Constraint-Based Typing for ML via Semiunification
Computer Science Technical Report CS-2008-10

Brad Lushman Gordon V. Cormack

May 15, 2008

Abstract

We characterize ML type inference as a constraint satisfaction based on a new generalization of the semiunification problem, based on a new class of constraint variables that we call *unknowns* (we call the semiunification problem with unknowns *USUP*). Unlike previous characterizations based on ordinary semiunification, ours maintains a one-to-one correspondence between terms in the source language and constraints in the problem instance. This correspondence promises to facilitate reasoning about types by backward propagation of information back to the original source program—for example, unsolvable constraint sets could be directly translated back into specific type error information. Previous formulations of ML type inference in terms of semiunification have been less direct, thereby making practical adoption more difficult. We show that our syntax-directed formulation of ML type inference in terms of USUP is sound and complete. We give a solution semiprocedure for USUP, and show that the USUP image of any ML program is decidable. Through these contributions we wish to establish semiunification as a viable alternative to existing constraint-based typing systems for ML.

1 Introduction

Although Damas and Milner's Algorithm W [3] is, in many ways, the reference implementation for ML type inference, modern ML implementations tend to implement type inference as a constraint satisfaction problem. Typically, implementers devise a language of constraints, and then separately implement two phases of the type inference process: the translation from the type inference problem to the constraint language, and then the solution of the resulting constraint problem.

Semiunification is one of several constraint languages capable of expressing the type inference problem for ML. However, currently known translations (in particular, that of Kfoury and Wells [6]) suffer from the need to first transform the source term into a canonical form before translating to a set of semiunification constraints. Inelegance aside, the need for such a preprocessing step means that there can be no straightforward one-to-one correspondence between subterms of the source program and subsets of the set of constraints encoded in the semiunification instance. Hence, analysis made at the semiunification level cannot easily be communicated back to the source level; in particular, reporting type errors in a way that the user can understand becomes difficult.

As we show in this paper, there appears to be no syntax-directed translation from an arbitrary ML program to a semiunification instance such that the latter is solvable if and only if the former is typable. To address this problem, we consider the addition of a new class of identifier (the "unknown") to the semiunification problem. Unknowns operate almost as a hybrid between constants (i.e., nullary functors) and ordinary variables, and can be used to encode the types of broadly-scoped monomorphic variables.

$$v \in x \mid \lambda x.E$$

$$(\lambda x.E)v \to E[v/x]$$

$$\frac{E_1 \to E_2}{E_1\ E \to E_2 E}$$

$$\frac{E_1 \to E_2}{v\ E_1 \to v\ E_2}$$

$$\frac{E_1 \to E_2}{\texttt{let}\ x = E_1\ \texttt{in}\ E \to \texttt{let}\ x = E_2\ \texttt{in}\ E}$$

$$\texttt{let}\ x = v\ \texttt{in}\ E \to E[v/x]$$

Figure 1: Reduction semantics for ML.

We show that there is a syntax-directed translation from arbitrary ML source programs to a semiunification problem augmented with unknowns, such that the source program is typable if and only if the semiunification problem is solvable. Moreover, although the generalized semiunification problem is obviously undecidable, we show that the image under the translation of the set of all ML source programs is a decidable subset of the general problem.

2 Background: ML and Semiunification

The core of the ML programming language is the following grammar:

$$E \quad ::= \quad x \mid \lambda x.E \mid E\ E \mid \texttt{let}\ x = E\ \texttt{in}\ E$$

In expressions $\lambda x.E$, the variable x is bound in E; in expressions `let` $x = E_1$ `in` E_2, the variable x is bound in E_2, but not in E_1. We assume throughout this paper that all bound variables have distinct names from each other and from all free variables.

Reduction rules for this language are given in Figure 1. Although we give a call-by-value semantics to match the full ML language, the choice of reduction strategy is not essential. Note that we allow free variables to act as values in this calculus, simply to avoid the explicit introduction of literal constants. Such extensions are, however, straightforward, as are the introduction of other constructors, including products and sums.

More importantly, the type rules for the language are given in Figure 2. We use the metavariables τ and σ to range, respectively, over monotypes and polytypes. Note the two difference between the two binding operations with respect to typing. A variable bound in a λ-abstraction is forced to be given a monotype in the environment, whereas a variable bound in a `let`-construction may receive a polytype in the environment. Note that, operationally speaking, the construction `let` $x = E_1$ `in` E_2 is equivalent to the application $(\lambda x.E_2)E_1$. From a typing perspective, the two constructions would be equivalent if the x in $(\lambda x.E_2)E_1$ were allowed to assume a polymorphic type. We therefore annotate the abstraction as $(\lambda^p x.E_2)E_1$ to indicate that the function part of

$$\tau ::= \alpha \mid \tau \to \tau$$
$$\sigma ::= \tau \mid \forall \alpha . \sigma$$

$$\Gamma, x : \sigma \vdash x : \sigma$$

$$\frac{\Gamma, x : \tau_1 \vdash E : \tau_2}{\Gamma \vdash (\lambda x . E) : \tau_1 \to \tau_2}$$

$$\frac{\Gamma \vdash E_1 : \tau_1 \to \tau_2 \quad \Gamma \vdash E_2 : \tau_1}{\Gamma \vdash (E_1\ E_2) : \tau_2}$$

$$\frac{\Gamma \vdash E_1 : \sigma \quad \Gamma, x : \sigma \vdash E_2 : \tau}{\Gamma \vdash (\texttt{let}\ x = E_1\ \texttt{in}\ E_2) : \tau}$$

$$\frac{\Gamma \vdash E : \sigma}{\Gamma \vdash E : \forall \alpha . \sigma}\ (\alpha \text{ not free in } \Gamma)$$

$$\frac{\Gamma \vdash E : \forall \alpha . \sigma}{\Gamma \vdash E : \sigma[\tau / \alpha]}$$

Figure 2: Type rules for ML.

the application is a *polymorphic abstraction*—one in which the parameter x is assumed to take on a polymorphic type. To complete the distinction between the two classes of functions, we may, if we wish annotate all other abstractions as $\lambda^m z.E$, to denote *monomorphic abstractions*, in which the parameter cannot assume a polymorphic type. We will adhere to the convention that x and z denote, respectively, a polymorphic function parameter and a monomorphic function parameter.

For convenience, then, we will confine ourselves to the language expressed by the following grammar:

$$E \ ::= \ x \mid \lambda^m x.E \mid \lambda^p x.E \mid E\ E\ .$$

We restrict ourselves to the sublanguage that arises from translating `let`-constructions in ML to the application of a polymorphic abstraction to an argument. All other abstractions are assumed to be monomorphic.

Note that an abstraction may only be polymorphic if it is applied to an argument; abstractions that remain unapplied when the program terminates are necessarily monomorphic. If we like, we may further restrict our attention to the case where *all* abstractions whose arguments are present are permitted to be polymorphic abstractions. This relaxation corresponds to translating all function applications in the original source language into `let`-constructions. Since every monotype is, by inclusion, a polytype, a typable term cannot become untypable by making such otherwise monomorphic abstractions polymorphic; it is, however, possible for an untypable term to become typable under this transformation. For simplicity, we will assume for most of this paper that all abstractions whose argument is present are polymorphic abstractions (and all others are monomorphic abstractions). We call this restricted language ML$_0$. As we shall eventually see, moving to the more general case is easy.

We will consider constraint-based type inference for ML in the context of the *semiunification* *problem* (SUP), which is a problem related to terms in some term algebra. We therefore implicitly assume in the remainder of this paper that we are working in a term algebra $\mathcal{T}$ over some signature Σ, and that there is a countably infinite supply $\mathcal{V}$ of variables. The metavariables f and g will range over functors in Σ, and we will denote elements of $\mathcal{V}$ as Greek letters near the beginning of the alphabet.

Substitutions on terms in $\mathcal{T}$ are defined as follows:

Definition 1 (Substitution) *A* substitution *is a map $\sigma : \mathcal{V} \to \mathcal{T}$ that is an identity map on all but finitely many variables. We define the* domain of σ, *written* $\mathrm{dom}(\sigma)$ *as the set $\{\alpha \in \mathcal{V} \mid \sigma(\alpha) \neq \alpha\}$ of variables on which σ is not an identity. Substitutions then extend homomorphically to maps of type $\mathcal{T} \to \mathcal{T}$. We typically write substitutions as postfix operators, so that if $\tau \in \mathcal{T}$, then $\tau\sigma$ is equivalent notation to $\sigma(\tau)$.*

Definition 2 (SUP) *Let Σ be a signature, and $\mathcal{T}$ a term algebra over Σ. An instance Γ of SUP is a set $\{\tau_i \leq \mu_i\}_{i=1}^{N}$, where, for each i, τ_i and μ_i are members of $\mathcal{T}$. A solution of Γ is a substitution σ such that there exist substitutions $\sigma_1, \ldots, \sigma_N$ such that for each i,*

$$\tau_i \sigma \sigma_i = \mu_i \sigma \ .$$

Throughout this paper, as in the above definition, we will be working in a term algebra $\mathcal{T}$ over a signature Σ. For any inequality $\tau \leq \mu$, we define $\mathrm{LVars}(\tau \leq \mu)$ and $\mathrm{RVars}(\tau \leq \mu)$, to be, respectively, $\mathrm{Vars}(\tau)$ and $\mathrm{Vars}(\mu)$.

Although SUP is known to be undecidable [5], a subset of ML_0 corresponds to a decidable subset of SUP, known as R-ASUP [8]. R-ASUP is best expressed by viewing SUP instances as graphs:

Definition 3 (Graph of a SUP instance) *Let Γ be an instance of SUP. The* graph of Γ, *denoted $G(\Gamma)$, is the graph G obtained as follows:*

- *the vertices of G are the inequalities of Γ;*
- *there is an edge from v_1 to v_2 in G if $\mathrm{RVars}(v_1) \cap \mathrm{LVars}(v_2) \neq \emptyset$.*

Definition 4 *Let Γ be a SUP instance. Define relations R and R' on the variables of Γ such that, for variables α and β, we say $\alpha R \beta$ (resp. $\alpha R' \beta$) if there exist vertices v_i and v_j in $G(\Gamma)$ such that $\alpha \in \mathrm{RVars}(v_i)$, $\beta \in \mathrm{RVars}(v_j)$ and there is a path in $G(\Gamma)$ (resp. a path of nonzero length) from v_i to v_j.*

Definition 5 (R-acyclicity) *For a SUP instance Γ, the graph $G(\Gamma)$ (and, by extension, Γ itself) is said to be R-*acyclic *if, whenever $\alpha R' \beta$, for variables α and β, we have $\neg(\beta R^* \alpha)$, where R^* is the transitive closure of the (already reflexive) relation R.*

Definition 6 (R-ASUP) *R-ASUP is the restriction of SUP to R-acyclic problem instances.*

R-ASUP was shown to be a decidable subset of SUP in earlier work [8].

The solution procedure for R-ASUP is the redex-based semiprocedure given by Kfoury, Tiuryn, and Urzczyn [5] for SUP, reproduced below:

Definition 7 (Redex procedure) *Let Γ be an instance of SUP. The* redex procedure *consists of applying the following reduction rules throughout Γ as long as possible:*

- *(Redex-I reduction) Let Σ be a path and $1 \leq i \leq N$ be such that $\Sigma(\mu_i)$ is a variable and $\Sigma(\tau_i)$ is not a variable. Then apply the substitution $[\Sigma(\tau_i)'/\Sigma(\mu_i)]$ throughout Γ, where $\Sigma(\tau_i)'$ is $\Sigma(\tau_i)$ with all variables renamed consistently to fresh variables.*

- *(Redex-II reduction) Let Σ_1 and Σ_2 be paths, α a variable, and $1 \leq i \leq N$ be such that $\Sigma_1(\tau_i) = \Sigma_2(\tau_i) = \alpha$ and $\Sigma_1(\mu_i) \neq \Sigma_2(\mu_i)$. If $\Sigma_1(\mu_i)$ and $\Sigma_2(\mu_i)$ are not unifiable, terminate with failure. Else, let θ be the most general unifier of $\Sigma_1(\mu_i)$ and $\Sigma_2(\mu_i)$, as output by standard unification algorithms, and apply θ throughout Γ.*

- *If neither of the above steps is possible, but there exists i such that μ_i is not a substitution instance of τ_i, then there is a functor mismatch in the i-th inequality; terminate with failure.*

The redex procedure terminates on all solvable instances of the full SUP. For ASUP and R-ASUP, the redex procedure is a full solution procedure.

The translation from the typability problem in ML_0 to R-ASUP is based on the original ASUP translation from Kfoury and Wells [6]. Like the original, the translation relies on a source-level transformation known as *θ-reduction* that reduces every source program to the canonical form ("θ-normal form"):

$$(\lambda^p y_1.(\lambda^p y_2.\cdots(\lambda^p y_n.E_{n+1})E_n\cdots)E_2)E_1 ,$$

where each expression E_i contains no λ^p-abstractions.

