A Larger Decidable Semiunification Problem

Brad Lushman
University of Waterloo
bmlushma@plg.uwaterloo.ca

Gordon V. Cormack
University of Waterloo
gvcormac@uwaterloo.ca

ABSTRACT

We present a graph-theoretic framework in which to study instances of the semiunification problem (SUP), which is known to be undecidable, but has several known and important decidable subsets. One such subset, the *acyclic* semiunification problem (ASUP), has proved useful in the study of polymorphic type inference. We present graph-theoretic criteria in our framework that exactly characterize the ASUP acyclicity constraint. We then use our framework to find a decidable subset of SUP (which we call R-ASUP), which has a more natural description than ASUP, and strictly contains it.

1. INTRODUCTION

Given a term algebra comprising a set $\mathbb{F}$ of functors with given arities and a set of variables, unification is the problem of determining, for a given set of term equations, whether there is a substitution on the variables in the equations that satisfies them all. Formally, we are given a set $\{\tau_i = \mu_i\}_{i=1}^N$ of term equations, and we seek a substitution S such that, for all i, $\tau_i S = \mu_i S$.

Semiunification is a related problem, in which the set of term equations becomes a set of term *inequations* $\{\tau_i \leq \mu_i\}_{i=1}^N$, and is defined formally in Section 3. Here, a solution requires only that each μ_i be a substitution instance of the corresponding τ_i, rather than an exact match.

While unification is known to be solvable in linear time, semiunification in general is undecidable. A subset known as the acyclic semiunification problem is known to be solvable (though DEXPTIME-complete), and has proved useful in at least one application domain. In this paper, we introduce another subproblem, the *R-acyclic* semiunification problem, which is more natural than ASUP, strictly larger, and still decidable. Moreover, it produces quadratically smaller problem instances than ASUP in the domain of typing algorithms, and we conjecture that it may be similarly advantageous in other contexts.

Apart from its connections to polymorphic type inference, SUP is an interesting problem, itself worthy of study. Applications of SUP can be found in fields such as logic programming [2], computational linguistics [3], and program analysis [4]; for this reason, we present the results in this paper in an application-independent way, so that they may be readily adopted to other application domains.

2. TERMS, VARIABLES, AND SUBSTITUTIONS

In this section, we define some basic notions related to terms, variables and substitutions, which we will need throughout the remainder of this paper.

DEFINITION 1 (VARS). *Given a term τ, we denote by* $\mathrm{Vars}(\tau)$ *the set of variables occurring in τ:*

$$\begin{aligned} \mathrm{Vars}(\alpha) &= \{\alpha\} \\ \mathrm{Vars}(f(\tau_1, \ldots, \tau_n)) &= \bigcup_{i=1}^{n} \mathrm{Vars}(\tau_i) \quad \textit{(where } \mathrm{arity}(f) = n\textit{)} \end{aligned}$$

DEFINITION 2 (PATH). *For a term algebra comprising a set $\mathbb{F}$ of functors, a path (denoted by Σ) is a string over the set*

$$\{f_i | f \in \mathbb{F}, 1 \leq i \leq \mathrm{arity}(f)\}$$

that acts as a partial function on terms as follows:

$$\begin{aligned} \epsilon(\tau) &= \tau \textit{ for all } \tau \\ (\Sigma f_i)(f(\tau_1, \ldots, \tau_{\mathrm{arity}(f)})) &= \Sigma(\tau_i)\ (1 \leq i \leq \mathrm{arity}(f)), \end{aligned}$$

where τ ranges over terms and ϵ is the empty path.

DEFINITION 3 (SUBSTITUTION). *Given a term algebra $\mathbb{T}$ comprising a set $\mathbb{F}$ of functors with associated*

arities, and a set $\mathbb{X}$ *of variables, a* substitution *is a map* $\sigma : \mathbb{X} \to \mathbb{T}$ *which is an identity map on all but finitely many variables. The* domain *of a substitution* σ*, denoted* $\mathrm{dom}(\sigma)$*, is the set* $\{x \in \mathbb{X} \mid \sigma(x) \neq x\}$ *of variables on which* σ *is not an identity map. The notation* $[\tau/\alpha]$ *denotes a substitution* σ *for which* $\mathrm{dom}(\sigma) = \{\alpha\}$ *and* $\sigma(\alpha) = \tau$*. Substitutions are often written postfix, so that* $\alpha\sigma$ *has the same meaning as* $\sigma(\alpha)$*. Substitutions extend naturally to maps from terms to terms by the following rule:*

$$f(\tau_1, \ldots, \tau_n)\sigma \equiv f(\tau_1\sigma, \ldots, \tau_n\sigma) \ ,$$

where f *is a functor. Given terms* τ *and* μ*,* μ *is called a* substitution instance *of* τ *if there exists a substitution* σ *such that* $\tau\sigma = \mu$*.*

We include for completeness the definition of the unification problem, of which SUP is a generalization.

Definition 4 (Unification). *Within a given term algebra, an instance of the* unification problem *is a set* $\Gamma = \{\tau_i = \mu_i\}_{i=1}^N$ *of term equations. A substitution* σ *is a solution of the instance* Γ *if, for all* i*,* $\tau_i\sigma = \mu_i\sigma$*.*

The unification problem was first formulated and solved by Robinson [21]. Linear time solutions have since been found [16, 19].

An important property of Robinson's algorithm (though it is by no means unique in this regard) is that it always outputs a *most general unifier* whenever the term equations in the problem instance are satisfiable. In particular, if the algorithm outputs a substitution σ_0 as a solution for a unification instance Γ, and if σ is any other solution of Γ, then there exists a substitution σ' such that $\sigma = \sigma' \circ \sigma_0$.

3. SEMIUNIFICATION AND THE REDEX PROCEDURE

The semiunification problem (SUP) is defined below:

Definition 5 (SUP). *An instance of SUP is a set* $\{\tau_i \leq \mu_i\}_{i=1}^N$ *of inequalities in some term algebra. A substitution* σ *is a solution of SUP if there exist substitutions* $\sigma_1, \ldots, \sigma_N$ *such that*

$$\begin{aligned} \tau_1\sigma\sigma_1 &= \mu_1\sigma \\ &\cdots \\ \tau_N\sigma\sigma_N &= \mu_N\sigma \end{aligned}$$

In particular, semiunification differs from ordinary unification in that we may perform additional substitutions on the left-hand sides of the inequalities in order to make them match the right-hand sides. In other words, σ is a solution of the instance iff, after applying σ throughout the instance, each right-hand side is a substitution instance of the corresponding left-hand side.

