The R-Acyclic Semiunification Problem

Brad Lushman Gordon V. Cormack

March 13, 2006

Abstract

We recast Kfoury and Wells' formulation of the acyclic semiunification problem (ASUP) in graph-theoretic terms and prove equivalence between the two formulations. We then relax and simplify the graph-theoretic formulation; we call the resulting problem the R-acyclic semiunification problem (R-ASUP), which we show to be a strict superset of ASUP. We prove that the ASUP solution procedure terminates and produces most general solutions for R-ASUP (and hence for ASUP) in the same sense as Robinson's unification algorithm. We thus extend the class of semiunification instances known to be decidable.

1 Introduction

Given an algebra of terms, consisting of variables, along with a set of functors with associated arities, *unification* [6] is the problem of finding, for a set of pairs of terms, a variable assignment that equates each pair in the set[1]. The unification problem is well studied; linear time solutions are known. A related problem, the *semiunification problem* (SUP), arises in settings related to polymorphic type assignment in programming languages. Semiunification differs from unification in that a variable assignment that solves the problem instance need not equate pairs of terms; it is enough that, for each pair, a chosen term (called the "right-hand side") be a substitution instance of the other term (called the "left-hand side").

Although SUP is now known to be undecidable [3], a restriction of SUP, known as the *acyclic semiunification problem* (ASUP), has a solution procedure [4]. The standard description of ASUP is based on assigning the pairs (called "inequalities") in a SUP instance into columns, subject to certain constraints. This description is unintuitive, difficult to communicate, and does not clearly reveal the "acyclicity" inherent in this restriction of SUP.

This paper uses a graph-theoretic setting to describe ASUP; we prove equivalence between our graph-theoretic description and the column-based description. We then present a new graph-theoretic restriction on SUP that is both easier to understand (it clearly reveals the "acyclic" nature of the problem description) and strictly more general than the first formulation. Along the way, we present the first archival proof that the ASUP solution procedure is correct, confluent, and produces "most general" results in the same sense as Robinson's original unification algorithm.

2 Background and Related Work

Unification abounds in computer science; it lies at the heart of many systems for formal manipulation of symbols and formulas. Such systems have applications in theorem proving, software engineering, and compiler construction; a survey can be found in [5]. Unification algorithms solve the problem of term equality in a given symbolic framework; a solution is an assignment of terms to variables such that performing the assignment equates pairs of given terms.

Semiunification [1] is a related problem that works in the domain of term *inequalities*. It arises frequently, particularly in the context of polymorphic type inference. Semiunification has been used to study polymorphic recursion in the Milner-Mycroft calculus [2], and general parametric polymorphism in the context of System F [7]. In both cases, the undecidability of semiunification led to undecidability results for the respective typability problems. The semiunification problem, SUP, is defined below:

Definition 1 (SUP). *An instance of SUP is a set $\{\tau_i \leq \mu_i\}$ of inequalities in some algebra consisting of variables and some set of functors. A substitution S is a solution of SUP if there exist substitutions $S_1, \ldots, S_N$ such that*

$$\tau_1 S S_1 \quad = \quad \mu_1 S$$

[1] Some formulations demand only a variable assignment that equates a single pair of terms; the two formulations are equivalent, as long as at least one functor has arity greater than 1.

$$\cdots$$
$$\tau_N S S_N \quad = \quad \mu_N S$$

Although SUP is itself undecidable [3], a particular subset, known as the *acyclic* semiunification problem (ASUP) is decidable [4]:

Definition 2 (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 3 (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.

Definition 4 (ASUP). *ASUP is the restriction of SUP to acyclic problem instances.*

For problems that have been shown to be undecidable via equivalence with SUP, ASUP provides a means of extracting decidable subproblems. For example, Kfoury and Wells [4] observed that typability in an important sublanguage of System F is, in fact, reducible to ASUP, and therefore decidable.