The rules for θ-reduction are as follows:

- (θ_1) $((\lambda^p y.N)P)Q \rightarrow (\lambda^p y.NQ)P$

- (θ_2) $\lambda^m z.(\lambda^p y.N)P \rightarrow (\lambda^p v.\lambda^m z.(N'))(\lambda^m w.(P'))$, where $N' = N[vz/y]$, $P' = P[w/z]$, and v and w are fresh variables

- (θ_3) $N((\lambda^p y.P)Q) \rightarrow (\lambda^p y.NP)Q$

The θ-reduction operation preserves both types and β-normal forms.

The translation from θ-normal ML_0 terms to ASUP is as follows. Assume that the λ^m-variables bound in the subterms E_i are named $z_{i,j}$ for $j = 1, 2, \ldots$, and that the free variables in the source term are $w_1, w_2, \ldots$. Then for each subterm M of each E_i, the ASUP instance contains the following inequalities:

- if $M = y_j$, the inequality $\beta^y_{i,j} \leq \delta_M$;

- if $M = w_j$, the inequality $\beta^w_{i,j} \leq \delta_M$;

- if $M = z_{i,j}$, the inequality $\gamma_{i,j} = \delta_M$;

- if $M = \lambda^m z_{i,j}.E$, the equality $\delta_M = \gamma_{i,j} \rightarrow \delta_E$;

- if $M = PQ$, the equality $\delta_P = \delta_Q \rightarrow \delta_M$,

where an equality $\tau = \mu$ is syntactic sugar for the inequality $\alpha \rightarrow \alpha \leq \tau \rightarrow \mu$, where α is a fresh variable. In addition, the instance contains the following inequalities:

- for each y_j, $j < i \leq n$, the inequality $\beta^y_{i,j} \leq \beta^y_{i+1,j}$;

- for each w_j, $1 \leq i \leq n$, the inequality $\beta^w_{i,j} \leq \beta^w_{i+1,j}$.

Each redex $(\lambda^p y_i.E_{i+1})E_i$ contributes the equality $\beta^y_{i,i+1} = \delta_{E_i}$. Finally, for the free variables w_j, we consult a type environment A. If $A(w_j)$ yields a type τ, we add the equality $\beta^w_{1,j} = \tau$; otherwise we add no new inequalities.

The variables that we introduce into the ASUP problem instance have the following intended interpretations:

- for each y_j, the variable $\beta^y_{i,j}$ represents its type in subexpression E_i, for $i > j$;
- for each w_j, the variable $\beta^w_{i,j}$ represents its type in subexpression E_i;
- for each $z_{i,j}$, the variable $\gamma_{i,j}$ represents its type;
- for each subexpression M of each E_i, the variable δ_M represents its derived type (different occurrences of the same subexpression are assigned different δ-variables).

The result type of the source term is the image of the variable $\delta_{E_{n+1}}$ under the substitution that solves the R-ASUP instance.

The effect of performing a full θ-reduction on a given source term can be quite drastic; accordingly, the set of θ-normal forms is quite restricted. While having a small target set of normal forms may reduce the descriptional complexity of the remainder of the algorithm, as a side effect the resemblance between the θ-reduced term and the original term may be slight at best. Subterms that appear in the original source term may, due to the rearrangement of expressions, be absent in the θ-normal form, and vice-versa. Moreover, the θ_2-reduction in particular introduces variables and abstractions that simply do not appear anywhere in the source term.

Although the R-ASUP instance built from the θ-reduced source term does accurately describe the type of the original source term as the solution of a set of constraints, the fact that the translation from the source term to R-ASUP is not syntax-directed means that it is difficult to propagate information *backwards* from the R-ASUP instance back to the source program. In particular, if a source term is untypable, then the corresponding R-ASUP instance will fail to have a solution and the solution procedure will produce an error. However, in a real implementation, it is not sufficient to simply alert the programmer that a given source program yields a type error; the implementation must point the programmer to the part of the source program at which the error occurs, and give the derived types of the subterms involved, so that the programmer can observe the type mismatch and correct it. Since the translation to R-ASUP is not syntax-directed, there is no easy map from inequalities in the R-ASUP instance to subterms of the source program. Even if we could pinpoint a precise location within the source program that gives rise to the type error, the terms involved may not even be present in the θ-reduced version and therefore we may not have computed types for them to show the programmer in order to illustrate the error. Even in well-typed terms, we may be interested in type information about subterms; to compute the types of subterms we either need a way to derive them from the types of subterms of the θ-normal form, or a syntax-directed translation to a system of semiunification constraints.

3 Naive Syntax-Directed Translation

We consider in this section a straightforward, syntax-directed translation from ML_0 to SUP. As we shall discover, this naive translation is unsound, from which we conclude that the full SUP itself is not a suitable target for syntax-directed translation non-θ-normal terms.

The translation from θ-normal forms takes advantage of the fact that the term is θ-normal in its handling of λ^p-abstractions. We could consider, therefore, a more direct translation that works on arbitrary terms and has ordinary syntax-directed rules for λ^p-abstractions:

Definition 8 (Naive SUP translation) *Let E be a term in ML_0. Define a map NT (for "naive translation") from E to SUP as follows:*

$$\begin{array}{rcl} NT(x) & = & \{\beta_x \leq \delta_x\} \\ NT(z) & = & \{\gamma_z \leq \delta_z\} \\ NT(MN) & = & \{\delta_M = \delta_N \to \delta_{MN}\} \cup NT(M) \cup NT(N) \\ NT(\lambda^p x.M) & = & \{\delta_{\lambda^p x.M} = \beta_x \to \delta_M\} \cup NT(M) \\ NT(\lambda^m z.M) & = & \{\delta_{\lambda^m z.M} = \gamma_z \to \delta_M\} \cup NT(M) \ , \end{array}$$

where x ranges over λ^p-bound variables and z ranges over λ^m-bound variables.

We reinforce the distinction between polymorphic and monomorphic variables in the SUP instance by representing their types with β and γ, respectively, as in the original ASUP translation. Notice that we make no distinction in this definition between λ^{p_1}- and λ^{p_2}-abstractions, as both simply indicate that the parameter may have a polymorphic type. Also notice that for each node in the abstract syntax tree of the source term, there is precisely one inequality in the R-ASUP instance; hence, a reverse mapping from inequalities in the R-ASUP instance to subterms of the original term is easy to compute.

For a term E, the type of E is computed from $NT(E)$ by applying the substitution that solves $NT(E)$ to the variable δ_E, and quantifying the parameter types.

3.1 Unsoundness

It turns out that the naive translation of source terms to SUP is unsound. Consider the following term:

$$E = \lambda^m z.(\lambda^p y.yy)z \ .$$

Computing $NT(E)$ produces the following SUP instance:

$$\begin{array}{rcl} \delta_E & = & \gamma_z \to \delta_{(\lambda^p y.yy)z} \\ \delta_{\lambda^p y.yy} & = & \delta_z \to \delta_{(\lambda^p y.yy)z} \\ \delta_{\lambda^p y.yy} & = & \beta_y \to \delta_{yy} \\ \delta_{y_1} & = & \delta_{y_2} \to \delta_{yy} \\ \beta_y & \leq & \delta_{y_1} \\ \beta_y & \leq & \delta_{y_2} \\ \gamma_z & = & \delta_z \ . \end{array}$$

Notice that we use the subscripts 1 and 2 to distinguish the two occurrences of the variable y in the term, when necessary. Substituting the equalities in $NT(E)$ yields the partially reduced instance,

$$\begin{array}{rcl} \delta_E & = & \gamma_z \to \delta_{(\lambda^p y.yy)z} \\ \beta_y \to \delta_{yy} & = & \gamma_z \to \delta_{(\lambda^p y.yy)z} \\ \beta_y & \leq & \delta_{y_2} \to \delta_{yy} \\ \beta_y & \leq & \delta_{y_2} \ . \end{array}$$

By matching subterms on either side of the second equality, we obtain the equivalent formulation,

$$\begin{array}{rcl}\delta_E &=& \gamma_z \to \delta_{(\lambda^p y.yy)z} \\ \beta_y &=& \gamma_z \\ \delta_{yy} &=& \delta_{(\lambda^p y.yy)z} \\ \beta_y &\leq& \delta_{y2} \to \delta_{yy} \\ \beta_y &\leq& \delta_{y2} \ ,\end{array}$$

whence further substitution yields

$$\begin{array}{rcl}\delta_E &=& \gamma_z \to \delta_{yy} \\ \gamma_z &\leq& \delta_{y2} \to \delta_{yy} \\ \gamma_z &\leq& \delta_{y2} \ ,\end{array}$$

which is then solved. Hence, the naive translation to SUP leads to the conclusion that the source term is typable. However, the source term β-reduces as follows:

$$\lambda^m z.(\lambda^p y.yy)z \to_\beta \lambda^m z.zz \ ,$$

which is clearly not typable because the monomorphic variable z is involved in a self-application.

On the other hand, the term is not in θ-normal form. To reach θ-normal form, we first perform a θ_2-reduction:

$$(\lambda^p v.\lambda^m z.(vz)(vz))(\lambda^m w.w) \ ,$$

and the resulting term is θ-normal. The standard R-ASUP translation of this term yields a problem instance with no solution; hence we conclude that the term is, indeed, untypable. Hence, the naive SUP translation is indeed unsound.

The key difference between the original term and its θ-normal form, which gives rise to the unsoundness, is that it contains a λ^m-abstraction that outscopes a λ^p-abstraction. A broadly-scoped monomorphic abstraction allows for the possibility that a polymorphic variable, like y in our example, becomes bound, via β-reduction, to a monomorphic variable (z in our example). Thus y, being polymorphic, could be used in an expression like yy in a completely valid way, but by becoming bound to a monomorphic variable z, y effectively becomes monomorphic itself, thus rendering an expression like yy invalid.

Our task is then to find a way to express the monomorphism inherited by polymorphic variables like y above within a SUP (or SUP-like) setting. We develop a syntax-directed translation with this property in the next section.

4 Unknowns

Consider again the reduced SUP instance that comes from the naive syntax-directed translation of the term $E = \lambda^m z.(\lambda^p y.yy)z$:

$$\begin{array}{rcl}\delta_E &=& \gamma_z \to \delta_{yy} \\ \gamma_z &\leq& \delta_{y2} \to \delta_{yy} \\ \gamma_z &\leq& \delta_{y2} \ .\end{array}$$

This reduced instance has a solution because the definition of SUP allows us to supply a different substitution for γ_z between the second and third inequalities—but if γ_z represents the type of a monomorphic variable, then it should not be available for specialization to multiple values at different occurrences of the variable z. Perhaps, then, it would be better regarded as a constant. In the context of SUP, the notion of "constant" is modelled by nullary functors like *Int* and *Bool*, which are certainly monomorphic types. Indeed, if γ_z were treated as a nullary constructor, rather than as a variable, then we could immediately call the instance unsolvable, because the second inequality would contain a functor mismatch between γ_z and $\rightarrow$.

Ignoring the functor mismatch for the moment, another property of this instance (under the assumption that γ_z is a nullary functor) is noteworthy. In particular, if γ_z is a nullary functor, and therefore not a variable, then there is a redex-I in the last inequality, and the instance reduces under redex-I reduction to

$$\begin{aligned} \delta_E &= \gamma_z \rightarrow \delta_{yy} \\ \gamma_z &\leq \gamma_z \rightarrow \delta_{yy} \\ \gamma_z &\leq \gamma_z \ . \end{aligned}$$

Here we have retained the last inequality, even though it is solved, for illustrative purposes. Notice how the redex-reduction has caused γ_z to propagate to *both* sides of both the second and third inequalities. Such an occurrence would normally disqualify any instance from being either acyclic or R-acyclic, and indeed might signal non-termination. Here, however, since γ_z is a functor, there is no problem; other functors, like *Int*, *Bool*, and $\rightarrow$, have no restrictions on where they may occur, and the same must now be true for γ_z. On the other hand, we now have *two* reasons to call this instance unsolvable: in addition to the functor mismatch we previously pointed out between γ_z and $\rightarrow$, we now also have an occurs-check violation in the second inequality, because a constant can certainly not become an expression containing itself after substitution, especially since substitutions have no effect on constants! Though this second disqualification of the instance—the occurs-check violation—may seem trivial, especially in light of the first disqualification, it will turn out to be the more important of the two.