Though it was widely believed to be decidable for years, SUP is now known to be undecidable [11]. This result has formed the basis for other undecidability results within SUP's application domains [22]. Kfoury, Tiuryn, and Urzyczyn [11] present a solution semi-procedure for SUP, which we call the *redex procedure* (see Kfoury, Tiuryn, and Urzyczyn [12] and Baaz [1] for alternative solution semi-procedures). Our formulation of the redex procedure is given in Figure 1.

Redex Procedure
Input: SUP instance $\Gamma = \{\tau_i \leq \mu_i\}_{i=1}^N$
Output: substitution σ_Γ that solves Γ

1. Set $\sigma_0 = []$ and $k = 0$.
2. If $\mu_i\sigma_k$ is a substitution instance of $\tau_i\sigma_k$ for all i, set $\sigma_\Gamma = \sigma_k$ and terminate with success.
3. Perform one of the following steps:
   (a) (Redex-I reduction)
   Let Σ be a path and $1 \leq i \leq N$ be such that $\Sigma(\mu_i\sigma_k)$ is a variable and $\Sigma(\tau_i\sigma_k)$ is not a variable.
   Set $\sigma_{k+1} = [\Sigma(\tau_i\sigma_k)'/\Sigma(\mu_i\sigma_k)] \circ \sigma_k$, where $\Sigma(\tau_i\sigma_k)'$ is $\Sigma(\tau_i\sigma_k)$ with all variables renamed consistently to fresh variables.
   (b) (Redex-II reduction)
   Let Σ_1 and Σ_2 be paths, α a variable, and $1 \leq i \leq N$ be such that
   $$\Sigma_1(\tau_i\sigma_k) = \Sigma_2(\tau_i\sigma_k) = \alpha$$
   and
   $$\Sigma_1(\mu_i\sigma_k) \neq \Sigma_2(\mu_i\sigma_k) \ .$$
   If $\Sigma_1(\mu_i\sigma_k)$ and $\Sigma_2(\mu_i\sigma_k)$ are not unifiable, terminate with failure.
   Else, let θ be the most general unifier of $\Sigma_1(\mu_i\sigma_k)$ and $\Sigma_2(\mu_i\sigma_k)$, as output by Robinson's unification algorithm, and set $\sigma_{k+1} = \theta \circ \sigma_k$.
   (c) If neither of steps 3a and 3b is possible, then there is a functor mismatch; terminate with failure.
4. $k := k + 1$; go to step 2.

Figure 1: The redex procedure of Kfoury, Tiuryn, and Urzyczyn.

The redex procedure has the property that it terminates with a correct answer on all SUP instances that possess a solution, and either returns an error or loops forever on SUP instances that do not possess a solution [11].

The substitution output by the redex procedure (when it terminates) is principal (or "most general") in a specific sense, as described below [5]:

THEOREM 1. *Let Γ be a solvable SUP instance on which the redex procedure terminates, and σ_0 the substitution returned by the redex procedure. Let σ be any other substitution that solves Γ. Then there is a substitution σ' such that, for any variable α occurring in Γ, $\alpha\sigma = \alpha\sigma_0\sigma'$. In other words, when viewed as maps with domains restricted to variables in Γ, we have $\sigma = \sigma' \circ \sigma_0$.*

PROOF. See Kfoury, Tiuryn, and Urzyczyn [11]. □

This result is slightly weaker than the corresponding result for unification, in which we would have $\sigma = \sigma' \circ \sigma_0$, without qualification [21].

4. ACYCLIC SEMIUNIFICATION

The subset of SUP known as the *acyclic* semiunification problem (ASUP) was first presented by Kfoury, Tiuryn, and Urzyczyn [12], and is defined as follows:

DEFINITION 6 (LVARS, RVARS). *For an inequality $\tau \leq \mu$, define*

$$\mathrm{LVars}(\tau \leq \mu) = \mathrm{Vars}(\tau)$$
$$\mathrm{RVars}(\tau \leq \mu) = \mathrm{Vars}(\mu).$$

DEFINITION 7 (ACYCLIC). *An instance Γ of SUP is* acyclic *if its inequalities can be arranged into m columns such that the sets $V_0, \ldots, V_m$ defined by*

$$\begin{aligned}
V_0 &= \bigcup_{v \in \text{ col. } 1} \mathrm{LVars}(v) \\
&\cdots \\
V_k &= \left(\bigcup_{v \in \text{ col. } k-1} \mathrm{RVars}(v) \right) \bigcup \left(\bigcup_{v \in \text{ col. } k} \mathrm{LVars}(v) \right) \\
&\cdots \\
V_m &= \bigcup_{v \in \text{ col. } m} \mathrm{RVars}(v)
\end{aligned}$$

are pairwise disjoint.

For example, the instance

$$\{\alpha \leq g(\beta),\ \beta \leq f(\gamma, \gamma),\ f(\alpha, \delta) \leq \epsilon\}\ ,$$

where f is a binary functor and g is a unary functor, is acyclic—if we assign the first and third inequalities to column 1, and the second to column 2, then we have $V_0 = \{\alpha, \delta\}$, $V_1 = \{\beta, \epsilon\}$, and $V_2 = \{\gamma\}$, and these are pairwise disjoint.

DEFINITION 8 (ASUP). *ASUP is the restriction of SUP to acyclic problem instances.*

ASUP is a decidable subset of SUP, and the redex procedure is known to terminate (with either success or failure) on all instances of ASUP. Termination of the redex procedure on ASUP instances forms the basis for a well-known typing algorithm [13].