The original unification algorithm is due to Robinson [6]. The formulation of Robinson's algorithm that we present here relies on *paths*:

Definition 5 (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:

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

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

We phrase Robinson's algorithm as follows:

Input: set $\Gamma = \{\tau_i = \mu_i\}_{i=1}^N$ of term equations

1. Set $\sigma_0 = []$ and $k = 0$; go to step 2.
2. If $\tau_i \sigma_k = \mu_i \sigma_k$ for all i, set $\sigma_\Gamma = \sigma_k$ and terminate with success.
3. Let Σ be a path and $1 \leq i \leq N$ be such that $\Sigma(\tau_i \sigma_k) \neq \Sigma(\mu_i \sigma_k)$ and at least one of $\Sigma(\tau_i \sigma_k)$ and $\Sigma(\mu_i \sigma_k)$ is a variable (say $\Sigma(\tau_i \sigma_k)$ is a variable)—if this is impossible, then there is a functor mismatch and Γ is not unifiable—terminate with failure. If $\Sigma(\tau_i \sigma_k)$ does not occur in $\Sigma(\mu_i \sigma_k)$, then set $\sigma_{k+1} = [\Sigma(\mu_i \sigma_k)/\Sigma(\tau_i \sigma_k)] \circ \sigma_k$, add 1 to k, and go to step 2; otherwise terminate with failure.

The ASUP solution procedure, which we call simply the redex algorithm, is due to Kfoury and Wells [4]. We present it below:

Input: set $\Gamma = \{\tau_i \leq \mu_i\}_{i=1}^N$ of term inequalities, such that for each i, $\mathrm{Vars}(\tau_i) \cap \mathrm{Vars}(\mu_i) = \emptyset$.

1. Set $\sigma_0 = []$ and $k = 0$; go to step 2.
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. Add 1 to k, and go to step 2.
   (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 $\sigma_{k+1} = \theta \circ \sigma_k$. Add 1 to k, and go to step 2.
   (c) If neither of steps 3a and 3b is possible, then there is a functor mismatch; terminate with failure.

An important property of Robinson's algorithm is that it always outputs a *most general unifier*. In other words, if Robinson's algorithm outputs a solution σ of the problem instance, then for any other solution θ of the instance, there is a variable substitution λ such that $\theta = \lambda \circ \sigma$. In Section 4, we prove the corresponding result for the redex algorithm. To the best of our knowledge, no previous proof of this result occurs in the literature.

3 A Graph-Theoretic Formulation of ASUP

In this section, we rephrase the acyclicity condition that distinguishes ASUP from SUP in the language of graph theory. We state an equivalence between the original formulation and our graph-thoretic formulation. In later sections, we show how the graph-theoretic model can be generalized without sacrificing decidability.

Our discussion relies on the concepts of directed and undirected paths for a directed graph, and of signed and unsigned path length. We define these below.

Definition 6 (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 it is a path where we pretend that G is undirected.)*

Definition 7 (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.*

We also introduce the following notation:

Definition 8. *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.*

The following theorem provides a graph-theoretic characterization of the column-based definition of acyclicity, as it appears in Definition 3.

Theorem 1. *Let $\Gamma = \{\mu_i \le \tau_i\}_{i=1}^n$ be an instance of SUP. Form a directed graph G as follows:*

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

Then Γ is an acyclic instance of SUP iff the following four symmetric conditions hold for G:

- *for any given variables α_1 and α_2, whenever $\pi : v_1 \to_U^* v_2$, such that $\alpha_1 \in \mathrm{LVars}(v_1)$ and $\alpha_2 \in \mathrm{LVars}(v_2)$, $|\pi|$ is constant.*
- *for any given variables α_1 and α_2, whenever $\pi : v_1 \to_U^* v_2$, such that $\alpha_1 \in \mathrm{LVars}(v_1)$ and $\alpha_2 \in \mathrm{RVars}(v_2)$, $|\pi|$ is constant.*
- *for any given variables α_1 and α_2, whenever $\pi : v_1 \to_U^* v_2$, such that $\alpha_1 \in \mathrm{RVars}(v_1)$ and $\alpha_2 \in \mathrm{LVars}(v_2)$, $|\pi|$ is constant.*
- *for any given variables α_1 and α_2, whenever $\pi : v_1 \to_U^* v_2$, such that $\alpha_1 \in \mathrm{RVars}(v_1)$ and $\alpha_2 \in \mathrm{RVars}(v_2)$, $|\pi|$ is constant.*

We leave a full proof to the appendix.