Although treating γ_z as a constant correctly renders the SUP instance from our example unsolvable, our previous experience with SUP and type inference suggests that this measure solves one problem only to create another. Consider, for example, the expression

$$(\lambda^p y.yy)(\lambda^m z.z) \ .$$

If we worked through the corresponding SUP instance, we would find that the derived type of the subexpression $(\lambda^m z.z)$ is $\gamma_z \rightarrow \gamma_z$. Equating β_y with $\gamma_z \rightarrow \gamma_z$ gives rise to the following partially-reduced instance:

$$\begin{aligned} \gamma_z \rightarrow \gamma_z &\leq \delta_{y_2} \rightarrow \delta_{yy} \\ \gamma_z \rightarrow \gamma_z &\leq \delta_{y_2} \ . \end{aligned}$$

Reducing the redex-I in the second inequality and the redex-II in the first inequality yields

$$\begin{aligned} \gamma_z \rightarrow \gamma_z &\leq (\alpha \rightarrow \alpha) \rightarrow (\alpha \rightarrow \alpha) \\ \gamma_z \rightarrow \gamma_z &\leq \alpha \rightarrow \alpha \ . \end{aligned}$$

This instance is solved, *provided that γ_z is treated as a variable*—the first inequality becomes an equation via the substitution $[\alpha \rightarrow \alpha/\gamma_z]$, while the second inequality becomes an equation via $[\alpha/\gamma_z]$. If, on the other hand, γ was a constant, then the second inequality would force $\alpha = \gamma_z$ and the first inequality would cause a functor mismatch between γ_z and $\rightarrow$ (and also an occurs-check violation).

We have, therefore, an analogous situation to our previous example: a term involving a λ^m-abstraction, whose SUP instance contains the monomorphic variable γ_z. In both cases, treating γ_z as a constant yielded a SUP instance with no solution, while treating γ_z as a variable rendered the instance solvable. In our previous example, we argued that treating γ_z as a constant yielded a correct answer (i.e., no solution). In this case however, even though z is a monomorphic entity represented by a monomorphic type variable γ_z, the *abstraction* $\lambda^m z.z$ is a *polymorphic* entity, and its type, $\gamma_z \rightarrow \gamma_z$, is a polymorphic type. Further, if we look at the computation expressed by our example term, $(\lambda^p y.yy)(\lambda^m z.z)$, we see that we are simply applying an identity function, which is polymorphic, to itself. Hence, this example, unlike the previous example, should type-check.

In summary, certain identifiers (the γ-variables) are sometimes more appropriately treated as constants (i.e., nullary functors) and sometimes more appropriately treated as variables. This observation itself strongly suggests that SUP may not be the right unification-like problem to serve as the target of a syntax-directed translation from typability. However, before we consider a generalization of SUP that fits our needs, we must examine more closely whether treating γ_z as a constant is truly the right solution, even for our first example.

Consider now the following source program:

$$E = \lambda^m z.\lambda^m w.zw \ .$$

In this example, the types of the monomorphic term variables z and w are given by the monomorphic type variables γ_z and γ_w, respectively. This source program gives rise to the following unreduced SUP instance:

$$\begin{aligned}
\delta_E &= \gamma_z \rightarrow \delta_{\lambda^m w.zw} \\
\delta_{\lambda^m w.zw} &= \gamma_w \rightarrow \delta_{zw} \\
\delta_z &= \delta_w \rightarrow \delta_{zw} \\
\gamma_z &= \delta_z \\
\gamma_w &= \delta_w \ .
\end{aligned}$$

Here, γ_z and γ_w are used in reference to the variables z and w themselves; hence, consistency with our previous example demands that we treat γ_z and γ_w as constants. We then reduce the instance by first substituting out the variables δ_z and δ_w:

$$\begin{aligned}
\delta_E &= \gamma_z \rightarrow \delta_{\lambda^m w.zw} \\
\delta_{\lambda^m w.zw} &= \gamma_w \rightarrow \delta_{zw} \\
\gamma_z &= \gamma_w \rightarrow \delta_{zw} \ .
\end{aligned}$$

Our treatment of γ_z and γ_w as constants would now lead us to conclude that there is a functor mismatch in the third inequality between γ_z and $\rightarrow$, and therefore that the term is not typable. However, we know from experience with rank 1 type inference that this term must be typable, and indeed should have the type $\forall\alpha.\forall\beta.(\alpha \rightarrow \beta) \rightarrow (\alpha \rightarrow \beta)$. Further, the original ASUP translation procedure would map the source term in this example to a solvable ASUP instance that yields precisely this type. Hence, naively treating γ-variables as constants, even in contexts where they refer to the term variables themselves, rather than the enclosing abstractions, creates incompleteness in our translation procedure.

What we observe, then, is that even when γ-variables denote monomorphic entities, and therefore behave more as constants than as variables, their behaviour is not quite consistent with that of constants either—unlike constants, γ-variables must be eligible for substitution, at least in some limited fashion. We conclude that γ-variables, when acting as monomorphic entities, constitute a new class of identifier, more constant than a variable, and more variable than a constant. In reality their behaviour can be accurately described as constants whose identities are not yet known—the substitutions we applied in our last example serve to reveal, step-by-step, more and more about the identities of these constants, without ever requiring a constant to undergo mutually incompatible substitutions (as in our second example, where γ_z acts as a variable). For this reason, we call these identifiers *unknowns*.

5 SUP with Unknowns

Our investigation in the previous section culminates in the following extension of SUP to accommodate unknowns:

Definition 9 (USUP) *An instance of USUP (i.e., SUP with Unknowns) is a set* $\{\langle \tau_i \le \mu_i, \vec{\alpha_i}\rangle\}_{i=1}^{N}$ *of pairs, each consisting of an inequality* $\tau_i \le \mu_i$ *and a (possibly empty) set* $\vec{\alpha_i}$ *of identifier names, in some term algebra. A substitution* σ *is a solution of USUP if there exist substitutions* $\sigma_1, \ldots, \sigma_N$ *such that*

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

and for each i, $\mathrm{dom}(\sigma_i) \cap \mathrm{Vars}(\vec{\alpha_i}\sigma) = \emptyset$.

The sets $\vec{\alpha_i}$ in the definition represent the identifiers in each inequality that function as unknowns; those identifiers not contained in $\vec{\alpha_i}$ function as ordinary variables. These sets $\vec{\alpha_i}$ capture our observation that identifiers that function as unknowns (i.e., γ-variables) do so only within limited contexts; elsewhere, they are simply ordinary variables. Hence, we associate a set of unknowns to each inequality, rather than globally.

The final restriction on the domain of σ_i captures our previous discussion about what kinds of substitutions unknowns may undergo. As outlined by this restriction, the key feature of unknowns is that they may be replaced by the global solution substitution σ, but they may not be replaced by the auxiliary substitutions σ_i. Moreover, an unknown replaced by substitution can only be replaced with another unknown, or an expression with only unknowns at the leaves. In particular, an unknown cannot become a variable (or an expression containing a variable) as a result of substitution.

To avoid confusion, we will use the term "identifier" to refer to either unknowns or variables in the ordinary sense. In addition, we will prefer the term "true variable" in the remainder of this paper when referring to variables that are not unknowns. However, in keeping with our previous use of the term and with the way it is used in Definition 9, the notation $\mathrm{Vars}(\tau)$ will denote the set of all *identifiers* in τ, and not merely the true variables in τ.

Before we explore USUP in detail, we introduce a notational convention. The sets $\vec{\alpha_i}$ are useful for presenting USUP formally, but cumbersome when working with concrete USUP instances. For this reason, we adopt the convention that identifiers denoting unknowns in a particular inequality will be underlined. For example, in the inequality

$$\alpha \to \underline{\beta} \ \le \ (\underline{\gamma} \to \underline{\gamma}) \to \delta \ ,$$

the identifiers β and γ denote unknowns. Of course, consistency demands that within a single inequality, either all occurrences of a given variable be underlined, or no occurrences be underlined. For example, we would not be permitted to underline only one of the occurrences of γ above. In the context of discussion, we will also adopt the convention (unless stated otherwise) that underlined identifiers denote unknowns.

5.1 Reduction Rules

In order to actually solve instances of USUP, we introduce two new reduction rules to augment the existing redex procedure. We first define some notation:

Definition 10 *Let μ be a term. Denote by μ^* the result of consistently replacing all true variables in μ with fresh unknowns.*

Then the following two reductions constitute our addition to the redex procedure to accommodate unknowns (over an assumed USUP instance $\Gamma = \{\langle \tau_i \leq \mu_i, \vec{\alpha_i} \rangle\}_{i=1}^{N}$):

- (Redex-III reduction) Let $1 \leq i \leq N$ and Σ be a minimal path such that $\Sigma(\tau_i)$ contains only unknowns at the leaves (i.e., $\text{Vars}(\Sigma(\tau_i)) \subseteq \text{Vars}(\vec{\alpha_i})$) and $\Sigma(\mu_i)$ exists, but is not equal to $\Sigma(\tau_i)$. Then apply the substitution θ throughout the problem instance, where θ is the most general unifier of $\Sigma(\tau_i)$ and $\Sigma(\mu_i)$.
- (Redex-IV reduction) Let Σ be a path and $1 \leq i \leq N$ be such that $\Sigma(\mu_i)$ is an unknown (i.e., $\Sigma(\mu_i) \in \text{Vars}(\vec{\alpha_i})$) and $\Sigma(\tau_i)$ is not a true variable (i.e., is either a functor application or an unknown). Then apply the substitution $[\Sigma(\tau_i)^*/\Sigma(\mu_i)]$ throughout the problem instance.

Implicit in Definition 9 and the above rules for redex-III's and redex-IV's is the notion of applying a substitution σ to a list $\vec{\alpha_i}$ of inequalities. Formally, we define this operation as follows:

$$\vec{\alpha_i}\sigma := \bigcup_{\gamma \in \alpha_i} \{\text{Vars}(\gamma\sigma)\} \ .$$

In addition, the existing rules must be reinterpreted in the presence of unknowns. In the case of Redex-I reduction, the phrase "not a variable" now includes not only functor applications, but also unknowns (i.e., it means "not a *true* variable"). In the case of Redex-II reduction, the issue lies with what it means to unify terms that might contain unknowns. For the purpose of unification, we will treat unknowns simply as ordinary variables, so that no extensions to standard unification algorithms are needed. Although this view of unknowns during unification may seem to allow an unknown to be replaced by a true variable, it in fact does not—as defined above, when a substitution is applied to the inequalities in the SUP instance, it is also applied to the lists of unknowns. Hence, if an unknown $\underline{\gamma}$ is replaced during unification by a true variable β, then in the sets $\vec{\alpha_i}$, occurrences of γ will also be replaced by β, thus turning β into an unknown, and producing an instance identical (up to renaming) to what would have resulted if we had insisted that $\underline{\gamma}$ replace β, rather than the reverse.

Finally, the introduction of unknowns to the problem means that we must expand our application of the occurs-check. Currently it only appears in the context of unification as part of redex-II reduction. Now, however, we must include the following check, after every redex reduction: if Σ and Π are paths, such that $\Sigma(\tau_i) = \Pi\Sigma(\mu_i) = \underline{\gamma}$ or $\Pi\Sigma(\tau_i) = \Sigma(\mu_i) = \underline{\gamma}$, for some unknown $\underline{\gamma}$, then the algorithm fails due to occurs-check violation.

5.2 Examples Recast

Having formalized USUP and its associated reduction rules, we can now revisit our examples from Section 4, this time treating the γ-variables as unknowns when they denote monomorphic entities. Our motivating example was $E = \lambda^m z.(\lambda^p y.yy)z$. Its USUP instance is

$$\begin{aligned}
\delta_E &= \gamma_z \to \delta_{yy} \\
\beta_y &= \delta_z \\
\delta_{y_1} &= \delta_{y_2} \to \delta_{yy} \\
\beta_y &\leq \delta_{y_1} \\
\beta_y &\leq \delta_{y_2} \\
\underline{\gamma_z} &= \delta_z \ .
\end{aligned}$$

Although $\underline{\gamma_z}$ only occurs within the last inequality, the identifier γ_z would be regarded as an unknown in all but the first inequality, since all of the last five inequalities arise from syntax lying strictly within the λ^m-abstraction, where γ_z can only refer to the monomorphic variable z. Therefore, reduction produces the following reduced instance:

$$\begin{aligned}
\delta_E &= \gamma_z \to \delta_{yy} \\
\underline{\gamma_z} &\leq \delta_{y_2} \to \delta_{yy} \\
\underline{\gamma_z} &\leq \delta_{y_2} \ .
\end{aligned}$$

The above instance arises from simply reducing according to the redex-I and redex-II rules from before. Now, applying the redex-III rule to final inequality equates δ_{y_2} with $\underline{\gamma_z}$, thus producing the instance

$$\begin{aligned}
\delta_E &= \gamma_z \to \delta_{yy} \\
\underline{\gamma_z} &\leq \underline{\gamma_z} \to \delta_{yy} \ .
\end{aligned}$$

Notice that we can no longer conclude that the instance is unsolvable simply because there is a functor mismatch between $\underline{\gamma_z}$ on the left and $\to$ on the right, because these now simply indicate the presence of a redex-III. Instead, we conclude that the instance is unsolvable because an unknown is being compared with an expression involving itself, which is an occurs-check violation.