5. SUP INSTANCES AS GRAPHS

We now introduce a graph-theoretic framework in which to reason about SUP instances, and give a characterization of ASUP within this framework. We first need to establish some terminology:

DEFINITION 9 (UNDIRECTED PATH). *Given a directed graph G and vertices v_1 and v_2 in G, an* undirected path *from v_1 to v_2 is a path from v_1 to v_2, in which we are not required to follow the direction of the edges.*

In other words, an undirected path is a one that exists when we pretend that the underlying directed graph is undirected. Ordinary (directed) paths may also be considered undirected. There are two notions of path length associated with undirected paths on directed graphs:

DEFINITION 10 (UNSIGNED, SIGNED PATH LENGTH). *Given a directed graph G, with an undirected path π joining vertices v_1 and v_2, the* unsigned path length *of π, denoted $||\pi||$, is the number of edges in π. The* signed path length *of π, denoted $|\pi|$, is the length of π, where each forward arrow (i.e., pointing away from v_1 and towards v_2) counts for +1, and each reverse arrow (i.e., pointing towards v_1 and away from v_2) counts for -1.*

The difference between unsigned and signed path length is analogous to the distinction between displacement and distance in physics. We also introduce the following notation:

DEFINITION 11. *For a directed graph G containing vertices v_1 and v_2,we write $v_1 \to v_2$ (resp. $v_1 \to_U v_2$) if there is a directed (resp. undirected) edge from v_1 to v_2. We write $v_1 \to^* v_2$ (resp. $v_1 \to_U^* v_2$) if there is a directed (resp. undirected) path from v_1 to v_2. We write $v_1 \to^+ v_2$ (resp. $v_1 \to_U^+ v_2$) if there is a directed (resp. undirected) path of nonzero length from v_1 to v_2. Finally, we write $\pi : v_1 \to^* v_2$ (and analogously for the other cases) to indicate that π is a directed path from v_1 to v_2.*

We define the *graph* of a SUP instance as follows:

DEFINITION 12 (GRAPH OF A SUP INSTANCE). *Let $\Gamma = \{\tau_i \leq \mu_i\}_{i=1}^N$ be an instance of SUP. Then the* graph of Γ, *denoted $G(\Gamma)$, is defined as follows:*

- *the inequalities $\tau_i \leq \mu_i$ are the vertices v_i in G;*
- *$v_i \to v_j$ iff $\mathrm{RVars}(v_i) \cap \mathrm{LVars}(v_j) \neq \emptyset$*

For example, suppose we have the following SUP instance Γ:

$$\alpha \leq f(\beta, \gamma) \quad \beta \leq \delta \quad \gamma \leq \epsilon \quad \eta \leq \delta \quad \zeta \leq f(\eta, \gamma) \quad g(\delta) \leq \theta\ ,$$

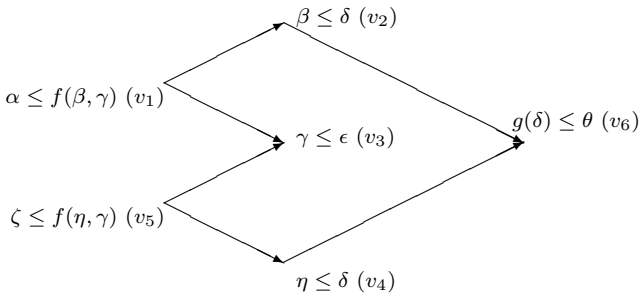

Figure 2: The graph of a SUP instance. The symbols f and g denote, respectively, a binary functor and a unary functor.

where f is a binary functor and g is a unary functor. The graph $G(\Gamma)$ of the instance is given in Figure 2. Let the inequalities in the instance be labelled as vertices v_1, v_2, v_3, v_4, v_5, and v_6, respectively. Then we see that there are directed paths from v_1 to v_3 and v_6, and also from v_5 to v_3 and v_6. On the other hand, there are two[1] *undirected* paths from v_3 to v_6: one going through v_1 and v_2, and the other going through v_5 and v_4. Both have unsigned length equal to 3, and signed length equal to 1.

The following theorem establishes graph-theoretic criteria that are necessary and sufficient for a SUP instance to be an instance of ASUP:

THEOREM 2. *Let $\Gamma = \{\tau_i \leq \mu_i\}_{i=1}^N$ be an instance of SUP. Then Γ is an acyclic instance of SUP iff the following four symmetric conditions hold for $G(\Gamma)$:*

- *for any given variables α_1 and α_2, all paths $\pi : v_1 \rightarrow_U^* v_2$, such that $\alpha_1 \in \mathrm{LVars}(v_1)$ and $\alpha_2 \in \mathrm{LVars}(v_2)$, have the same signed length.*
- *for any given variables α_1 and α_2, all paths $\pi : v_1 \rightarrow_U^* v_2$, such that $\alpha_1 \in \mathrm{LVars}(v_1)$ and $\alpha_2 \in \mathrm{RVars}(v_2)$, have the same signed length.*
- *for any given variables α_1 and α_2, all paths $\pi : v_1 \rightarrow_U^* v_2$, such that $\alpha_1 \in \mathrm{RVars}(v_1)$ and $\alpha_2 \in \mathrm{LVars}(v_2)$, have the same signed length.*
- *for any given variables α_1 and α_2, all paths $\pi : v_1 \rightarrow_U^* v_2$, such that $\alpha_1 \in \mathrm{RVars}(v_1)$ and $\alpha_2 \in \mathrm{RVars}(v_2)$, have the same signed length.*

We begin with a brief outline of the proof, which is somewhat lengthy:

PROOF SKETCH. The forward direction is a proof by induction on $||\pi||$ that $|\pi|$ measures the difference in the column assignments of two inequalities. The reverse direction assigns columns to inequalities according to the constraints provided by the structure of the graph—namely that an inequality must have a column number one less than its successors in the graph and one more than its predecessors. We then show that this procedure and the conditions of the theorem together guarantee the disjointness of the resulting sets of variables. □

[1]In fact there are infinitely many undirected paths between any two distinct connected vertices; but all but finitely many (in this case, all but two) of these will involve backtracking over previously-traversed edges.

The full proof is as follows:

PROOF OF THEOREM 2. We begin with the forward direction. Suppose Γ is an acyclic instance of SUP. Then there is an arrangement of the inequalities in Γ into m columns such that the following sets:

$$V_0 = \bigcup_{v \in \text{col. } 1} \mathrm{LVars}(v)$$

$$V_i = \left(\bigcup_{v \in \text{col. } i} \mathrm{RVars}(v) \right) \bigcup \left(\bigcup_{v \in \text{col. } i+1} \mathrm{LVars}(v) \right)$$