A few characteristics of this formulation 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 lefthand sides, or only on righthand 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 5, we generalize the condition for acyclicity, while maintaining decidability. The new condition clearly has an acyclic flavour.

4 "Most General"

In this section, we prove that, analogous to Robinson's algorithm, the redex algorithm outputs most general semiunifiers; to our knowledge, a previous proof of this result does not appear in the literature. As an important corollary, we prove that the redex algorithm is confluent; in other words, the order in which we apply the reductions outlined in the algorithm does not affect the final outcome beyond

renaming substitutions. Confluence is an important building block for our generalization of acyclicity in Section 5.

We begin by presenting a version of the original proof that Robinson's Unification Algorithm computes most general unifiers. The corresponding proof for semiunification is of a similar flavour.

Theorem 2. *If Γ is unifiable, then Robinson's algorithm will terminate with success and return a unifier σ_Γ, with the property that if θ is any unifier of Γ, then there is a substitution λ such that $\theta = \lambda \circ \sigma_\Gamma$.*

Proof. The idea is to prove that if Γ is unifiable, then the algorithm will terminate with success, and that the following statement is true for $k \geq 0$ until termination: there is a substitution λ_k such that $\theta = \lambda_k \circ \sigma_k$. If $k = 0$, then $\sigma_k = []$ and we can take $\lambda_k = \theta$. For a general k, suppose $\theta = \lambda_k \circ \sigma_k$ for some λ_k. If σ_k unifies Γ, then we are done, and we just take $\sigma_\Gamma = \sigma_k$. Otherwise, we perform step 3. Since λ_k unifies $\Gamma\sigma_k$, it also unifies $\Sigma(\tau_i\sigma_k)$ and $\Sigma(\mu_i\sigma_k)$, i.e., $\Sigma(\tau_i\sigma_k)\lambda_k = \Sigma(\mu_i\sigma_k)\lambda_k$. Now, if $\Sigma(\tau_i\sigma_k)$ (which is a variable) occurs in $\Sigma(\mu_i\sigma_k)$, then $\Sigma(\tau_i\sigma_k)\lambda_k$ occurs in $\Sigma(\mu_i\sigma_k)\lambda_k$, which is impossible since these expressions are equal. Hence, this step of the algorithm does not terminate with failure; rather, we set $\sigma_{k+1} = [\Sigma(\mu_i\sigma_k)/\Sigma(\tau_i\sigma_k)] \circ \sigma_k$. Let λ_{k+1} be the restriction of λ_k to variables other than $\Sigma(\tau_i\sigma_k)$. Then we have

$$\begin{aligned}
\lambda_k &= \lambda_{k+1} \cup [\Sigma(\tau_i\sigma_k)\lambda_k/\Sigma(\tau_i\sigma_k)] \\
&= \lambda_{k+1} \cup [\Sigma(\mu_i\sigma_k)\lambda_k/\Sigma(\tau_i\sigma_k)] \\
&= \lambda_{k+1} \cup [\Sigma(\mu_i\sigma_k)\lambda_{k+1}/\Sigma(\tau_i\sigma_k)] \\
&= \lambda_{k+1} \circ [\Sigma(\mu_i\sigma_k)/\Sigma(\tau_i\sigma_k)],
\end{aligned}$$

where the third equality follows because $\Sigma(\tau_i\sigma_k)$ does not occur in $\Sigma(\mu_i\sigma_k)$. Thus, $\theta = \lambda_k \circ \sigma_k = \lambda_{k+1} \circ [\Sigma(\mu_i\sigma_k)/\Sigma(\tau_i\sigma_k)] \circ \sigma_k = \lambda_{k+1} \circ \sigma_{k+1}$. The result now follows by induction. □

The proof of the same result for semiunification is slightly more difficult. We begin with the following lemma, which characterizes the structure of solvable SUP instances that are not yet solved:

Lemma 1. *Let τ and μ be expressions with no functor mismatches, such that* $\mathrm{Vars}(\tau)$ *and* $\mathrm{Vars}(\mu)$ *are disjoint, and suppose that μ is not a substitution instance of τ. Then at least one of the following is true:*