Also worth noting is that we could have reduced the second inequality before attempting to reduce the third inequality. In this case, we would obtain

$$\begin{aligned}
\delta_E &= (\alpha_1 \to \alpha_2) \to \delta_{yy} \\
\underline{\alpha_1} \to \underline{\alpha_2} &\leq \delta_{y_2} \to \delta_{yy} \\
\underline{\alpha_1} \to \underline{\alpha_2} &\leq \delta_{y_2} \ .
\end{aligned}$$

Redex-III reduction on the second inequality yields

$$\begin{aligned}
\delta_E &= (\alpha_1 \to \alpha_2) \to \alpha_2 \\
\underline{\alpha_1} \to \underline{\alpha_2} &\leq \underline{\alpha_1} \to \underline{\alpha_2} \\
\underline{\alpha_1} \to \underline{\alpha_2} &\leq \underline{\alpha_1} \ ,
\end{aligned}$$

and now we have an occurs-check violation in the third inequality. Either way, we conclude that the term is untypable.

Our second example was $E = (\lambda^p y.yy)(\lambda^m z.z)$. The full USUP instance for this term is as follows:

$$\begin{aligned}
\delta_{\lambda^p y.yy} &= \delta_{\lambda^m z.z} \rightarrow \delta_E \\
\delta_{\lambda^p y.yy} &= \beta_y \rightarrow \delta_{yy} \\
\delta_{y_1} &= \delta_{y_2} \rightarrow \delta_{yy} \\
\beta_y &\leq \delta_{y_1} \\
\beta_y &\leq \delta_{y_2} \\
\delta_{\lambda^m z.z} &= \gamma_z \rightarrow \delta_z \\
\underline{\gamma_z} &= \delta_z
\end{aligned}$$

In this case, γ_z is only regarded as an unknown in the last inequality. In all other inequalities, γ_z occurs in a context in which it refers to the abstraction $\lambda^m z.z$, which is polymorphic; hence γ_z is simply a variable. Reduction of the last inequality replaces δ_z with γ_z throughout the instance; we are left with

$$\begin{aligned}
\delta_{\lambda^p y.yy} &= \delta_{\lambda^m z.z} \rightarrow \delta_E \\
\delta_{\lambda^p y.yy} &= \beta_y \rightarrow \delta_{yy} \\
\delta_{y_1} &= \delta_{y_2} \rightarrow \delta_{yy} \\
\beta_y &\leq \delta_{y_1} \\
\beta_y &\leq \delta_{y_2} \\
\delta_{\lambda^m z.z} &= \gamma_z \rightarrow \gamma_z \ .
\end{aligned}$$

Now the only inequality in which γ_z acts as an unknown is solved, and removed from the presentation. We are left with an instance that reduces, just as before, to

$$\begin{aligned}
\gamma_z \rightarrow \gamma_z &\leq (\alpha \rightarrow \alpha) \rightarrow (\alpha \rightarrow \alpha) \\
\gamma_z \rightarrow \gamma_z &\leq \alpha \rightarrow \alpha \ ,
\end{aligned}$$

which, as before, is solvable.

Our third example was $E = \lambda^m z.\lambda^m w.zw$. Its USUP instance is

$$\begin{aligned}
\delta_E &= \gamma_z \rightarrow \delta_{\lambda^m w.zw} \\
\delta_{\lambda^m w.zw} &= \gamma_w \rightarrow \delta_{zw} \\
\delta_z &= \delta_w \rightarrow \delta_{zw} \\
\underline{\gamma_z} &= \delta_z \\
\underline{\gamma_w} &= \delta_w \ ,
\end{aligned}$$

in which w is an unknown in all but the first two inequalities, and z is an unknown in all but the first inequality. Ordinary redex-I and redex-II reduction yields

$$\begin{aligned}
\delta_E &= \gamma_z \rightarrow (\gamma_w \rightarrow \delta_{zw}) \\
\underline{\gamma_z} &= \delta_w \rightarrow \delta_{zw} \\
\underline{\gamma_w} &= \delta_w \ .
\end{aligned}$$

The last two inequalities now contain a redex-III's, whose reduction yields

$$\begin{aligned}
\delta_E &= \gamma_z \rightarrow (\gamma_w \rightarrow \delta_{zw}) \\
\underline{\gamma_z} &= \underline{\gamma_w} \rightarrow \delta_{zw} \ ,
\end{aligned}$$

and then

$$\begin{array}{rcl} \delta_E & = & (\gamma_w \to \delta_{zw}) \to (\gamma_w \to \delta_{zw}) \\ \underline{\gamma_w} \to \underline{\delta_{zw}} & = & \underline{\gamma_w} \to \underline{\delta_{zw}} \ . \end{array}$$

Reading off the first inequality gives us precisely the type we would have obtained from ordinary ML type inference.

6 Properties of USUP Reduction

In this section we prove several properties of USUP and the USUP reduction procedure, in order to establish USUP as a viable basis for translation from ML_0.

Our first observation is straightforward:

Theorem 1 *USUP is undecidable.*

The result follows simply because SUP may be viewed as a subset of USUP in which there are no unknowns, or equivalently in which $\vec{\alpha_i} = \emptyset$ for all i.

Before we pursue questions of termination and decidability any further, however, we first establish some basic correctness results about our reduction procedure.

6.1 Soundness and Completeness

There are two notions of soundness and completeness surrounding our presentation of USUP. The first is whether the USUP translation of a source term faithfully and completely captures the typability of the term. The second is whether the solutions output by the reduction procedure are consistent with the definition of a solution of USUP. We treat the latter in this section, and defer the former until Section 7.2, once we have formally specified the translation from typability to USUP.

6.1.1 Soundness of the Reduction Procedure

In order to establish soundness for the reduction procedure, we must show that it never outputs wrong answers. In other words, when it produces a solution, it must be a solution according Definition 9, and it must not produce a solution for an instance that does not have one.

Lemma 1 *Suppose an instance Γ of USUP is not solved. Then, in the absence of functor mismatches, Γ contains either a redex-I, redex-II, redex-III, or redex-IV.*

Proof If $\Gamma = \{\langle \tau_i \leq \mu_i, \vec{\alpha_i} \rangle\}_{i=1}^{N}$ is not solved, then for some i, τ_i is not a substitution instance of μ_i via a substitution whose domain only contains true variables. Let Σ be as large as possible such that $\Sigma(\tau_i)$ is not a substitution instance of $\Sigma(\mu_i)$ (again, considering only substitutions whose domains contain only true variables). Let $\tau = \Sigma(\tau_i)$ and $\mu = \Sigma(\mu_i)$. Then τ is not a true variable (otherwise, τ is trivially a substitution instance of μ). On the other hand, μ can be one of:

- a true variable—then there is a redex-I reduced by $[\tau'/\mu]$.
- a functor application $f(\mu_{i_1}, \ldots, \mu_{i_n})$—then because there are no functor mismatches, if τ is also a functor application, then $\tau = f(\tau_{i_1}, \ldots, \tau_{i_n})$, for some $\tau_{i_1}, \ldots, \tau_{i_n}$. By maximality of Σ, each τ_{i_j} is a substitution instance of μ_{i_j} via a substitution σ_{i_j} whose domain only contains true

variables. Now, if all of the σ_{i_j} are compatible for $1 \leq j \leq n$, then τ itself is a substitution instance of μ, via, for example, σ_{i_1}, which is a contradiction. Hence at least two of the σ_{i_j} are not compatible. For notational convenience, suppose σ_{i_1} and σ_{i_2} are incompatible. Then there exists a variable β such that $\beta\sigma_{i_1} \neq \beta\sigma_{i_2}$. Let Σ_1 and Σ_2 be paths that, when applied to τ, produce these two occurrence of β. If $\Sigma_1(\mu)$ and $\Sigma_2(\mu)$ both exist, then $\Sigma_1(\mu) \neq \Sigma_2(\mu)$, otherwise, $\beta\sigma_{i_1} = \beta\sigma_{i_2}$. Hence, there is a redex-II, reduced by $\mathrm{MGU}(\Sigma_1(\mu), \Sigma_2(\mu))$. If, for example, $\Sigma_1(\mu)$ does not exist, then let Π be the largest prefix of Σ_1 such that $\Pi(\mu)$ exists. Then $\Pi(\mu)$ is an identifier, and $\Pi(\tau)$ is a compound expression. If $\Pi(\mu)$ is a true variable, we have a redex-I, reduced by $[\Pi(\tau)'/\Pi(\mu)]$. If $\Pi(\mu)$ is an unknown, we have a redex-IV, reduced by $[\Pi(\tau)^*/\Pi(\mu)]$.

  On the other hand, if τ is *not* a functor application, then τ must be an unknown. In this case, let Σ' be the smallest possible prefix of Σ such that $\Sigma'(\tau_i)$ contains only unknowns at the leaves (possibly Σ' is Σ itself). Then there is a redex-III at $\Sigma'(\tau_i)$, reduced by the most general unifier of $\Sigma'(\tau_i)$ and $\Sigma'(\mu_i)$.

- an unknown—if τ is a functor application, then there is a redex-IV, reduced by $[\tau^*/\mu]$. Otherwise, τ is an unknown. We cannot have $\tau = \mu$, by construction of Σ. Hence, $\tau \neq \mu$, and again there is a redex-IV, reduced by $[\tau^*/\mu]$, which is simply $[\tau/\mu]$.

As there are no other possibilities, the result follows—if an instance with no functor mismatches is not solved, it contains a redex. $\square$

Theorem 2 *If, for a USUP instance* Γ*, the USUP redex procedure outputs a substitution* σ *on termination, then* σ *is a solution of* Γ*.*

Proof If the USUP instance terminates, then $\Gamma\sigma$, the result of applying the returned substitution σ throughout Γ, must not contain any redexes. Further, it contains no functor mismatches, as otherwise, the procedure, upon encountering no redexes, would have signalled a functor mismatch. Then by Lemma 1, $\Gamma\sigma$ must be a solved instance (i.e., it has the identity substitution as solution). Hence, σ is a solution of Γ. $\square$

The following statement is immediate, but worth noting:

Corollary 1 *If a USUP instance is unsolvable, the redex procedure either loops forever, or outputs an error.*

Proof This is simply the contrapositive of Theorem 2.

These results together establish soundness for our reduction procedure—it never reports a solution when one does not exist, and any substitution it produces is guaranteed to be a solution of the problem instance.

6.1.2 Completeness of the Reduction Procedure

In considering the notion of completeness for our procedure with respect to the definition of USUP, we must be careful, as we already know USUP to be undecidable—hence, the redex procedure, being sound, cannot also be complete.

Instead, we restrict our attention to the set of USUP instances upon which the reduction procedure terminates (itself an undecidable set), and prove completeness on that set. Our completeness results are consequences of the following general theorem:

Theorem 3 *Let $\Gamma = \{\langle \tau_i \leq \mu_i, \vec{\alpha_i}\rangle\}_{i=1}^{N}$ be a solvable USUP instance, upon which the USUP redex procedure terminates, and let σ_0 be a substitution performed by one iteration of the redex procedure. Then $\Gamma\sigma_0 = \{\langle \tau_i\sigma_0 \leq \mu_i\sigma_0, \mathrm{Vars}(\vec{\alpha_i\sigma_0})\rangle\}_{i=1}^{N}$ is solvable. Moreover, if σ is any solution of the problem instance, then, when the domains are restricted to identifiers in Γ, we have $\sigma = \sigma' \circ \sigma_0$ for some substitution σ'. More precisely, $\sigma|_{\mathrm{Vars}(\Gamma)} = (\sigma' \circ \sigma_0)|_{\mathrm{Vars}(\Gamma)}$.*

Proof Let Γ be solvable. Then there is a substitution σ for which there are substitutions σ_i such that for all i,

$$\tau_i\sigma\sigma_i = \mu_i\sigma \text{ and } \mathrm{dom}(\sigma_i) \cap \mathrm{Vars}(\vec{\alpha_i}\sigma) = \emptyset \ .$$

There are four cases to consider:

- Redex-I reduction. For some i and path Σ, $\Sigma(\tau_i) = \tau$ for some expression τ, where τ is not a true variable, and $\Sigma(\mu_i) = \alpha$ for some true variable α. Since $\tau_i\sigma\sigma_i = \mu_i\sigma\sigma_i$, we have $\alpha\sigma = \tau\sigma\sigma_i$. $\alpha\sigma$ must be at least as big as $\tau\sigma$, otherwise there is no solution. Therefore, σ, restricted to Vars(Γ), can be written as $(\sigma' \circ [\tau'/\alpha])|_{Vars(\Gamma)}$ (where τ' is the consistent replacement of variables in τ by fresh ones) for some σ'. Since $[\tau'/\alpha]$ is the reduction σ_0 our procedure performs, the result follows for this case.