$$V_m = \bigcup_{v \in \text{col. } m} \mathrm{RVars}(v)$$

are pairwise disjoint. Now, consider any edge from $\mu_i \leq \tau_i$ to $\mu_j \leq \tau_j$ in G. Then τ_i and μ_j share at least one variable in common. Thus, by the disjointness of the V_i's $\mu_i \leq \tau_i$ and $\mu_j \leq \tau_j$ must be in adjacent columns, say $\mu_i \leq \tau_i$ is in column k and $\mu_j \leq \tau_j$ is in column $k+1$ (so that the variables in τ_i and μ_j are in the set V_k). Hence the edge points from an inequality in column k to one in column $k+1$. (Since all edges point from a given column to the one immediately following it, it follows immediately that G is acyclic.) Now, let $\mu_1 \leq \tau_1$ and $\mu_2 \leq \tau_2$ be inequalities in Γ, i.e., edges in G, in columns k_1 and k_2, respectively, and suppose $\pi : \mu_1 \leq \tau_1 \rightarrow_U^* \mu_2 \leq \tau_2$ in G. We prove by induction on $||\pi||$ that $|\pi| = k_2 - k_1$. If $||\pi|| = 0$, then the two vertices coincide, and the result is immediate. Otherwise, we can decompose π into a directed (though possibly reversed) path π_1 followed by an undirected path π_2 via a vertex $\mu_3 \leq \tau_3$, in column k_3. Since every edge joins consecutive columns, $|\pi_1|$ must be precisely $k_3 - k_1$. By induction, we claim that $|\pi_2| = k_2 - k_3$. Thus $|\pi| = k_2 - k_1$, independently of our choice of path. Hence, for any pair of vertices in G, all undirected paths joining them have the same signed length. (Note that this implies that all *directed* paths joining two given vertices also have the same length.) Now, choose $i, j \in \{1, 2\}$. By the pairwise disjointness of $V_0, \ldots, V_m$, all inequalities $\tau_{11} \leq \tau_{12}$ such that $\alpha_1 \in \mathrm{Vars}(\tau_{1i})$ are in some column k, and all inequalities $\tau_{21} \leq \tau_{22}$ such that $\alpha_2 \in \mathrm{Vars}(\tau_{2j})$ are in some column k'. Hence all undirected paths joining such vertices must be of signed length precisely $k - k'$. This establishes the forward direction.

For the reverse direction, we begin with an acyclic digraph G satisfying our hypotheses, and arrange the inequalities into columns as follows:

- for each connected component of G:

  - label any vertex v with any integer c
  - while there are unlabelled vertices:
    * choose a labelled vertex w, with label l_w
    * for all unlabelled vertices w' such that $w \to w'$, label w' with label $l_w + 1$
    * for all unlabelled vertices w' such that $w' \to w$, label w' with label $l_w - 1$
- while possible:
  - let G_1 and G_2 be connected components of G, such that there are vertices $v_1 \in G_1$, $v_2 \in G_2$, with respective labels l_1 and l_2, such that $\mathrm{Vars}(v_1) \cap \mathrm{Vars}(v_2) \neq \emptyset$
  - subtract $l_2 - l_1$ from all labels in G_2
  - create a new vertex v_3, with no variables and label $l_1 + 1$, and edges from v_1 to v_3, and from v_2 to v_3, so that G_1 and G_2 are now connected
- let l_0 be the smallest label in G and subtract l_0 from all labels in G
- erase all edges and vertices added to G in the second loop above; each vertex's label is its column

The following observation is immediate: if there is an assignment of the inequalities into columns, such that the V_i's are disjoint, then this algorithm will find it—every choice of label it makes is forced upon it by the edges of the graph, which constrain the possible column assignments. What we must show is that there is always such an assignment. In particular, after the algorithm is finished, if we form the V_i's, will these sets be pairwise disjoint?

First note that the edges of G actually used by the algorithm in assigning labels induce a spanning tree on each connected component of G. So between any two vertices v_1 and v_2 (labelled l_1 and l_2, respectively) within a connected component of G, there is a unique path along the spanning tree that joins them. Moreover the signed length of the path from v_1 to v_2 along the spanning tree is $l_2 - l_1$ (easy induction on path lengths).

Let each vertex's label be its column and form the sets $V_0, \ldots, V_m$. Suppose there are sets V_i and V_j with a variable ϕ such that $\phi \in V_i \cap V_j$. We first assume that the two corresponding occurrences of ϕ lie within the same connected component of G. Then there are four cases, depending on whether ϕ is found on the left-hand sides or the right-hand sides of the inequalities involved. We consider one case in detail here—there are inequalities $\tau_1 \leq \mu_1$ and $\tau_2 \leq \mu_2$, in columns i and j, respectively, such that $\phi \in \mathrm{Vars}(\mu_1) \cap \mathrm{Vars}(\mu_2)$. The signed path length between these two vertices is $j - i$. By hypothesis, all paths between two inequalities having ϕ on the right-hand side must then have this signed length. Consider now the distance from the vertex $\tau_1 \leq \mu_1$ to itself. It must also have value $i - j$, by hypothesis on G, but of course the distance from a vertex to itself is 0. Hence $i - j = 0$, from which we obtain $i = j$. The remaining three cases are similarly easy.

We now suppose that $\tau_1 \leq \mu_1$ and $\tau_2 \leq \mu_2$ lie in *different* connected components of G, so that there is no path joining them. There are then two possibilities:

- $\tau_1 \leq \mu_1$ and $\tau_2 \leq \mu_2$ were the vertices considered in the second part of the algorithm—then they were assigned the same label; hence $i = j$.
- otherwise two vertices v_1 and v_2, with a variable ψ in common, were used by the algorithm to temporarily join the connected components. Say v_1 and $\tau_1 \leq \mu_1$ are in the same connected component, as are v_2 and $\tau_2 \leq \mu_2$. By hypothesis on G, the signed path length from v_1 to $\tau_1 \leq \mu_1$ is equal to the signed path length from v_2 to $\tau_2 \leq \mu_2$. Since the algorithm assigns v_1 and v_2 the same column, it follows again that $i = j$.

□

Looking again at the example SUP instance illustrated in Figure 2, we see that, for example, all undirected paths from $\alpha \leq f(\beta, \gamma)$ to $g(\delta) \leq \theta$ have signed length equal to 2. More generally, all undirected paths from a vertex with, say, β on the right-hand side to a vertex with δ on the left-hand side have signed length equal to 2. Similarly, all undirected paths from a vertex with η on the left-hand side to a vertex with γ on the left-hand side have signed length equal to 0. Analogous properties hold for all pairs of variables, and therefore the SUP instance illustrated in Figure 2 is acyclic.

A few characteristics of the formulation of acyclicity given in Theorem 2 are worth noting. First, the disjointness of the sets $V_0, \ldots, V_m$ is modelled by a condition requiring constancy of path lengths. Second, although the constants mentioned in the four conditions are, of course, related to one another, we still need all four conditions—this is because a given variable might occur only on left-hand sides, or only on right-hand sides. In these cases, not all four constants may exist for a given choice of α_1 and α_2. Finally, although any directed graph satisfying the conditions of the theorem must be acyclic, there is no direct notion of acyclicity mentioned in the theorem. In Section 6, we generalize the condition for acyclicity, while maintaining decidability. The new condition clearly has an acyclic flavour.

