove that this algorithm has time complexity $\mathcal{O}(n^3\alpha(n))$ where $\alpha(n)$ is the inverse of the Ackermann function.

AnSUP is quite different from other decidable semi-unification problems we know of.

- The uniform fragment [8, 17] only allows for a single substitution variable (or, in the usual representation, a single inequality). In contrast to this, our reduction requires many substitution variables.
- The acyclic fragment [10] and its extension to the R-acyclic fragment [15] disallow cyclic inequalities in the following sense. There must not be a sequence of inequalities $s_1 \dot{\preceq} t_1, \ldots, s_n \dot{\preceq} t_n$ such that t_n and s_1 share a variable and for all $i \in \{1, \ldots, n-1\}$, t_i and s_{i+1} share a variable. In contrast to this, we allow arbitrary cycles and need them in the reduction.
- The left-linear fragment [7] does not allow that a variable occurs twice in the left-hand side s of an inequality $s \dot{\preceq} t$. Adding inequalities of the form $(s, s) \dot{\preceq} (s, t)$ is impossible since this would allow for the same expressive power as unrestricted semi-unification, which is undecidable. So it is impossible to express ordinary unification with the left-linear fragment.
- The quasi-monadic fragment [14] does not allow terms containing two different variables.
- There is a decidable fragment that only allows two variables [12].

To the best of our knowledge, AnSUP is the only fragment that allows for cyclic inequalities, an unrestricted number of variables and subsumes ordinary unification.

Future Work

We would like to see if our algorithms can be used in compiler verification. There, it would be especially interesting to see how they perform in practice. Since we have a certifying algorithm for $\mathcal{S}$-equivalence, it should be possible to integrate our method into a formally verified compiler by only verifying the certificate checker while leaving the reduction to SUP and the algorithm for AnSUP unverified but validated by the certificate checker.

For future research, it might be interesting to investigate the combination of $\mathcal{S}$-equivalence with additional theories like in E-unification [1]. This might lead to applications in compiler verification that go beyond the equivalences in the Herbrand interpretation that we can detect currently.

The anchored fragment of semi-unification is incomparable to all known fragments. So it might be possible to combine it with another fragment to obtain one that strictly contains both. We expect that this is possible for the quasi-monadic fragment.

Bibliography

[1] F. Baader and W. Snyder. Unification theory. In *Handbook of Automated Reasoning*, volume 1, pages 445–534. Elsevier, 2001.

[2] B. Courcelle. A representation of trees by languages II. *Theoretical Computer Science*, 7(1):25–55, 1978.

[3] N. Dershowitz, S. Kaplan, and D.A. Plaisted. Rewrite, rewrite, rewrite, rewrite, rewrite, ... *Theoretical Computer Science*, 83(1):71–96, 1991.

[4] E. Eder. Properties of substitutions and unifications. *Journal of Symbolic Computation*, 1(1):31–46, 1985.

[5] Eiichi Goto. Monocopy and associative algorithms in extended LISP. Technical Report TR-74-03, University of Tokyo, 1974.

[6] F. Henglein. Type inference and semi-unification. In *LISP and functional programming*, pages 184–197. ACM, 1988.

[7] F. Henglein. Fast left-linear semi-unification. In *Advances in Computing and Information — ICCI'90*, volume 468 of *Lecture Notes in Computer Science*, pages 82–91. Springer, 1990.

[8] D. Kapur, D. Musser, P. Narendran, and J. Stillman. Semi-unification. In *Foundations of Software Technology and Theoretical Computer Science*, pages 435–454. Springer, 1988.

[9] A.J. Kfoury, J. Tiuryn, and P. Urzyczyn. The undecidability of the semi-unification problem. In *Proceedings of the twenty-second annual ACM symposium on Theory of computing*, pages 468–476. ACM, 1990.

[10] A.J. Kfoury, J. Tiuryn, and P. Urzyczyn. An analysis of ML typability. *Journal of the ACM (JACM)*, 41(2):368–398, 1994.

[11] D.S Lankford and D.R. Musser. A finite termination criterion. *Unpublished draft*, 1978.

[12] H. Leiß. Decidability of semi-unification in two variables. Technical Report INF-2-ASE-9-89, Siemens, Munich, 1989.

[13] H. Leiß. Polymorphic recursion and semi-unification. In *CSL'89*, pages 211–224. Springer, 1990.

[14] H. Leiß and F. Henglein. A decidable case of the semi-unification problem. In *Mathematical Foundations of Computer Science 1991*, volume 520 of *Lecture Notes in Computer Science*, pages 318–327. Springer, 1991.

[15] B. Lushman and G.V. Cormack. A larger decidable semiunification problem. In *Proceedings of the 9th ACM SIGPLAN international conference on Principles and practice of declarative programming*, pages 143–152. ACM, 2007.

[16] R.M. McConnell, K. Mehlhorn, S. Näher, and P. Schweitzer. Certifying algorithms. *Computer Science Review*, 5(2):119–161, 2011.

[17] A. Oliart and W. Snyder. Fast algorithms for uniform semi-unification. *Journal of Symbolic Computation*, 37(4):455–484, 2004.

[18] P. Pudlák. On a unification problem related to Kreisel's conjecture. *Commentationes Mathematicae Universitatis Carolinae*, 29(3):551–556, 1988.

[19] B.K. Rosen. Program equivalence and context-free grammars. *Journal of Computer and System Sciences*, 11(3):358–374, 1975.

[20] V. Sabelfeld. The tree equivalence of linear recursion schemes. *Theoretical Computer Science*, 238(1–2):1–29, 2000.

[21] D. Sangiorgi. On the bisimulation proof method. *Mathematical Structures in Computer Science*, 8(5):447–479, 1998.

[22] G. Sénizergues. The equivalence problem for deterministic pushdown automata is decidable. In *Automata, languages and programming*, volume 1256 of *Lecture Notes in Computer Science*, pages 671–681. Springer, 1997.

[23] G. Smolka and T. Tebbi. Unification modulo nonnested recursion schemes via anchored semi-unification. In *24rd International Conference on Rewriting Techniques and Applications (RTA'13)*, Leibniz International Proceedings in Informatics (LIPIcs). Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2013. To appear.

[24] C. Stirling. Deciding DPDA equivalence is primitive recursive. In *Automata, languages and programming*, volume 2380 of *Lecture Notes in Computer Science*, pages 774–774. Springer, 2002.

[25] R.E. Tarjan. Efficiency of a good but not linear set union algorithm. *Journal of the ACM (JACM)*, 22(2):215–225, 1975.