- Redex-II reduction. For some i and paths Σ_1 and Σ_2, we have $\Sigma_1(\tau_i) = \Sigma_2(\tau_i) = \alpha$, and $\Sigma_1(\mu_i) \neq \Sigma_2(\mu_i)$. Call these μ_{i_1} and μ_{i_2}, respectively. We have $\mu_1\sigma = \alpha\sigma\sigma_i = \mu_2\sigma$, so σ unifies μ_{i_1} and μ_{i_2}. Therefore, $\sigma = \sigma' \circ \sigma_U$, where $\sigma_U = \mathrm{MGU}(\mu_{i_1}, \mu_{i_2})$. Since σ_U is the reduction σ_0 we perform, the result holds for redex-II reduction.

- Redex-III reduction. For some i and minimal path Σ, we have $\Sigma(\tau_i) = \tau$ and $\Sigma(\mu_i) = \mu$, where τ is an expression containing only unknowns at the leaves and μ is any expression not equal to τ. Then $\tau\sigma\sigma_i = \mu\sigma$. Since σ cannot map an unknown to any expression containing true variables, and σ_i cannot have unknowns in its domain, we have $\tau\sigma\sigma_i = \tau\sigma$. Therefore, σ unifies $\underline{\gamma}$ and μ. As a result, $\sigma = \sigma' \circ \sigma_U$ for some substitution σ', where $\sigma_U = \mathrm{MGU}(\tau, \mu)$. Since σ_U is the reduction σ_0 we perform, the result holds for redex-III reduction.

- Redex-IV reduction. For some i and path Σ, we have $\Sigma(\mu_i) = \underline{\gamma}$ and $\Sigma(\tau_i) = \tau$, where $\underline{\gamma}$ is an unknown and τ is any expression. Then $\tau\sigma\sigma_i = \underline{\gamma}\sigma$. $\underline{\gamma}\sigma$ must be at least as big as τ, and must be an expression involving only unknowns. Therefore σ, when restricted to Vars(Γ), can be written as some substitution composed with $\sigma_0 = [\tau^*/\gamma]$ (both sides restricted, as usual, to Vars(Γ)), which is what the reduction rule prescribes. Hence, the result holds for redex-IV reduction.

Since this is the entire set of redex forms we reduce, our procedure does not render a solvable instance unsolvable. □

Corollary 2 *Let Γ be a solvable USUP instance, on which the USUP redex procedure terminates. Then the procedure produces a solution for Γ.*

Proof By repeated application of the theorem, each reduction results in a solvable instance. Therefore, when the procedure terminates upon finding no more redexes, the instance is solvable. Therefore, there will be no functor mismatches (and there can be no occurs-check violations in the absence of a redex). Then the only other possibility is that the redex procedure succeeds and returns a substitution on termination. Therefore, the procedure produces a solution for all solvable instances on which it terminates (correctness of this solution follows from soundness). □

Corollary 3 *Let Γ be a solvable USUP instance on which the redex procedure terminates and let σ be the solution produced by the procedure (whose existence is guaranteed by the previous corollary). Let σ' be any other solution of Γ. Then there exists a substitution σ'' such that $\sigma'|_{\mathrm{Vars}(\Gamma)} = (\sigma'' \circ \sigma)|_{\mathrm{Vars}(\Gamma)}$.*

To aid in the proof of the corollary, we introduce the following notation:

Definition 11 (Range of a Substitution) *Given a substitution σ define*

$$\mathrm{ran}(\sigma) = \bigcup_{\alpha \in \mathrm{dom}(\sigma)} \mathrm{Vars}(\alpha\sigma) \ .$$

Proof of Corollary 3 Given a set V of identifiers and a substitution σ such that $\mathrm{dom}(\sigma) \subseteq V$, define a function f on σ and V as follows:

$$f(\sigma, V) = \mathrm{ran}(\sigma) \cup (V \setminus \mathrm{dom}(\sigma)) \ .$$

The application $f(\sigma, V)$ denotes the set of values upon which a substitution σ' may act such that $(\sigma' \circ \sigma)|_V \neq \sigma$. Now, let σ_i denote the substitution performed by the i-th iteration of the redex procedure. Let $V_0 = \mathrm{Vars}(\Gamma)$. Then, from the theorem, we have

$$\sigma|_{V_0} = (\sigma' \circ \sigma_1)|_{V_0} \ ,$$

i.e.,

$$\sigma|_{V_0} = (\sigma'|_{f(\sigma_1, V_0)} \circ \sigma_1|_{V_0})|_{V_0} \ .$$

Now, for all $i > 0$, define $V_i = f(\sigma_i, V_{i-1})$, so that the equation becomes

$$\sigma|_{V_0} = (\sigma'|_{V_1} \circ \sigma_1|_{V_0})|_{V_0} \ .$$

Now, since σ solves Γ, so does $\sigma|_{V_0}$, and therefore, σ'_{V_1} solves $\Gamma\sigma_1$ (note that $\sigma_1|_{V_0} = \sigma_1$, by construction). Therefore, by a second application of the theorem,

$$\sigma'|_{V_1} = (\sigma''|_{V_2} \circ \sigma_2|_{V_1})|_{V_1} \ ,$$

for some substitution σ''. A third application of the theorem gives

$$\sigma''|_{V_2} = (\sigma'''|_{V_3} \circ \sigma_3|_{V_2})|_{V_2} \ ,$$

for some substitution σ'''. If the procedure performs n reductions before terminating, then the n-th application of the theorem gives

$$\sigma^{(n-1)}|_{V_{n-1}} = (\sigma^{(n)}|_{V_n} \circ \sigma_n|_{V_{n-1}})|_{V_{n-1}} \ ,$$

for some substitution $\sigma^{(n)}$. Substituting these equations into one another yields

$$\begin{aligned}
\sigma|_{V_0} &= (((\cdots(\sigma^{(n)}|_{V_n} \circ \sigma_n|_{V_{n-1}})|_{V_{n-1}} \circ \cdots \circ \sigma_3|_{V_2})|_{V_2} \circ \sigma_2|_{V_1})|_{V_1} \circ \sigma_1|_{V_0})|_{V_0} \\
&= (((\cdots(\sigma^{(n)} \circ \sigma_n)|_{V_{n-1}} \circ \cdots \circ \sigma_3|_{V_2})|_{V_2} \circ \sigma_2|_{V_1})|_{V_1} \circ \sigma_1|_{V_0})|_{V_0} \\
&= \cdots \\
&= (((\sigma^{(n)} \circ \sigma_n \circ \cdots \circ \sigma_3)|_{V_2} \circ \sigma_2|_{V_1})|_{V_1} \circ \sigma_1|_{V_0})|_{V_0} \\
&= (\sigma^{(n)} \circ \sigma_n \circ \cdots \circ \sigma_1)|_{V_0} \\
&= (\sigma^{(n)} \circ (\sigma_n \circ \cdots \circ \sigma_1))|_{V_0} \ .
\end{aligned}$$

Since $\sigma_n \circ \cdots \circ \sigma_1$ is the substitution returned by the procedure, we obtain the desired result. □

Thus, the USUP redex procedure is not only complete for the set of instances on which it terminates, but the solutions it outputs are most general semiunifiers (MGSU's) for the problem instance.

6.2 Termination

Having established soundness and completeness of the USUP redex procedure for instances on which it terminates, we now turn our attention to characterizing the set of instances on which the procedure terminates. As this set is not decidable, our goal will, of course, be to produce large, interesting, decidable subsets of the full set.

Our first termination result is that, in analogy to SUP, the redex procedure for USUP terminates on all USUP instances that possess a solution:

Theorem 4 *Let Γ be a USUP instance that possesses a solution. Then the USUP redex procedure terminates when applied to Γ.*

Proof Given two substitutions σ_1 and σ_2, and a USUP instance Γ, we define $\sigma_1 >_\Gamma \sigma_2$ iff

$$\sum_{\alpha \in \mathrm{Vars}(\Gamma)} |\alpha\sigma_1| > \sum_{\alpha \in \mathrm{Vars}(\Gamma)} |\alpha\sigma_2| \ ,$$

or

$$\sum_{\alpha \in \mathrm{Vars}(\Gamma)} |\alpha\sigma_1| = \sum_{\alpha \in \mathrm{Vars}(\Gamma)} |\alpha\sigma_2| \quad \text{and} \quad |\mathrm{Vars}(\Gamma\sigma_1)| < |\mathrm{Vars}(\Gamma\sigma_2)| \ ,$$

where, for an expression τ, $|\tau|$ denotes it size:

$$\begin{aligned} |\alpha| &= 1 \\ |f(\tau_1, \ldots, \tau_n)| &= 1 + \sum_{i=1}^{n} |\tau_i| \ . \end{aligned}$$

We show that the procedure must maintain the invariant that $\sigma \geq_\Gamma \sigma_k$, where σ is any solution of the instance Γ, and σ_k is the accumulated substitution after k iterations of the procedure, and moreover, $\sigma_k >_\Gamma \sigma_{k-1}$ for all $k > 1$. Since $>_\Gamma$ may be rephrased as simply a lexicographic ordering on the tuple

$$\left\langle \sum_{\alpha \in \mathrm{Vars}(\Gamma)} |\alpha\sigma_k|, \ |\mathrm{Vars}(\Gamma\sigma)| - |\mathrm{Vars}(\Gamma\sigma_k)| \right\rangle \ ,$$

it is a well-ordering, and therefore the above conditions are sufficient to guarantee termination.

Let $V = \mathrm{Vars}(\Gamma)$. For the first part of the claim, suppose σ is a solution of the instance and σ_k is the result of k iterations of the algorithm. Then by k-fold application of Theorem 3, we have that there exists a substitution σ' such that

$$\sigma|_V = (\sigma' \circ \sigma_k)|_V \ .$$

Suppose $\sigma \not\geq_\Gamma \sigma_k$. Then there are two possibilities:

- $\sum_{\alpha \in V} |\alpha\sigma| < \sum_{\alpha \in V} |\alpha\sigma_k|$. Since composing an additional substitution σ' onto σ_k can only replace identifiers at the leaves of $\alpha\sigma_k$, for each α, with other identifiers or with compound expressions, composing σ' onto σ_k cannot reduce σ_k's total size over identifiers in V; hence it cannot reduce σ_k's total size over V to that of σ, and therefore it cannot satisfy $\sigma|_V = (\sigma' \circ \sigma_k)|_V$, which contradicts our previous assertion.

- $\sum_{\alpha \in V} |\alpha\sigma| = \sum_{\alpha \in V} |\alpha\sigma_k|$, and $|\mathrm{Vars}(\Gamma\sigma)| > |\mathrm{Vars}(\Gamma\sigma_k)|$. Composing an additional substitution σ' onto σ_k replaces identifiers at the leaves of $\alpha\sigma_k$, for each α, with compound expressions or identifiers. The former would increase the total size over V of σ_k beyond that of σ, which, as we have already seen, is impossible. In the latter case, the total size of σ_k over V remains unchanged, as identifiers are simply replaced with other identifiers. In doing so, a substitution may unify two identifiers, thus reducing the total number of identifiers in the reduced instance, but it cannot split two occurrences of the same identifier into two different identifiers. Thus, composing a substitution σ' onto σ_k cannot decrease the number of variables in the reduced instance, and therefore $|\mathrm{Vars}(\Gamma\sigma_k\sigma')|$ cannot be equal to $|\mathrm{Vars}(\Gamma\sigma)|$. Thus, σ' cannot satisfy $\sigma|_V = (\sigma' \circ \sigma_k)|_V$, which contradicts our previous assertion.

Thus, indeed, any solution σ of Γ satisfies $\sigma \geq_\Gamma \sigma_k$ for all k.

It remains to show that each redex reduction makes positive progress towards a solution σ, according to $>_\Gamma$, i.e., that for all $k > 1$, $\sigma_k >_\Gamma \sigma_{k-1}$, where σ_k and σ_{k-1} denote, respectively, the accumulated substitutions produced by k and $k-1$ iterations of the algorithm. We then have that $\sigma_k = \sigma_{k,k-1} \circ \sigma_{k-1}$, where $\sigma_{k,k-1}$ is the redex reduction performed by the k-th iteration of the procedure. The proof then proceeds according to the type of replacement performed by $\sigma_{k,k-1}$:

- $\sigma_{k,k-1}$ replaces a variable α with a compound expression. Then $\alpha \in V_k$, since the algorithm, on the k-th iteration, only replaces variables in V_k. Hence, there is a variable $\beta \in V$ such that $|\beta\sigma_k| > |\beta\sigma_{k-1}|$; hence $\sigma_k > \Gamma\sigma_{k-1}$.