6. R-ACYCLICITY

We now define an acyclicity criterion for the graph G corresponding to a SUP instance Γ. We call this criterion *R-acyclicity*; we show that R-acyclicity is sufficient to guarantee termination of the redex algorithm, and is more general than the original, column-based criterion.

DEFINITION 13 (R-ACYCLIC). *For a graph G of a SUP instance $\Gamma = \{\tau_i \leq \mu_i\}_{i=1}^N$, define relations R,*

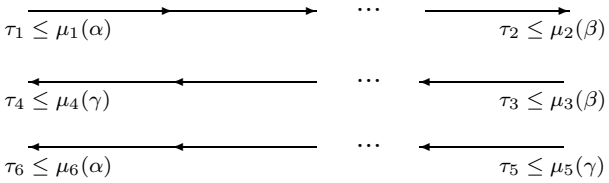

Figure 3: The relation R and R-acyclicity. The notation $\mu(\alpha)$ denotes an expression μ in which α occurs as a subexpression. Here, $\alpha R \beta$—indeed, $\alpha R' \beta$. Since also $\beta R \gamma R \alpha$, we have $\beta R^+ \alpha$; therefore, this graph is not R-acyclic.

R' on variables in G as follows: $\alpha R \beta$ (resp. $\alpha R' \beta$) if there exist vertices v_i and v_j with $\alpha \in \text{RVars}(v_i)$, $\beta \in \text{RVars}(v_j)$, and $v_i \rightarrow^* v_j$ (resp. $v_i \rightarrow^+ v_j$). G is said to be R-acyclic *if whenever $\alpha_i R' \alpha_j$, we have $\neg(\alpha_j R^+ \alpha_i)$, where R^+ is the transitive closure of the relation R.*

The "R" in R-acyclic refers, of course, to the relation R in Definition 13. However, it also highlights the asymmetry in the definition between RVars and LVars—in particular, that we impose conditions on RVars, but not on LVars. Hence, "R-acyclic" may be read as "right-acyclic".

Although the statement of R-acyclicity is somewhat involved, R-acyclicity is not itself difficult to understand. For illustrative purposes, Figure 3 depicts a graph that is not R-acyclic. Note that R-acyclicity implies graph acyclicity in the ordinary sense.

We wish to show that the redex procedure will terminate on R-acyclic instances. Our proof hinges on the observation that redex reduction preserves R-acyclicity:

THEOREM 3 (INVARIANCE OF R-ACYCLICITY). *Let Γ be a SUP instance, and Γ' be a SUP instance obtained by reducing a redex in Γ. If $G(\Gamma)$ is R-acyclic, then so is $G(\Gamma')$.*

PROOF. Suppose a redex-I is reduced in Γ. Then all occurrences of some variable α are replaced with some expression τ, containing only fresh variables. Hence all vertices that contained α now contain the variables (if any) of τ, and no other vertices contain these variables because they are all fresh—therefore no edges are created by this reduction. Hence, no R-cycles can be created, and G remains R-acyclic.

Suppose now that a redex-II is reduced in Γ. If reduction causes G to lose R-acyclicity, then there are two possibilities:

- there is a variable replacement $[\tau/\alpha]$ that occurs during reduction, which induces, for some $\beta_1, \ldots \beta_n$, the relations $\beta_1 R \cdots R \beta_n$, and such that $\beta_n R' \beta_1$. Since $[\tau/\alpha]$ caused the violation, it created an edge that completed one of the paths from β_i to $\beta_{(i \bmod n)+1}$. For such an i, there is an edge from some $v_j \rightarrow v_k$ lying along this path, that was created by the substitution $[\tau/\alpha]$. Hence, one of RVars(v_j) and LVars(v_k) contains the variable α; the other contains a variable from τ, say γ. Now, for the redex $[\tau/\alpha]$ to exist, there must exist an inequality v_h that satisfies the conditions for this redex-II; hence α and τ are both in RVars(v_h). Since one of α and γ is in LVars(v_k), we have $v_h \rightarrow v_k$. Since either α or γ is in RVars(v_j), and both are in RVars(v_h), the transitive closure of R connects the path ending with v_j to the path beginning with v_h, and followed by v_k. Hence, the removal of the edge from v_j to $\tau_k \leq \mu_k$ does not restore R-acyclicity. Thus, removing edges introduced by redex-II reductions cannot convert graphs that are not R-acyclic to graphs that are.
- we had $\beta_i R \cdots R \beta_j$ and $\beta_k R \cdots R \beta_l$, for some β_i, β_j, β_k, and β_l, and the redex reduction unifies β_j and β_k, thus linking the two R-chains. In this case, if a redex-II reduction unifies β_j and β_k, then thse two variables must occur together on the right-hand side of some inequality (the one in which the redex occurs). Hence $\beta_j R \beta_k$, and we already had $\beta_i R \cdots R \beta_j R \beta_k R \cdots R \beta_l$, i.e., the two R-chains were already linked. Again, the redex-II reduction can only result in a non-R-acyclic instance if the instance was non-R-acyclic to begin with.

In both cases, we see that redex-II reduction cannot reduce a graph that is R-acyclic to one that is not. In summary, then, the redex procedure preserves R-acyclicity. □

COROLLARY 1. *Let α and β be variables in an R-ASUP instance Γ, with $\alpha R' \beta$. Then no reduction σ of Γ will produce $(\beta\sigma) R (\alpha\sigma)$.*

PROOF. Consider the instance

$$\Gamma' := \Gamma \cup \{x_1 \leq f(\alpha, x_2),\ x_2 \leq \beta\} ,$$

where f is a binary functor, and x_1 and x_2 are fresh variables. Then this extra inequality gives us $\alpha R' \beta$, which we already had, and $x_2 R' \beta$, which is of no consequence because x_2 does not occur anywhere else. Therefore, this instance is R-acyclic iff Γ is. If Γ reduces such that we obtain $(\beta\sigma) R (\alpha\sigma)$, then in Γ', we have $(\alpha\sigma) R' (\beta\sigma) R (\alpha\sigma)$ (because no reduction is going to affect the two extra inequalities). Hence Γ' is non-R-acyclic. But this contradicts the invariance of R-acyclicity. Therefore, Γ cannot reduce so as to produce $(\beta\sigma) R (\alpha\sigma)$.