- *there is a path Σ such that $\Sigma(\mu)$ is a variable and $\Sigma(\tau)$ is not a variable;*
- *there are paths Σ_1 and Σ_2 and a variable α such that $\Sigma_1(\tau) = \Sigma_2(\tau) = \alpha$ and $\Sigma_1(\mu) \neq \Sigma_2(\mu)$.*

Proof. Choose Σ as large as possible such that $\Sigma(\mu)$ is not a substitution instance of $\Sigma(\tau)$. In particular, choose Σ such that there is no path Σ', of which Σ is a prefix, such that $\Sigma'(\mu)$ is not a substitution instance of $\Sigma'(\tau)$. Two cases arise:

- $\Sigma(\mu)$ is a variable. Then $\Sigma(\tau)$ cannot be a variable (otherwise a renaming (or identity) substitution would map $\Sigma(\tau)$ onto $\Sigma(\mu)$), so the result follows.
- $\Sigma(\mu)$ is not a variable. Then $\Sigma(\mu) = f(\mu_1, \ldots, \mu_n)$. Since $\Sigma(\tau)$ cannot be a variable (otherwise, $\Sigma(\mu)$ is obviously a substitution instance of $\Sigma(\tau)$), and there are no functor mismatches, we have $\Sigma(\tau) = f(\tau_1, \ldots, \tau_n)$. By the maximality of Σ, each μ_i is a substitution instance of the corresponding τ_i, i.e., for each i, there is a substitution σ_i such that $\tau_i\sigma_i = \mu_i$. If the sets $\mathrm{Vars}(\tau_i)$ are pairwise disjoint, then the substitutions σ_i have pairwise disjoint domains and if we let $\sigma = \sigma_1 \circ \cdots \circ \sigma_n$, then $\Sigma(\tau)\sigma = \Sigma(\mu)\sigma$, a contradiction. Thus there exist $1 \leq j, k \leq n$, $j \neq k$, and a variable α such that $\alpha \in \mathrm{Vars}(\tau_j) \cap \mathrm{Vars}(\tau_k)$. For all such j, k, α, if $\alpha\sigma_j = \alpha\sigma_k$, then the substitutions all agree where there domains intersect and again we obtain the contradiction $\Sigma(\tau)\sigma = \Sigma(\mu)\sigma$. Thus, we can assume that j, k, and α are chosen such that $\alpha\sigma_j \neq \alpha\sigma_k$. For these two occurrences of α within τ (the first in τ_j) and the second in τ_k, let Σ_1 and Σ_2 be, respectively, the paths that reach them. Then $\Sigma_1(\tau) = \Sigma_2(\tau) = \alpha$, and $\Sigma_1(\mu) = \alpha\sigma_j \neq \alpha\sigma_k = \Sigma_2(\mu)$, as required.

In either case, we obtain the required result. □

The lemma guarantees that the two types of substitution outlined in the redex algorithm are the only possibilities. The theorem establishing correctness and completeness for the redex algorithm is as follows:

Theorem 3. *If $\Gamma = \{\tau_i \leq \mu_i\}_{i=1}^N$ is semiunifiable and has the property that for each i,* $\mathrm{Vars}(\tau_i) \cap \mathrm{Vars}(\mu_i) = \emptyset$*, then the above redex algorithm will terminate with success and return a semiunifier σ_Γ, which has the property that if θ is any semiunifier of σ_Γ, then there is a substitution λ such that $\theta = \lambda \circ \sigma_\Gamma$.*

Proof. As before, we show that if Γ is semiunifiable, then the algorithm will terminate with success, and that the following statement is true for $k \geq 0$ until termination: there is a substitution λ_k such that $\theta = \lambda_k \circ \sigma_k$. If $k = 0$, then $\sigma_k = []$ and we can take $\lambda_k = \theta$. In general, suppose $\theta = \lambda_k \circ \sigma_k$ for some λ_k. If σ_Γ

semiunifies Γ, then we are done, and we just take $\sigma_\Gamma = \sigma_k$. Otherwise, we perform step 3. Since the instance is not solved, then by Lemma 1, there are two possibilities, corresponding respectively to steps 3a and 3b. Hence, there are two cases to consider:

1. We perform step 3a. Since λ_k semiunifies $\Gamma\sigma_k$, it also semiunifies $\Sigma(\tau_i\sigma_k)$ and $\Sigma(\mu_i\sigma_k)$. Let λ'_k be the restriction of λ_k to $\{\Sigma(\mu_i\sigma_k)\}$. Then λ'_k also semiunifies $\Sigma(\tau_i\sigma_k)$ and $\Sigma(\mu_i\sigma_k)$. Since $\Sigma(\mu_i\sigma_k)$ does not occur in $\Sigma(\tau_i\sigma_k)$, λ'_k semiunifies $\Sigma(\tau_i\sigma_k)'$ and $\Sigma(\mu_i\sigma_k)$, i.e., there is a substitution σ such that $\Sigma(\tau_i\sigma_k)'\lambda'_k\sigma = \Sigma(\mu_i\sigma_k)\lambda'_k$. Upon simplification, we obtain $\Sigma(\tau_i\sigma_k)'\sigma = \Sigma(\mu_i\sigma_k)\lambda'_k$, which we rewrite as $\Sigma(\mu_i\sigma_k)(\sigma \circ [\Sigma(\tau_i\sigma_k)'/\Sigma(\mu_i\sigma_k)]) = \Sigma(\mu_i\sigma_k)\lambda'_k$. Since λ'_k and $\sigma \circ [\Sigma(\tau_i\sigma_k)'/\Sigma(\mu_i\sigma_k)]$ both have domain $\{\Sigma(\mu_i\sigma_k)\}$, we obtain $\lambda'_k = \sigma \circ [\Sigma(\tau_i\sigma_k)'/\Sigma(\mu_i\sigma_k)]$. The algorithm sets $\sigma_{k+1} = [\Sigma(\tau_i\sigma_k)'/\Sigma(\mu_i\sigma_k)] \circ \sigma_k$. Let λ''_k be the restriction of λ_k to variables other than $\Sigma(\mu_i\sigma_k)$. Let $\lambda_{k+1} = \sigma \circ \lambda''_k$. Then we have

$$\begin{aligned}\lambda_{k+1} \circ \sigma_{k+1} &= \sigma \circ \lambda''_k \circ [\Sigma(\tau_i\sigma_k)'/\Sigma(\mu_i\sigma_k)] \circ \sigma_k \\ &= \sigma \circ [\Sigma(\tau_i\sigma_k)'/\Sigma(\mu_i\sigma_k)] \circ \lambda''_k \circ \sigma_k \\ &= \lambda'_k \circ \lambda''_k \circ \sigma_k \\ &= \lambda_k \circ \sigma_k \\ &= \theta,\end{aligned}$$

   where the second equality follows because λ''_k and $[\Sigma(\tau_i\sigma_k)'/\Sigma(\mu_i\sigma_k)]$ have disjoint domains.

2. We perform step 3b. Since $\Sigma_1(\tau_i\sigma_k) = \Sigma_2(\tau_i\sigma_k) = \alpha$, it follows that $\Sigma_1(\mu_i\sigma_k)\lambda_k = \Sigma_2(\mu_i\sigma_k)\lambda_k$, otherwise no substitution could map α to both of these. Hence, λ_k unifies $\Sigma_1(\mu_i\sigma_k)$ and $\Sigma_2(\mu_i\sigma_k)$. Therefore, Robinson's unification algorithm will not fail on these expressions, and consequently the semiunification algorithm will not fail for this value of k. Let σ 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. Let λ'_k be the restriction of λ_k to the domain of σ. Then λ'_k also unifies $\Sigma_1(\mu_i\sigma_k)$ and $\Sigma_2(\mu_i\sigma_k)$. Hence, there is a substitution σ' such that $\lambda'_k = \sigma' \circ \sigma$. The algorithm sets $\sigma_{k+1} = \sigma \circ \sigma_k$. Let λ''_k be the restriction of λ_k to variables other than those in the domain of σ. Let $\lambda_{k+1} = \sigma' \circ \lambda''_k$. Then we have

$$\begin{aligned}\lambda_{k+1} \circ \sigma_{k+1} &= \sigma' \circ \lambda''_k \circ \sigma \circ \sigma_k \\ &= \sigma' \circ \sigma \circ \lambda''_k \circ \sigma_k \\ &= \lambda'_k \circ \lambda''_k \circ \sigma_k \\ &= \lambda_k \circ \sigma_k \\ &= \theta,\end{aligned}$$

   where the second equality follows because σ and λ''_k have disjoint domains.