- $\sigma_{k,k-1}$ replaces an identifier α with an identifier. Then the total size of σ_k over V is the same as that of σ_{k-1}. Note that $\sigma_{k,k-1}$ will not replace α with a fresh identifier—the only reductions that generate fresh identifiers are redex-I and redex-IV reductions, and even within these, fresh variables are only generated when α is replaced by a functor application. Moreover, the identifier with which $\sigma_{k,k-1}$ replaces α is not α itself—the redex procedure never outputs an identity substitution. The only remaining possibility is that $\alpha\sigma_{k,k-1}$ is an identifier $\beta \in V_k$. Thus $\sigma_{k,k-1}$ unifies the identifiers α and β, so that $|\mathrm{Vars}(\Gamma\sigma_k)| = |\mathrm{Vars}(\Gamma\sigma_{k-1})| - 1$. Therefore, $\sigma_k >_\Gamma \sigma_{k-1}$.

- $\sigma_{k,k-1}$ is an MGU from redex-II or redex-III reduction. Then $\sigma_{k,k-1}$ may be viewed as a sequence of substitutions, each of which has one of the two forms above, and the result follows by the above arguments.

Having exhausted the possible forms of $\sigma_{k,k-1}$, we conclude that in all cases, $\sigma_k >_\Gamma \sigma_{k-1}$. This, combined with the fact that any solution σ of Γ bounds all σ_k from above under $>_\Gamma$, and that the double induction is well-ordered, implies termination of the USUP redex procedure on all solvable instances. $\square$

This last result implies that the USUP instances that give rise to non-termination all have no solution. Our task is now to prove that, even among the unsolvable instances, we still have termination for subsets of interest.

The problem of termination for unsolvable instances is considerably more subtle than for solvable instances—this is at least partly because USUP is undecidable, and therefore infinitely many unsolvable instances must cause the reduction procedure to loop forever.

We would ultimately like to find a graph-theoretic treatment of USUP instances, analogous to our treatment of SUP, upon which to base our termination argument. The principal difficulty in establishing such a formulation lies in deciding how to account for unknowns in the structure of the graph of a given USUP instance. While we might like, for example, to treat unknowns like true variables, such a treatment would mean that inequalities like $\alpha \to \underline{\gamma} \leq \underline{\gamma} \to \beta$, in which an unknown $\underline{\gamma}$ appears on both sides, would contain self-loops, and therefore violate all notions of acyclicity heretofore considered. We would, however, like to be able to allow an unknown to appear on both sides of the same inequality—in fact, redex-I, redex-III, and redex-IV reduction, which by definition involve copying unknowns across an inequality, depend on this ability. At the same time, we must have a notion of acyclicity (one that is pertinent to our intended application, namely type inference for ML_0) to go with our formulation of USUP graphs.

One possible approach to treating unknowns in a graph-theoretic setting might be to exclude them from consideration—since they are not strictly variables, they would not be allowed to contribute edges to the graph. An intuitive justification for this approach might be that since unknowns are meant to model constants (i.e., nullary functors like *Int* and *Bool*), which do not contribute edges to the graph, unknowns should similarly be excluded. This approach is not completely satisfactory, however, because unlike true constants, unknowns can be replaced during reductions, and therefore information can flow from one vertex to another via the replacement of an unknown.

As an example of an unknown reduction triggering an infinite reduction sequence, consider the following instance:

$$\begin{aligned} \alpha \to \alpha &\leq \beta \\ \beta &\leq \underline{\alpha}\ . \end{aligned}$$

If unknowns are not permitted to contribute edges, then the USUP graph for this instance is simply a single edge pointing from the first inequality to the second inequality, and is clearly R-acyclic. The first inequality contains a redex-I, whose reduction yields

$$\begin{aligned} \alpha \to \alpha &\leq \gamma \to \gamma \\ \gamma \to \gamma &\leq \underline{\alpha}\ . \end{aligned}$$

There is now a redex-IV in the second inequality, whose reduction yields

$$\begin{aligned} (\delta \to \delta) \to (\delta \to \delta) &\leq \gamma \to \gamma \\ \gamma \to \gamma &\leq \underline{\delta} \to \underline{\delta}\ , \end{aligned}$$

and the re-emergence of redex-I's in the first inequality makes it clear that this sequence of reductions will not terminate. On the other hand, if the unknown $\underline{\alpha}$ had been allowed to contribute an edge, then the graph would have exhibited a cycle between the two vertices, and the instance would have been rejected as a consequence.

A key feature of unknowns, not possessed by true variables, is that they permit bidirectional information flow across an inequality. With true variables, conditions on the left-hand side of an inequality can cause replacements on the right-hand side, but not vice versa. The same is possible for unknowns. Consider, for example, the following inequality:

$$\alpha \to \beta \leq \underline{\gamma}\ .$$

In this inequality, there is a redex-IV, whose reduction yields the following solved inequality:

$$\alpha \to \beta \leq \underline{\delta} \to \underline{\epsilon}\ ,$$

where $\underline{\delta}$ and $\underline{\epsilon}$ are fresh unknowns.

With unknowns, however, information flow in the reverse direction is also possible. Consider the same inequality, reversed:

$$\underline{\gamma} \leq \alpha \to \beta\ .$$

This inequality contains a redex-III, whose reduction yields

$$\underline{\alpha} \to \underline{\beta} \leq \alpha \to \beta\ .$$

This time, it is a condition on the *right-hand* side of the inequality that leads to a replacement on the *left-hand* side. Any graph-theoretic formulation of USUP, one would expect, then, must somehow be able to capture this kind of bi-directional information flow that can arise during unknown reduction.

One possible approach to constructing the graph of an instance containing unknowns might be to take each inequality $\tau \leq \mu$, in which an unknown appears, and add both the vertex $\tau \leq \mu$ and its reversal $\mu \leq \tau$ to the graph. In this way, since the inequality is present in both directions, the bidirectional information flow of unknowns is definitely captured. However, this approach also implies a bidirectional information flow for ordinary variables, which is simply not the case. The resulting graph, therefore, would be overly pessimistic about non-termination. Furthermore, by including both a vertex and its reversal in the graph, cycles will surely be introduced into the graph structure, and these would somehow need to be explained away as harmless (if this is indeed the case).

Alternatively, we might consider mapping a USUP instance into several graphs, each of which contains either $\tau \leq \mu$ or $\mu \leq \tau$ as a vertex, for each inequality $\tau \leq \mu$ that features an unknown. This approach, however, leads to an exponentially large number of graphs for a USUP instance of even modest size. Though USUP graphs are primarily a theoretical tool to facilitate reasoning, it may be advantageous to some program analysis applications to actually construct the graphs, which would prove intractable with an exponentially large number of USUP graphs. Further, such a formulation is overly *optimistic*—within a single given inequality, unknown information may flow in *both* directions. Hence, we could not be certain of catching all non-terminating instances with an acyclicity argument.

The correct approach is to treat all unknowns as occurring on right-hand sides when constructing the edges of the USUP graph:

Definition 12 (Graph of a USUP instance) *Let $\Gamma = \{\langle \tau_i \leq \mu_i, \vec{\alpha_i}\rangle\}_{i=1}^{N}$ be a USUP instance. 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* $(\mathrm{RVars}(v_i) \cup (\mathrm{LVars}(v_i) \cap \vec{\alpha_i})) \cap (\mathrm{LVars}(v_j) \setminus \vec{\alpha_j}) \neq \emptyset$.

In the absence of unknowns (i.e., if each $\vec{\alpha_i} = \emptyset$), this definition is identical to the original for SUP. When unknowns are present, however, regardless of which side of the inequality actually contains them, they are treated as if they occur on the right-hand side.

Having defined the graph of a USUP instance, we now obtain definitions of acyclic semiunification with unknowns (AUSUP) and R-acyclic semiunification with unknowns (R-AUSUP) for free.

Under this characterization, the USUP instance in our first example,

$$\begin{array}{rcl} \alpha \to \alpha & \leq & \beta \\ \beta & \leq & \underline{\alpha}\ , \end{array}$$

would have a graph in which each vertex had an edge leading to the other; hence, the graph would be cyclic, and therefore rejected. If we reversed the second inequality, thereby obtaining the instance

$$\begin{array}{rcl} \alpha \to \alpha & \leq & \beta \\ \underline{\alpha} & \leq & \beta\ , \end{array}$$

reduction of the redex-III in the second inequality would immediately create an occurs-check violation, and terminate the redex procedure. What is noteworthy about this example is that if we had simply treated unknowns as ordinary variables for the purpose of constructing the graph, the resulting graph would have had no edges, and therefore would have been trivially R-acyclic. As it is, there is now an edge from the second inequality to the first (since the unknown $\underline{\alpha}$ is regarded as being on the right-hand side, but the variable α remains on the left). Since the variable β occurs on the right-hand sides of both inequalities, which are connected by an edge, the graph cannot be R-acyclic, and is therefore rejected before the occurs-check violation arises.

Under this graph-theoretic characterization of USUP, we can allow unknowns to occur on both sides of the same inequality without creating a self-loop in the graph—since the left-hand side occurrences of the unknown are regarded as right-hand side occurrences for the purposes of the graph, there is no longer any basis for constructing an edge from the vertex to itself.

The most important property of R-acyclicity under USUP is that, like its counterpart under SUP, it is an invariant under redex reduction:

Theorem 5 (Invariance of R-acyclicity for USUP) *Let Γ be a USUP instance, and let Γ' be the result of performing one iteration of the USUP redex procedure on Γ (i.e., the result of reducing one redex in Γ). If $G(\Gamma)$ is R-acyclic, then $G(\Gamma')$ is R-acyclic.*

Proof The proof dispatches on the kind of redex reduced in Γ. Suppose the redex occurs in the inequality $\tau_i \leq \mu_i$. Then the argument proceeds as follows:

- redex-I. Then all occurrences of some true variable α are replaced with some expression τ', containing only fresh variables, along with whatever unknowns are present in the corresponding expression τ on the left-hand side. If τ contains no unknowns, then 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. If, on the other hand, τ contains an unknown $\underline{\gamma}$, then there are two possibilities:
  - the reduction $[\tau'/\alpha]$, where τ' contains the unknown $\underline{\gamma}$ induces, by the creation of edges, the relations $\beta_1 R \cdots R \beta_m$, and such that $\beta_m R' \beta_1$, for some $\beta_1, \ldots, \beta_m$. The introduction of $\underline{\gamma}$ in replacement of α can only create edges if there is an inequality $\tau_j \leq \mu_j$, in which γ appears on the left-hand side (i.e., within τ_j) as a true variable. Then an edge is created from any vertex $\tau_k \leq \mu_k$ which contains α on the right-hand side, to $\tau_k \leq \mu_k$. The introduction of this edge produces $\delta R' \beta$ for any $\delta \in \text{Vars}(\mu_k)$ and $\beta \in \text{Vars}(\mu_j)$. However, in the inequality $\tau_i \leq \mu_i$, which contains the redex, we have $\underline{\gamma} R' \beta$, and since μ_i and μ_k both contain α, we have $\delta R \alpha R \gamma R' \beta$ before reduction. Hence the introduction of $\delta R' \beta$ cannot introduce an R-cycle that was not already there. Thus $G(\Gamma')$ is R-acyclic.
  - 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-I reduction unifies β_j and β_k, then one of these two identifiers (say β_j) is the variable α, which occurs within μ_i. The other identifier (say β_k) is an unknown within τ' (without loss of generality, the unknown $\underline{\gamma}$), and occurs within τ_i. But since α occurs in μ_i, and $\underline{\gamma}$ is unknown and occurs in τ_i, we have $\alpha R \underline{\gamma}$, i.e., $\beta_j R \beta_k$. Thus, the variables β_k and β_k are R-related anyway, and any R-acyclicity violation involving these variables would have existed before they were unified. Thus, unifying these variables only results in a non-R-acyclic instance if the instance already was not R-acyclic.
- redex-II. If reduction causes G to lose R-acyclicity, then as in the case of redex-I reduction, 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 $v_j \to v_k$ lying along this path, that was created by the substitution $[\tau/\alpha]$. Hence, one of RVars(v_j) and LVars(v_k) contains the identifier α; the other contains an identifier from τ, say γ. Now, the redex $[\tau/\alpha]$ exists because of the inequality $v_i = (\tau_i \leq \mu_i)$ that satisfies the conditions for this redex-II; hence α and γ are both in RVars(v_i). Since one of α and γ is in LVars(v_k) (note that whichever of α or γ is in LVars(v_k) cannot be an unknown in v_k, otherwise the edge from v_j to v_k could not have existed in the first place), we have $v_i \to v_k$. Since either α or γ is in RVars(v_j), and both are in RVars(v_i), the transitive closure of R connects the path ending with v_j to the path beginning with v_i, 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 these two identifiers must occur together on the right-hand side of the inequality $v_i = (\tau_i \leq \mu_i)$, 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.