Note that this argument presumes the existence of at least one binary functor, f. But without a binary functor, there can be no redex-II's, and redex-I reduction cannot create edges. Thus, the result follows either way. □

COROLLARY 2. *Let v_1 and v_2 be vertices in the graph of an R-ASUP instance Γ, such that v_1 precedes v_2 in the partial order induced by the graph. Suppose that after k iterations of the redex procedure (i.e., after reduction of k redices), Γ reduces to an instance Γ_k. Let σ_k be the substitution that converts Γ to Γ_k (i.e., σ_k is the accumulated substitution encapsulating the k redex reductions). Then $v_2\sigma_k$ cannot precede $v_1\sigma_k$ in the graph of Γ_k.*

PROOF. Let $v_1 = \tau_1 \leq \mu_1$, $v_2 = \tau_2 \leq \mu_2$. Let $\alpha \in \text{Vars}(\mu_1)$, $\beta \in \text{Vars}(\mu_2)$. If v_1 precedes v_2 in the partial order induced by the graph, then we have $\alpha R' \beta$. If, after reduction, the graph has v_2 preceding v_1, then we would have $\beta R' \alpha$, contradicting the previous claim.

This argument presumes that μ_1 and μ_2 each contain at least one variable. We know that μ_1 must contain a variable; otherwise v_1 could not precede anything (it would have no out-edges). Further if μ_2 had no variables, then no reduction could make v_2 precede anything. Hence, the case where either inequality contains no variables on the right-hand side poses no difficulty. □

The following result, establishing the solvability, via the redex procedure, of singleton instances of SUP, will ultimately form the base case of our main result:

LEMMA 1. *Every instance of SUP comprising a single inequality $\tau \leq \mu$, with $\text{Vars}(\tau) \cap \text{Vars}(\mu) = \emptyset$, is solvable by the redex algorithm (that is, the redex algorithm will terminate on such an input).*

PROOF. We bound the number of redex reductions that can be performed in $\tau \leq \mu$:

- *The number of redex-I reductions in $\tau \leq \mu$ is bounded by the number of leaf nodes in τ (i.e., by the number of variable occurrences in τ).* Every redex-I reduction causes at least one variable α in τ to be matched against a variable in μ. No further reduction will ever again cause this occurrence of α to be part of a redex-I. Hence there can be no more redex-I's than leaves in τ. (Note that, because $\text{Vars}(\tau) \cap \text{Vars}(\mu) = \emptyset$, redex reduction does not change τ.)

- *The number of redex-II reductions that can occur in $\tau \leq \mu$ before a redex-I reduction must occur is bounded by* $|\text{Vars}(\mu)|$. This is because each redex-II reduction replaces at least one variable in μ; hence it decreases $|\text{Vars}(\mu)|$ by at least 1.

Since the number of redex-II reductions that can occur between redex-I reductions is bounded, and the total number of redex-I reductions is bounded, the redex algorithm must eventually terminate. □

We now prove the main result. The inductive step of the proof relies on the fact that every directed acyclic graph G creates a partial order $\sqsubseteq_G$ on its vertices, defined such that $v_1 \sqsubseteq_G v_2$ if there is a directed path from v_1 to v_2. The minimal elements in $\sqsubseteq_G$ are the source vertices, and the maximal elements are the sink vertices. Further, every directed acyclic graph has at least one source vertex and at least one sink vertex. Hence also, the relation $\sqsubseteq_G$ has at least one minimal element and at least one maximal element.

This theorem drives the induction in the main result, which we present below:

THEOREM 4 (TERMINATION FOR R-ASUP). *Let Γ be an instance of SUP and $G = G(\Gamma)$. If G is R-acyclic, then the redex algorithm will terminate on Γ.*

PROOF. For each i, let Γ_i be the result of performing i reductions on the instance Γ, according to the redex procedure, and let $G_i = G(\Gamma_i)$. With each G_i is associated a partial order $\sqsubseteq_{G_i}$, induced by its edges, as described above. By Corollary 2 of Theorem 3, if, for vertices v_x and v_y, we have $v_x \sqsubseteq_{G_i} v_y$ for some i, then there is no j such that $v_y \sqsubseteq_{G_j} v_x$. Therefore the union of all of the partial orders, namely,

$$\sqsubseteq := \bigcup_i \sqsubseteq_{G_i} ,$$

is a partial order, respecting all of the partial orders associated with all reduced intances Γ_i. Let $\leq$ be any total order of the vertices of Γ consistent with $\sqsubseteq$, and number the vertices in Γ according to this order. Then any reduction of a redex in a vertex v_i can only induce redices in vertices v_j for $i \leq j$. For each vertex v_i let n_i be the maximum number of redices that can be reduced in v_i before it (considered in isolation) is solved (this number is finite, by Lemma 1). We then proceed by induction on the ordinal $(n_1, \ldots, n_N)$, under lexicographic ordering, which is a well-ordering of the N-tuple. Since reduction of a redex in any v_i reduces n_i, and can only increase n_j for $j > i$, and since the instance is solved when the ordinal is $(0, \ldots, 0)$, the result follows by induction. □

COROLLARY 3. *The set of SUP instances that have R-acyclic graphs forms a decidable subset of SUP.*

DEFINITION 14 (R-ASUP). *R-ASUP is the restriction of SUP to R-acyclic problem instances.*

The previous theorem establishes R-ASUP as a decidable subset of SUP, and moreover, one for which the redex procedure is a full solution procedure (that is, it is guaranteed to terminate on instances with no solution). It remains to establish the relationship between R-ASUP and the original ASUP.