The result now follows by induction. □

Corollary 1. *The redex algorithm is confluent—any two chosen sequences of redex reductions produce solutions that differ at most by variable renamings.*

Proof. Let S_1 and S_2 be solutions produced by the redex procedure. Then by the theorem, there exist substitutions S'_1 and S'_2 such that $S_2 = S'_1 \circ S_1$ and $S_1 = S'_2 \circ S_2$. Hence, $S_1 = S'_2 \circ S'_1 \circ S_1$ and $S_2 = S'_1 \circ S'_2 \circ S_2$, which gives us $S'_2 \circ S'_1 = S'_1 \circ S'_2 = []$. Hence S'_1 and S'_2 are inverses, which can only happen if S'_1 and S'_2 are either identity or renaming substitutions. □

We do not explicitly demonstrate that the redex algorithm always terminates on instances of ASUP; this follows as an immediate consequence of Theorems 6 and 7.

The theorem shows that the redex algorithm outputs a most general semiunifier for any instance of SUP on which it terminates. Further, we are free to choose any "reduction strategy" we like, without having to worry about the generality of the answer. In the next section, we show that the algorithm terminates on a wider class of instances than originally allowed.

5 Generalized "Acyclicity"

In this section, we present a more general, graph-theoretic formulation of "acyclic", upon which the redex algorithm still terminates.

Lemma 2. *If $\tau_i\sigma_k \leq \mu_i\sigma_k$ contains a redex that is reduced in step $k+1$ of the redex algorithm, then the domain of the substitution σ such that $\sigma_{k+1} = \sigma \circ \sigma_k$ (i.e., the substitution that reduces the redex) is a subset of* $\mathrm{Vars}(\mu_i\sigma_k)$. *In particaular, the domain of σ does not intersect* $\mathrm{Vars}(\tau_i\sigma_k)$.

Proof. Immediate from the statement of the algorithm. □

Lemma 3. *If a SUP instance Γ contains an inequality $\tau \leq \mu$, and the semiunifier σ output by the redex algorithm replaces a variable α in τ, then there exists an inequality $\tau' \leq \mu'$ in Γ such that $\tau' \leq \mu'$ contains a redex and α occurs in μ.*

Proof. Immediate consequence of the previous lemma. □

Our termination proof is based on removing inequalities from the problem instance until none remain, at which point the algorithm terminates. An inequality may be removed if we can prove that it contains no redex, and further will never contain a redex. Such inequalities are called *solved*:

Definition 9 (Solved). *An inequality $\tau \leq \mu$ is* solved of order 0 *if there is a substitution S such that $\tau S = \mu$, and the variables of τ do not occur on the right-hand side of any inequality in the instance. An inequality $\tau \leq \mu$ is* solved of order k *for $k > 0$ if there is a substitution S such that $\tau S = \mu$, and the variables of t do not occur on the right-hand side of any inequality in the instance that is not solved of order j for some $j < k$. An inequality is* solved *if it is solved of order k for some $k \geq 0$.*

Informally, $\tau \leq \mu$ is solved if $\tau S = \mu$ for some S, and the variables of τ do not occur on the righthand side of any unsolved inequality in the instance. The property we desire of solved inequalities, namely that they will never contain a redex, is established in the following two lemmas:

Lemma 4. *If an inequality $\tau \leq \mu$ is solved of order 0, then the redex algorithm will not find a redex in it.*

Proof. If there is a substitution S such that $\tau S = \mu$, then the empty substitution semiunifies τ and μ. Thus, if the redex algorithm produces a semiunifier σ, then by Theorem 3, there is a substitution σ' such that $\sigma' \circ \sigma = []$. This is only possible if σ is a renaming substitution, and the redex algorithm does not produce renaming substitutions. □

Lemma 5. *If an inequality is solved, then the redex algorithm will never find a redex in it.*