- redex-III. A redex-III reduction, which unifies an expression containing unknowns on the left with an arbitrary expression on the right, may be viewed as a sequence of substitutions of the form $[\tau^*/\underline{\alpha}]$ for unknowns $\underline{\alpha}$ on the left, and redex-I reductions on the right. As we have already completed the argument for redex-I reductions, we can focus our attention on a single replacement $[\tau^*/\underline{\alpha}]$ on the left-hand side. Then, as before, there are two possibilities:
  - Suppose a reduction $[\tau^*/\underline{\alpha}]$ in the vertex $v_i = (\tau_i \leq \mu_i)$ creates an edge from a vertex v_j to a vertex v_k. Let $\underline{\gamma}$ be an unknown in τ^* (if τ^* contains no unknowns, then it contains no identifiers and the replacement $[\tau^*/\alpha]$ cannot create edges). Then one of $\underline{\alpha}$ and $\underline{\gamma}$ is in RVars(v_j) (or unknown in v_j and in LVars(v_j)), and the other is in LVars(v_k) and is therefore not an unknown in v_k. On the other hand, $\underline{\alpha} \in$ LVars(v_i) and $\underline{\gamma} \in$ RVars(v_i). Thus, $v_i \to v_k$, and therefore either $\alpha R' \gamma$ or $\gamma R' \alpha$, depending on which is in LVars(v_k). But this gives, for any $\beta \in$ RVars(v_j), either $\beta R \alpha R' \gamma$ or $\beta R \gamma R' \alpha$. Hence, the edge created by the reduction is of no consequence in terms of creating non-vacuous R-cycles, and therefore, the reduced instance remains R-acyclic.
  - Suppose instead that an R-acyclicity violation arises because 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. Then these two identifiers occur in corresponding positions on opposite sides of the inequality v_i, and both are unknown. Hence, $\beta_j R \beta_k$ anyway, and any R-acyclicity violation would therefore have already existed. Hence, the reduced instance is R-acyclic.

- redex-IV. The argument for redex-IV reduction is identical to the argument for redex-III reduction, except that τ^* (hence $\underline{\gamma}$) and $\underline{\alpha}$ now occur, respectively on the left-hand and right-hand sides of v_i. However, since unknowns are considered to reside on right-hand sides for the purpose of constructing USUP graphs, we still obtain either $\alpha R'\gamma$ or $\gamma R'\alpha$, depending on whether α or γ is in $\text{LVars}(v_k)$.

In summary, then, the USUP redex procedure preserves R-acyclicity. $\square$

Corollary 4 *Let α and β be variables in an R-AUSUP instance Γ, with $\alpha R'\beta$. Let $\alpha' \in \text{Vars}(\alpha\sigma)$, $\beta' \in \text{Vars}(\beta\sigma)$. Then no reduction σ of Γ will produce $\beta' R\alpha'$.*

Proof Consider the instance

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

where x_1 and x_2 are fresh true 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 $\beta' R\alpha'$, then in Γ', we have $\alpha' R'\beta' R\alpha'$ (because no reduction will affect the two extra inequalities). Hence Γ' is now non-R-acyclic. But this contradicts the invariance of R-acyclicity. Therefore, Γ cannot reduce so as to produce $\beta' R\alpha'$.

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 the other reductions cannot create edges. Thus, the result follows either way. $\square$

Corollary 5 *Let v_1 and v_2 be vertices in the graph of an R-AUSUP 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 redexes), 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 variables $\alpha' \in \text{Vars}(\alpha\sigma)$, $\beta' \in \text{Vars}(\beta\sigma)$ such that $\beta' R'\alpha'$, contradicting the previous claim.

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

To complete our proof of termination, we must show that an infinite redex reduction sequence cannot flow through an R-acyclic graph. We first establish termination for single-inequality USUP instances. As a first step, we present the following result:

Definition 13 *We use the term* unknown reduction *to denote a redex-III or redex-IV reduction. We use the term* variable reduction *to denote a redex-I or redex-II reduction.*

Lemma 2 *Let $\Gamma = \{\langle \tau_i \leq \mu_i, \vec{\alpha_i}\rangle\}_{i=1}^{N}$ be a USUP instance. Then the number of unknown reductions that can occur between successive variable reductions in Γ is finite.*

Proof Suppose an unknown reduction in Γ replaces an unknown $\underline{\gamma}$. The proof dispatches on the image of $\underline{\gamma}$ under the reduction:

- $\underline{\gamma}$ is replaced by a functor application τ^*. Then the total number of unknowns in the instance increases by $|\text{Vars}(\tau)| - 1$. This kind of reduction can only occur when there is a path Σ such that for some i, $\Sigma(\mu_i)$ is a functor application and $\Sigma(\tau_i)$ is an unknown (i.e., redex-III reduction) or vice versa (i.e., redex-IV reduction). Thus, the number of opportunities for this kind of reduction is bounded above by the number of paths Σ for which this condition holds, which is finite. Furthermore, reductions of this kind cannot create further opportunities for reduction, as no variables are replaced with larger expressions under this reduction (only unknowns are replaced). Thus, the paths Σ leading to unknowns grow in size, and consequently the size of corresponding variable expressions shrinks. Therefore, only finitely many of this kind of reduction can occur.

- $\underline{\gamma}$ is replaced by some other unknown $\underline{\gamma}'$. In this case, the total number of distinct unknowns in the instance decreases by one. As there are only finitely many unknowns at any given time, the number of replacements by unknowns that can occur between replacements by functor applications is finite. Further, replacing unknowns with unknowns cannot create an opportunity for further replacement by functor application because, as argued above, no variables are replaced. Therefore, the number of reductions of the first kind is finite in an absolute sense (assuming no variable reductions).

Since the number of reductions in the second category that can occur between reductions of the first kind is finite, and the total number of reductions in the first category is finite, we conclude that the total number of unknown reductions that can take place in the absence of variable reductions is finite. $\square$

Before proceeding, it will be convenient to distinguish between two types of redex-I:

Definition 14 (Redex-I of the first and second kind) *A redex-I in an inequality $\tau \leq \mu$, given by a path Σ such that $\Sigma(\mu)$ is a variable and $\Sigma(\tau)$ is not a variable is said to be* of the first kind *if $\Sigma(\tau)$ contains at least one variable that is not an unknown, and* of the second kind *if $\Sigma(\tau)$ contains no variables other than unknowns.*

Note that any redex-I of the second kind may also be viewed as part of a redex-III. We now show that a single inequality can only give rise to finitely many reductions:

Lemma 3 *Every instance of USUP comprising a single inequality $\tau \leq \mu$, with unknowns $\vec{\alpha}$, and $(\text{Vars}(\tau) \cap \text{Vars}(\mu)) \setminus \vec{\alpha} = \emptyset$, is solvable by the USUP redex procedure (that is, the USUP redex procedure 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 of the first kind 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 of the first kind 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.

- *The number of unknown reductions that can take place between successive variable reductions is finite.* This is simply Lemma 2.
- *The number of redex-I reductions of the second kind in $\tau \leq \mu$ that can occur before an redex-I reduction of the first kind must occur is bounded by $|\text{Vars}(\mu) \setminus \vec{\alpha}|$.* Each redex-I reduction of the first kind replaces a true variable (i.e., not an unknown) in $\text{Vars}(\mu)$ with an expression containing only unknowns; hence, $|\text{Vars}(\mu) \setminus \vec{\alpha}|$ decreases by 1.

Since the number of unknown reductions that can occur between variable reductions is bounded, the number of redex-II reductions and redex-I reductions of the second kind that can occur between redex-I reductions of the first kind is bounded, and the total number of redex-I reductions is bounded, the USUP redex procedure must eventually terminate. $\square$

The following result states that redexes can only be induced along edges in the USUP graph:

Lemma 4 *Let $\Gamma = \{\langle \tau_i \leq \mu_i, \vec{\alpha_i} \rangle\}_{i=1}^{N}$ be an instance of USUP, and suppose a redex reduction σ in an inequality $v_i = (\tau_i \leq \mu_i)$ induces a redex in an inequality $v_j = (\tau_j \leq \mu_j)$. Then there is an edge from v_i to v_j in $G(\Gamma)$.*

Proof We proceed according to the kind of redex induced:

- redex-I. If a redex-I is created in v_j then for some path Σ, $\Sigma(\mu_j\sigma)$ is a variable, and $\Sigma(\tau_j\sigma)$ is not a variable. Since the redex-I did not exist previously, $\Sigma(\tau_j)$ must be a variable. Hence, σ contains the replacement $[\Sigma(\tau_j\sigma)/\Sigma(\tau_j)]$. Since redex reductions indicate replacements of variables on the right-hand sides of inequalities in which they originate, or of unknowns on either side, it follows that either $\Sigma(\tau_j) \in \text{Vars}(\mu_i)$ or $\Sigma(\tau_j)$ is unknown. Either way, there is an edge from v_i to v_j.
- redex-II. If a redex-II is created in v_j, then for some paths Σ_1, Σ_2, $\Sigma_1(\tau_j\sigma) = \Sigma_2(\tau_j\sigma) = \alpha$ for some variable α and $\Sigma_1(\mu_j\sigma) \neq \Sigma_2(\mu_j\sigma)$. Since substitutions cannot "un-unify" two expressions, it follows that $\Sigma_1(\mu_j) \neq \Sigma_2(\mu_j)$, or at least one of these does not exist. In the former case, since the redex did not exist previously, we must have $\Sigma_1(\tau_j) \neq \Sigma_2(\tau_j)$. Hence, at least one of these was replaced during the redex reduction, and therefore either occurs in μ_j or is unknown. Either way, there is an edge from v_i to v_j. In the latter case, the non-existence of either $\Sigma_1(\mu_j)$ or $\Sigma_2(\mu_j)$ indicates that there is already at least one redex-I at that site, and reduction of this redex-I within v_j itself would create the redex-II anyway. Hence this redex-II does not arise strictly as a result of a reduction in v_i. In those cases in which it does, however, we always have $v_i \rightarrow v_j$.
- redex-III. If a redex-III is created in v_j, then for some path Σ, $\Sigma(\tau_j\sigma)$ contains only unknowns at the leaves, $\Sigma(\mu_j\sigma)$ exists and is not equal to $\Sigma(\tau_j\sigma)$, and no prefix of Σ has this property. Since the redex did not exist previously, and reductions cannot "un-unify" expressions, we conclude that $\Sigma(\tau_j)$ must contain a variable α. Then α is replaced in v_i, which implies that $\alpha \in \text{Vars}(\mu_i)$. Hence $v_i \rightarrow v_j$.
- redex-IV. If a redex-IV is created in v_j, then for some path Σ, $\Sigma(\mu_j\sigma)$ is an unknown $\underline{\gamma}$, and $\Sigma(\tau_j\sigma)$ exists, is not a variable, and is not equal to $\Sigma(\mu_j\sigma)$. Since the redex did not exist previously, either $\Sigma(\mu_j)$ is a variable, or $\Sigma(\tau_j\sigma)$ is a variable α. In the latter case, α is replaced in v_i; hence $\alpha \in \text{Vars}(\mu_i)$, and therefore $v_i \rightarrow v_j$. In the former case, if the latter case does not also hold, then there is a redex-I between $\Sigma(\tau_j)$ and $\Sigma(\mu_j)$. In this case, the reduction in v_i only changes the form of the redex in v_j (i.e., from redex-I to redex-IV), rather than introduce a new one.

□

We are now ready to establish our main result:

Theorem 6 *Let Γ be an instance of USUP such that $G(\Gamma)$ is R-acyclic. Then the USUP redex procedure terminates on input Γ.*

Proof The proof is similar to the termination proof for R-ASUP [8]—the invariance of R-acyclicity establishes a partial order $\sqsubseteq$ on the vertices in $G(\Gamma)$ that will not be violated by any reduction. Let $\leq$ be any total order consistent with $\sqsubseteq$ and number the vertices in Γ according to this order. Then for any vertex v_i, the only vertices v_j in which v_i can induce redexes are those for which $j > i$ (this is Lemma 4). We then proceed by induction on $(n_1, \ldots, n_N)$, under lexicographic ordering, where N is the number of inequalities in Γ, and for each i, n_i represents the number of redexes present in the inequality v_i considered in isolation (each n_i is finite by arguments mirroring those in [8]). The lexicographic ordering of $(n_1, \ldots, n_N)$ is a well-ordering of the N-tuple. If a reduction occurs in vertex v_i, then n_i decreases by one, and all n_k for $k < i$ are unchanged. Thus the ordinal approaches $(0, \ldots, 0)$ with each redex reduction. If the procedure does not abort early due to an error, the ordinal will inevitably reach $(0, \ldots, 0)$, whereupon the entire instance contains no redexes, and the procedure terminates. □

Theorem 6 establishes the decidability of the problem R-AUSUP, consisting of R-acyclic USUP instances. As a consequence, the problem AUSUP, consisting of column-acyclic USUP instances, is also decidable.