THEOREM 5. *R-ASUP is a strict superset of ASUP.*

PROOF. For variables α and β, if $\alpha R'\beta$ (where R' is as given in Definition 13), then α's column assignment is strictly smaller than β's. If also $\beta R^+\alpha$, then β's column assignment would be less than or equal to α's, which is a contradiction. Hence $\neg(\beta R^+\alpha)$, and therefore any instance that satisfies the column-based definition of acyclicity is R-acyclic. On the other hand, consider any instance containing the following inequalities:

$$\begin{aligned} \alpha &\leq \beta \\ \beta &\leq \gamma \\ \alpha &\leq \gamma \end{aligned}$$

This instance does not satisfy the column-based criterion for acyclicity—for suppose that $\alpha \in V_i$. Then by the second inequality, $\gamma \in V_{i+2}$, but by the third inequality, $\gamma \in V_{i+1}$. On the other hand, it is easy to see that these inequalities are R-acyclic. Hence, indeed, R-acyclicity is strictly more general than the column-based criterion. □

7. VARIABLE ACYCLICITY

The SUP instance used in the proof of the preceding theorem suggests a simpler graph-based formulation and acyclicity condition, which we might call *variable acyclicity*. In particular, we form a graph whose vertices are the variables of the instance, and for variables α and β, we have $\alpha \to \beta$ iff there is an inequality $\tau \leq \mu$ with $\alpha \in \text{Vars}(\tau)$ and $\beta \in \text{Vars}(\mu)$. Then an instance is called variable acyclic if the resulting graph, formed in this way, is an acyclic graph. Such an acyclic graph yields a partial ordering on the variables in the instance, and strongly suggests that if the instance is solved according to a strategy in which variables are substituted in an order respecting this partial order, then the redex procedure will terminate. For example, if we replace each "$\leq$" in $\{\alpha \leq \beta,\ \beta \leq \gamma,\ \alpha \leq \gamma\}$ with an edge pointing to the right, and group all occurrences of the same variable together as a single vertex, then we obtain a simple acyclic graph, and indeed, the instance is a terminating SUP instance.

It turns out, however, that this intuition is false in general. It is possible to construct terminating SUP instances with cyclic variable graphs (for example, $\{\alpha \leq \beta, \beta \leq \alpha\}$); more importantly, it is possible to construct non-terminating SUP instances with acyclic variable graphs. Consider, for example, the following instance:

$$\begin{aligned} f(\alpha,\alpha) &\leq f(\beta, f(\gamma,\gamma)) \\ \beta &\leq \gamma\ , \end{aligned}$$

where f is a binary functor. The variable acyclicity graph of this instance is presented in Figure 4—note that it is acyclic. Interestingly, the graph is identical to that of the instance $\{\alpha \leq \beta,\ \beta \leq \gamma,\ \alpha \leq \gamma\}$ mentioned above. Applying the redex procedure to the instance, we see that there is a redex-II in the first inequality mapping β to $f(\gamma,\gamma)$. As a result, the second inequality becomes $f(\gamma,\gamma) \leq \gamma$, which then produces an infinite sequence of redex-I's.

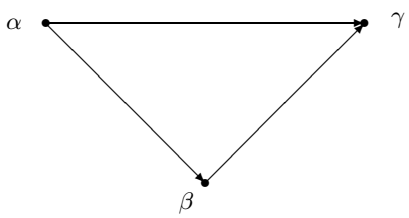

Figure 4: Variable acyclicity graph of the instance $\{f(\alpha,\alpha) \leq f(\beta, f(\gamma,\gamma)),\ \beta \leq \gamma\}$.

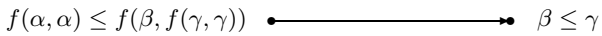

Figure 5: R-acyclicity graph of the instance $\{f(\alpha,\alpha) \leq f(\beta, f(\gamma,\gamma)),\ \beta \leq \gamma\}$.

On the other hand, from the perspective of R-acyclicity, the instance has the graph presented in Figure 5. We now see that $\beta R'\gamma$ because there is a path of length 1 (the only nonempty path in the graph, as it happens) in which β occurs on the right-hand side at the beginning and γ occurs on the right-hand side at the end. But we also have $\gamma R\beta$ because there is a path of length 0 (namely, the first inequality) in which γ occurs on the right-hand side at the beginning and β occurs on the right-hand side at the end (for a path of length 0, this simply means that they co-occur on a right-hand side). Since $\beta R'\gamma R\beta$, the graph is not R-acyclic and we reject the instance.

As this case study hopefully illustrates, SUP is deceptively subtle and intuition often goes awry when attempting to reason about it. Indeed, the authors went through several iterations of incorrect candidate invariants for the redex procedure before settling upon R-acyclicity. The fact that SUP had long been thought decidable is further evidence of the subtle nature of the problem.

8. A MOTIVATING APPLICATION

Although we present R-ASUP as a problem of general interest with varied application domains, our interest in R-ASUP arose from a study of ASUP within the realm of polymorphic type inference algorithms. Here, a typability procedure for terms in the rank 2 fragment of System F is phrased in terms of a reduction to ASUP [13]. However, the reduction is somewhat unnatural. Consider a term of the form

$$\lambda x_1.\cdots.\lambda x_m.(\lambda y_1.(\cdots(\lambda y_n.M_{n+1})M_n)\cdots)M_1\ ,$$

where variables are assumed to be named distinctly, and no M_k contains a λ-abstraction paired with an argu-

ment. The translation to ASUP of this term introduces

- one variable for each subexpression of each M_k;
- one variable for each (x_i, M_k) pair;
- one variable for each (y_j, M_k) pair for $k > j$;
- one variable for each (w_l, M_k) pair, where w_l is a free variable;
- one variable for each bound variable z not mentioned above.

Many of these variables are introduced only in association with trivial single-variable inequalities (i.e., of the form $\alpha \leq \beta$), in order to make the column assignments of the inequalities conform to the requirements of ASUP.

By working within R-ASUP rather ASUP, we are able to produce an equivalent translation [15] that uses only

- one variable for each subexpression of each M_k;
- one variable for each x_i;
- one variable for each y_j;
- one variable for each w_l, where w_l is a free variable;
- one variable for each bound variable z not mentioned above.

In other words, our R-ASUP-based translation uses quadratically fewer variables, introduces quadratically fewer inequalities, and is a more natural reflection of the original term—this second aspect of R-ASUP aided the authors' own understanding of the application domain; by facilitating a more natural and elegant translation to SUP, R-ASUP facilitates learning and aids reasoning about the application. We expect that effects similar to this may arise in other application domains as well.

9. RELATED WORK

Over the years, there have been several unsuccessful attempts to give a full solution procedure for SUP, many of which have yielded decidable subsets of varying complexity. We outline some of these here.

Henglein [5, 6, 7] did pioneering work on semiunification, establishing links between SUP and the type systems of languages like the Milner-Mycroft calculus [17]. As part of his work, he provided a solution procedure for the *linear* semiunification problem, in which all functors have arity one, and conjectured general solvability. Under the assumption that all functors are unary, an instance can have no redex-II's, and therefore ordinary graph acyclicity (rather than R-acyclicity) is sufficient to guarantee termination of the redex procedure.