Proof. If an inequality $\tau \leq \mu$ is solved, then it is solved of order k for some $k \geq 0$. For $k = 0$, the result follows from the previous lemma. Otherwise, there is a substitution S such that $\tau S = \mu$ and every occurrence of every variable in $\mathrm{Vars}(\tau)$ on a right-hand side is within an inequality that is solved of order $j < k$. By induction, no redex will be found in any such inequality, and therefore, no substitution performed by the algorithm will ever replace a variable in τ. Since the empty substitution already semiunifies $\tau \leq \mu$, there are currently no redices in $\tau \leq \mu$; since no substitution will ever replace a variable in τ, there will continue to be no redices in $\tau \leq \mu$. Hence, by induction, the redex algorithm will never find a redex in a solved inequality. □

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 10 (R-acyclic). *For a graph G on a SUP instance $\Gamma = \{\tau_i \leq \mu_i\}_{i=1}^N$, define relations R, R' on variables in G as follows: $\alpha R \beta$ (resp. $\alpha R' \beta$) if there exist vertices v_i and v_j with $\alpha \in \mathrm{RVars}(v_i)$, $\beta \in \mathrm{RVars}(v_j)$, and $v_i \to^* v_j$ (resp. $v_i \to^+ v_j$). 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 10. However, it also emphasises 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 1 depicts a graph that is not R-acyclic. Note that R-acyclicity implies acyclicity.

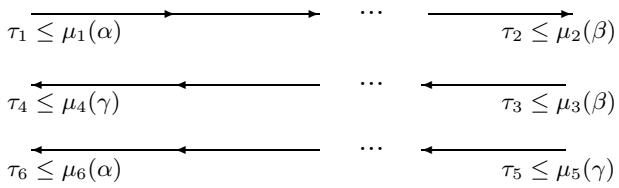

Figure 1: The relation R and R-acyclicity. The notation $\mu(\alpha)$ denotes an expression μ in which α occurs as a subexpression. Here, $\alpha R \beta$—indeed, $\alpha R' \beta$. Since also $\beta R \gamma R \alpha$, we have $\beta R^+ \alpha$; therefore, this graph is not R-acyclic.

Theorem 4 (Invariance of R-acyclicity). *Let G be the graph of a SUP instance $\Gamma = \{\tau_i \leq \mu_i\}_{i=1}^N$. If G is R-acyclic, then G continues to be R-acyclic after redex reduction.*

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 of τ, and therefore no edges are erased by this reduction. Since only those edges that contained α will now contain the variables of τ, this reduction cannot create any new edges. Hence, the cycle structure of G is unchanged 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 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 $v_j \to v_k$ lying along this path, that was created by the substitution $[\tau/\alpha]$. Hence, one of $\mathrm{RVars}(v_j)$ and $\mathrm{LVars}(v_k)$ 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 $\mathrm{RVars}(v_h)$. Since one of α and γ is in $\mathrm{LVars}(v_k)$, we have $v_h \to v_k$. Since either α or γ is in $\mathrm{RVars}(v_j)$, and both are in $\mathrm{RVars}(v_h)$, 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. Since removing edges introduced by redex-II reductions cannot convert graphs that are not R-acyclic to graphs that are, it follows that if a graph is R-acyclic, it will continue to be R-acyclic after redex-II reduction. □

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

Theorem 5. *Every instance of SUP comprising a single inequality $\tau \leq \mu$, with $\mathrm{Vars}(\tau) \cap \mathrm{Vars}(\mu) = \emptyset$, is solvable by the redex algorithm.*

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.
- *The number of redex-II reductions that can occur in $\tau \leq \mu$ before a redex-I reduction must occur is bounded by* $|\mathrm{Vars}(\mu)|$. This is because each redex-II reduction replaces at least one variable in μ; hence it decreases $|\mathrm{Vars}(\mu)|$ 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 $\leq_G$ on its vertices, defined such that $v_1 \leq_G v_2$ if there is a directed path from v_1 to v_2. The minimal elements in $\leq_G$ 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 $\leq_G$ has at least one minimal element and at least one maximal element.

Theorem 6. *Let Γ be an instance of SUP and G be the graph induced by Γ according to the statement of Theorem 1. If G is R-acyclic, then the redex algorithm will terminate on Γ.*

Proof. By induction on $|\Gamma|$. If Γ is a single inequality, then the result follows from Theorem 5. Otherwise, R-acyclicity implies that G is acyclic; therefore, let $\tau_0 \leq \mu_0$ be a source inequality in G. By confluence, we can reduce in any order we like, so let S_0 be the result of applying the redex algorithm to $\tau_0 \leq \mu_0$. Since $\tau_0 \leq \mu_0$ is a source vertex, it has no in-edges. Hence, there are no inequalities in G whose right-hand side contains any of the variables in τ_0. Then $\tau_0 S_0 \leq \mu_0 S_0$, which simplifies to $\tau_0 \leq \mu_0 S_0$, is solved of order 0, and no more redices will ever be found in this inequality. Hence all remaining redices are found in the new instance $\Gamma' = (\Gamma \setminus \{\tau_0 \leq \mu_0\})S_0$, which is strictly smaller than Γ, and whose graph, by Theorem 4, is still R-acyclic. By induction, the redex algorithm terminates on Γ', returning a solution S. Then it follows that the redex algorithm terminates on Γ, with solution $S \circ S_0$. □

Corollary 2. *The set of SUP instances that have R-acyclic graphs forms a decidable subset of SUP.*

Proof. By the theorem, there exists a terminating sequence of reductions for any instance whose graph is R-acyclic. The resulting substitution solves the instance, and is furthermore most general. □

We have now produced a solvable subset of SUP that has no dependence on path length, and further, makes no explicit requirements of variables on the left-hand sides of inequalities. This is not only a simpler description than the original, but as we shall now see, it describes a strictly larger subset of SUP:

Definition 11 (R-ASUP). *R-ASUP is the restriction of SUP to R-acyclic problem instances.*

Theorem 7. *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 10), 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. □

6 Summary and Future Work

When Henglein first presented SUP as part of his work on the Milner-Mycroft calculus, he presented a solution for the linear subset of SUP (i.e., the subset in which all functors are unary) and conjectured general solvability. SUP was later found to be solvable on all instances consisting of a single inequality. Kfoury, Tiuryn, and Urzyczyn [3] eventually showed that the full SUP is, in fact, unsolvable, though Kfoury and Wells formulated the subset of SUP known as ASUP, and demonstrated solvability.

This paper expands the class of 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 instance's graph;
- it replaces four conditions with a single condition, by eliminating explicit consideration of variables on the lefthand side;
- the relationship between R-acyclicity and the relation R clearly show 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, R-acyclicity is more widely applicable than the original formulation of acyclicity; hence, the class of known solvable instances of SUP is now increased.

Just as ASUP provides an opportunity to find meaningful solvable subsets of problems that are unsolvable due to equivalence with SUP, our new formulation of acyclicity allows us to find larger solvable subsets of these problems. In future work, we will explore the application of this work to known problems that are connected to SUP. Already (though we have not presented the result here), we can report that our work has led to a significant simplification of a well-known typing algorithm for a subset of System F. Moreover, we believe that R-acyclicity will yield simplifications beyond those we have already found.

References

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

[2] 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.

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

[4] 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.

[5] Kevin Knight. Unification: a multidisciplinary survey. *ACM Comput. Surv.*, 21(1):93–124, 1989.

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

[7] 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.

Appendix—Proof of Theorem 1

In this section, we prove Theorem 1, that our graph-theoretic formulation of acyclicity is equivalent to the original, column-based formulation:

Proof of Theorem 1. 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:

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

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

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 $\alpha_1 \in \text{Vars}(\tau_{1i})$ are in some column k, and all inequalities $\tau_{21} \leq \tau_{22}$ such that $\alpha_2 \in \text{Vars}(\tau_{2j})$ 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 $w \rightarrow w'$, label w' with label $l_w + 1$
    - for all unlabelled vertices w' such that $w' \rightarrow w$, 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 $\text{Vars}(v_1) \cap \text{Vars}(v_2) \neq \emptyset$
  - 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 $\phi \in V_i \cap V_j$. 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 $\phi \in \text{Vars}(\mu_1) \cap \text{Vars}(\mu_2)$. 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.

The other major case is 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 $i = j$.

This completes the proof. □