Baaz [1] gave a semi-procedure for general SUP problems that is based on a reduction to ordinary unification. Baaz's algorithm is less direct than the redex procedure, relying on variable renamings and appeals to unification, rather than performing any explicit term substitutions. A study of the behaviour of Baaz's algorithm on instances of R-ASUP is beyond the scope of this work.

Kapur et al [9] showed that SUP is decidable in polynomial time when restricted to instances containing a single inequality (this is called *uniform* semiunification). Oliart and Snyder [18] give a solution procedure that runs in $O(n^2\alpha(n)^2)$ time in general, $O(n^2 \log^2(n\alpha(n))\alpha(n)^2)$ time if principal unifiers are required, where α is the inverse Ackermann function. In the case of a single inequality, the only possible non-zero path in the instance's graph is a self-loop, which only arises if a variable occurs on both sides of the lone inequality. Since a self-loop is a cycle, both ASUP and R-ASUP prohibit this possibility. SUP is known to be undecidable as soon as the number of inequalities in the instance is at least 2 [9, 11, 20].

Left-linear semiunification restricts the problem instance such that within each left-hand side, no variable occurs more than once. Left-linear semiunification was introduced and shown decidable by Kfoury, Tiuryn, and Urzyczyn [10]. Henglein [8] gives a cubic time solution procedure. A left-linear instance cannot contain redex-II's, and therefore, as with linear semiunification, ordinary graph acylicity suffices to guarantee termination.

Leiss [14] showed that semiunification is decidable when restricted to two variables. Strictly speaking, this subset, like the others presented in this section, is neither a superset nor a subset of R-ASUP; nevertheless, it seems clear that R-ASUP is the largest and most significant decidable subset of SUP among all of these.

10. CONCLUSION

This paper extends the class of known solvable instances of SUP by replacing the column-based formulation of acyclicity by R-acyclicity. R-acyclicity enjoys several advantages over the original formulation:

- it eliminates the need for constancy of path lengths in an intance's graph;
- it replaces four conditions with a single condition, by eliminating explicit consideration of variables on the left-hand side;
- the relationship between R-acyclicity and the relation R clearly shows the acyclic character of this subset of SUP; the notion of acyclicity is not as apparent in the column-based formulation;
- by relaxing several of the conditions originally imposed on SUP instances, we have a simpler, more natural restriction on SUP that is more widely applicable than the original formulation of acycliciy;

hence the class of known solvable instances of SUP is now increased.

R-ASUP has proved to be of value in the application domain in which its predecessor, ASUP, was formulated. As we observed in Section 8, *R*-ASUP is a more natural fit than ASUP in this domain, leading to a more concise translation from typability instances to SUP. In other domains that make use of subsets of SUP, *R*-ASUP may prove to be of similar benefit.

11. REFERENCES

[1] Matthias Baaz. Note on the existence of most general unifiers. *Arithmetic, Proof Theory, and Logical Complexity*, pages 20–29, 1993.

[2] Pascal Brisset. Avoiding dynamic type checking in a polymorphic logic programming language. In *Symposium on Logic Programming*, page 674, 1994.

[3] Jochen Dörre and William C. Rounds. On subsumption and semiunification in feature algebras. *Journal of Symbolic Computation*, 13:441–461, 1992.

[4] Manuel Fähndrich, Jakob Rehof, and Manuvir Das. Scalable context-sensidive flow analysis using instantiation constraints. In *ACM SIGPLAN conference on Programming Language Design and Implementation*, pages 253–263. ACM Press, 2000.

[5] Friedrich Henglein. *Polymorphic Type Inference and Semi-Unification*. PhD thesis, Rutgers, New Brunswick, New Jersey, May 1989.

[6] Fritz Henglein. Semi-unification. Technical Report (SETL Newsletter) 223, New York University, April 1988.

[7] Fritz Henglein. Type inference and semi-unification. In *Proceedings of the 1988 ACM conference on LISP and functional programming*, pages 184–197. Association for Computing Machinery, ACM Press, 1988.

[8] Fritz Henglein. Fast left-linear semi-unification. In *ICCI'90: Proceedings of the international conference on Advances in computing and information*, pages 82–91, New York, NY, USA, 1990. Springer-Verlag New York, Inc.

[9] D. Kapur, D. Musser, P. Narendran, and J. Stillman. Semi-unification. In *Proceedings of the 8th Conference on Foundations of Software Technology and Theoretical Computer Science*, volume 338 of *LNCS*, pages 435–454, Berlin/New York, 1988. Springer-Verlag.

[10] A. Kfoury, J. Tiuryn, and P. Urzyczyn. Computational consequences and partial solutions of a generalized unification problem. In *Proceedings of the 4th IEEE Symposium on Logic in Computer Science (LICS)*, jun 1989.

[11] A. J. Kfoury, J. Tiuryn, and P. Urzyczyn. The undecidability of the semi-unification problem. *Information and Computation*, 102:83–101, 1993.

[12] A. J. Kfoury, J. Tiuryn, and P. Urzyczyn. An analysis of ML typability. *Journal of the Association for Computing Machinery*, 41(2):368–398, March 1994.

[13] A. J. Kfoury and J. B. Wells. A direct algorithm for type inference in the rank-2 fragment of the second-order λ-calculus. In *1994 ACM Conference on LISP and Functional Programming*, pages 196–207. ACM Press, 1994.

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

[15] B. Lushman and G. V. Cormack. A *m*ore direct algorithm for type inference in the rank-2 fragment of the second-order λ-calculus. Technical Report CS-2006-08, U. of Waterloo, 2006. Available at http://www.cs.uwaterloo.ca/research/tr/2006/CS-2006-08.pdf.

[16] A. Martelli and U. Montanari. An efficient unification algorithm. *ACM Transactions on Programming Languages and Systems*, 4:258–282, 1982.

[17] A. Mycroft. Polymorphic type schemes and recursive definitions. In Paul and Robinet, editors, *International Symposium on Programming*, pages 217–228, 1984.

[18] Alberto Oliart and Wayne Snyder. A fast algorithm for uniform semi-unification. In *15th Conference on Automated Deduction (CADE-15)*, number 1421 in LNAI. Springer-Verlag, 1998.

[19] M. Paterson and M. Wegman. Linear unification. *J. Computer and System Sciences*, 16:158–167, 1978.

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

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

[22] J. B. Wells. Typability and type checking in System F are equivalent and undecidable. *Annals of Pure and Applied Logic*, 98(1–3):111–156, 1999.