7 Syntax-Directed Translation to USUP

We have formally established a generalization of SUP, which we have named USUP, together with a reduction procedure that is both sound and complete, outputs most general semiunifiers, and terminates on all solvable instances and on all R-acyclic instances. These developments were done for the purpose of formulating a syntax-directed translation from ML typability to a set of SUP-like constraints. We establish the translation in this section.

7.1 Translation Rules

Figure 3 presents our translation, which derives statements of the form $E \Rightarrow \Gamma$, where E is a labelled λ-term, and Γ is a set of USUP inequalities. Note that, because of our introduction of unknowns, it is no longer formally necessary to treat monomorphic variables and polymorphic variables differently during translation. In particular, we no longer have to be careful to translate polymorphic variable occurrences as inequalities and monomorphic variable occurrences as equations. Instead, because unknowns will propagate, via redex-III and redex-IV reductions, across inequalities anyway, we can simply translate all variable occurrences as inequalities. A side-effect of this simplification is that we now formally label the type variable associated with a given term variable as a subscripted β, irrespective of whether the variable is monomorphic or polymorphic. In the context of discussion, however, we may continue to assign monomorphic term variables type variables labelled as subscripted γ's, for the sake of clarity.

Our translation for monomorphic abstractions makes use of the notation "$\Gamma + \underline{x}$", which we use to denote the addition of x as an unknown to each inequality in Γ. This notation is defined formally as follows:

$$\frac{}{x \Rightarrow \{\langle \beta_x \le \delta_x, \{\}\rangle\}}$$

$$\frac{M \Rightarrow \Gamma_1 \quad N \Rightarrow \Gamma_2}{MN \Rightarrow \{\langle \delta_M = \delta_N \to \delta_{MN}, \{\}\rangle\} \cup \Gamma_1 \cup \Gamma_2}$$

$$\frac{E \Rightarrow \Gamma}{\lambda^p x.E \Rightarrow \{\langle \delta_{\lambda^p x.E} = \beta_x \to \delta_E, \{\}\rangle\} \cup \Gamma}$$

$$\frac{E \Rightarrow \Gamma}{\lambda^m x.E \Rightarrow \{\langle \delta_{\lambda^m x.E} = \beta_x \to \delta_E, \{\}\rangle\} \cup (\Gamma + \underline{x})}$$

Figure 3: Syntax-directed translation from rank-2 typability to USUP.

Definition 15 ($\Gamma + \underline{x}$-notation) *Given a USUP instance Γ over a term algebra whose set $\mathbb{V}$ of variables contains the identifier x, we use the notation $\Gamma + \underline{x}$ to denote the USUP instance Γ', which is identical to Γ, except that the identifier x is regarded as an unknown throughout Γ':*

$$\begin{array}{rcl} \emptyset + \underline{x} & = & \emptyset \\ (\{\langle \tau \le \mu, \vec{\alpha}\rangle\} \cup \Gamma) + \underline{x} & = & \{\langle \tau \le \mu, \vec{\alpha} \cup \{x\}\rangle\} \cup (\Gamma + \underline{x}) \end{array}$$

With this definition in place, the rule for monomorphic abstractions now simply reads that we translate a monomorphic abstraction by translating its body, adding x as an unknown to the resulting inequalities, and then adding an additional inequality relating the parameter and body type of the abstraction to the type of the abstraction itself.

Finally, the syntax-directed translation does not directly address type annotations on free variables. Under the original, non-syntax-directed translation procedure, free variables obtain their types through a programmer-supplied type environment, which we translate into ASUP constraints simply by creating an equality between the type variable assigned to the free term variable, and its assigned type. This step has been omitted from our presentation, simply because the type environment is external to the program, and therefore cannot be treated in a syntax-directed way. However, the type environment is easily accommodated by adding equations for the free variables' types to the instance at the end (accompanied by an empty set of unknowns).

For convenience in later sections, we introduce notation to capture the translation from typability to USUP:

Definition 16 *Let E be a labelled λ-term and Δ a type environment with $FTV(E) \subseteq \mathrm{dom}(\Delta)$. We define $USUP(E, \Delta)$ to be the USUP instance obtained by applying our translation procedure to the term E under the type environment Δ. When Δ is understood without ambiguity, or unimportant to the discussion, we simply write $USUP(E)$ to denote $USUP(E, \Delta)$.*

Comparing our new translation procedure with the original procedure, we see that our procedure is considerably more concise, easier to understand, and fully syntax-directed—there is a one-to-one correspondence between inequalities and the syntax elements they represent. Further, the size of the USUP instance is linear[1] in the size of the source program. In the remaining sections, we show that the USUP instance output by our translation procedure actually captures the intended semantics of ML typability, and produces terminating instances.

[1]Strictly speaking, the size of the USUP instance is not quite linear in the size of the term, because each variable

7.2 Soundness and Completeness

In Section 6.1, we showed that the USUP redex procedure is sound and complete (when it terminates) with respect to the definition of a solution of a USUP instance. The other relevant notion of soundness and completeness for USUP concerns whether the USUP representation of an ML program faithfully represents the typability of the program. It is this question that we address in this section.

Before we address questions of soundness and completeness, however, we first present type rules for ML$_0$, so that we have a semantics with which to compare our translation. These rules are presented in Figure 4. We assume that a labelling step for λ-abstractions has taken place before

$$\begin{array}{rcl} \tau & ::= & \alpha \mid \tau \to \tau \\ \pi & ::= & \tau \mid \forall\alpha.\pi \end{array}$$

$$\frac{}{\Delta \vdash x\colon \Delta(x)}\ [\text{var}] \qquad \frac{\Delta \vdash M\colon \tau_1 \to \tau_2 \quad \Delta \vdash N\colon \tau_1}{\Delta \vdash MN\colon \tau_2}\ [\text{app}]\ (M \text{ not an abstraction})$$

$$\frac{\Delta,\ x : \tau_1 \vdash E : \tau_2}{\Delta \vdash \lambda^m x.E : \tau_1 \to \tau_2}\ [\text{m-abs}] \qquad \frac{\Delta \vdash\ N : \pi \quad \Delta, x : \pi \vdash M : \tau}{\Delta \vdash (\lambda^p x.M)N : \tau}\ [\text{redex}]$$

$$\frac{\Delta \vdash E : \forall\alpha.\pi}{\Delta \vdash E : \pi[\tau/\alpha]}\ [\text{spec}] \qquad \frac{\Delta \vdash E : \pi}{\Delta \vdash E : \forall\alpha.\pi}\ [\text{gen}]\ (\alpha \text{ not free in } \Delta)$$

Figure 4: Type rules for ML$_0$.

the type rules are applied. The metavariables τ and π run, respectively, over open types and polymorphic types. Also note that we disregard the order in which quantified variables are listed on a quantified type, and will freely specialize them in any order we wish. This convenience can be justified by repeated application of rules [spec] and [gen]; however it is simpler to merely adopt the convention that the list of quantified variables in a type is unordered.

7.2.1 Soundness of the Translation

To establish soundness, intuitively we must show that, for an expression E, if $USUP(E)$ is solved by a substitution σ, which, when applied to the type variable δ_E, representing the type of E, produces a type τ, then $\forall.\tau$ is derivable as a type for E from the type rules for ML$_0$.

Note that since (by definition) the polymorphic abstractions in the ML$_0$ program are all paired with arguments, even though the type system above includes quantifiers nested to the left of the $\to$-functor, such types will have been eliminated by the end of the type inference process.

The soundness of our translation procedure is a consequence of the following theorem:

Theorem 7 *Let E be a labelled λ-term in which each variable $x \in FV(E)$ is labelled as either monomorphic or polymorphic. Let Δ be a type environment such that $FV(E) \subseteq \text{dom}(\Delta)$, and for*

that plays the role of unknown is listed with every inequality in which it is an unknown. Hence, the size of the representation is dependent on the number of unknowns times the size of their scope. However, this non-linearity can be removed using a tree-based representation of the problem instance, so that the repetition of unknowns is avoided.

each monomorphic (resp. polymorphic) $x \in FV(E)$, $\Delta(x) = \tau$ (resp. $\Delta(x) = \forall.\tau$) for some (open) type τ. Suppose that $USUP(E, \Delta)$ possesses a solution σ. Then the statement $\Delta\sigma \vdash E : \delta_E\sigma$ is derivable from the rules in Figure 4, where for every $x \in FV(E)$, $(\Delta\sigma)(x) = \beta_x\sigma$ if x is monomorphic and $(\Delta\sigma)(x) = \forall.\beta_x\sigma$ if x is polymorphic (throughout the above, the notation $\forall.\tau$ denotes quantification over all of $FTV(\tau) \setminus FTV(\Delta)$).

Proof The proof is by structural induction, dispatching on the form of the expression E:

- E is a variable x. Since x is necessarily a free variable, then by hypothesis, x has a binding in Δ. Then, depending on whether x is monomorphic or polymorphic, we proceed as follows:
  - x is polymorphic, i.e., β_x is a true variable, and not an unknown. Then $\Delta(x) = \forall.\tau$ for some open type τ. The instance $\Gamma := USUP(x, \Delta)$ contains the inequalities $\beta_x \leq \delta_E$ and $\beta_x = \tau$. It suffices to show that $\Delta\sigma \vdash E : \delta_E\sigma$ is derivable for every σ that solves Γ. Now, if σ solves Γ, then there is a substitution σ_1 such that $\beta_x\sigma\sigma_1 = \delta_E\sigma$ and $\beta_x\sigma = \tau\sigma$. Then $\tau\sigma\sigma_1 = \delta_E\sigma$. By rule [var], $\Delta \vdash E : \forall\vec{\alpha}.\tau$ is derivable, where $\vec{\alpha} = FTV(\tau) \setminus FTV(\Delta)$. Then $\Delta\sigma \vdash E : \forall.\vec{\alpha'}\tau\sigma$ (i.e., $\Delta\sigma \vdash E : \forall.\vec{\alpha'}\beta_x\sigma$) is derivable, where $\vec{\alpha'} = FTV(\tau\sigma)\setminus FTV(\Delta\sigma)$. Since the action of σ_1 on $\beta_x\sigma$ is completely determined by its action on $\vec{\alpha'}$ (all other identifiers in Vars($\beta_x\sigma$) are unknowns), we can apply rule [spec] and specialize each $\alpha \in \vec{\alpha'}$ to $\alpha\sigma$, and arrive at $\Delta\sigma \vdash x : \beta_x\sigma\sigma_1$, and since $\beta_x\sigma\sigma_1 = \delta_E\sigma$, we obtain the desired result.
  - x is monomorphic, i.e., β_x is an unknown. Then $\Delta(x) = \tau$ for some open type τ. The instance $\Gamma := USUP(x, \Delta)$ contains the inequalities $\underline{\beta_x} \leq \delta_E$ and $\underline{\beta_x} = \tau$. It suffices to show that $\Delta \vdash E : \delta_E\sigma$ is derivable for every σ that solves Γ. Now, if σ solves Γ, then there is a substitution σ_1 such that $\underline{\beta_x}\sigma\sigma_1 = \delta_E\sigma$ and $\underline{\beta_x}\sigma = \tau\sigma$. Since dom($\sigma_1$) contains no unknowns, the first equation becomes $\underline{\beta_x}\sigma = \delta_E\sigma$. Then we obtain $\tau\sigma = \delta_E\sigma$. By rule [var], $\Delta \vdash E : \tau$ is derivable. Then $\Delta\sigma \vdash E : \tau\sigma$, i.e., $\Delta\sigma \vdash E : \delta_E\sigma$ is derivable.

- E is an application MN, where M is not an abstraction. Then $USUP(E, \Delta)$ contains the equality $\delta_M = \delta_N \to \delta_E$. Then for any solution σ of $USUP(E, \Delta)$, we will have $\delta_M\sigma = \delta_N\sigma \to \delta_E\sigma$. By induction, the statement $\Delta\sigma \vdash M : \delta_M\sigma$ is derivable, and for each $x \in FV(M)$, $(\Delta\sigma)(x) = \beta_x\sigma$ if x is monomorphic and $(\Delta\sigma)(x) = \forall.\beta_x\sigma$ if x is polymorphic. Hence, the statement $\Delta\sigma \vdash M : \delta_N\sigma \to \delta_E\sigma$ is derivable. Also by induction, the statement $\Delta\sigma \vdash N : \delta_N\sigma$ is derivable, and for each $x \in FV(N)$, $(\Delta\sigma)(x) = \beta_x\sigma$ if x is monomorphic and $(\Delta\sigma)(x) = \forall.\beta_x\sigma$ if x is polymorphic. Since the statements $\Delta\sigma \vdash M : \delta_N\sigma \to \delta_E\sigma$ and $\Delta\sigma \vdash N : \delta_N\sigma$ are derivable, it follows by rule [app] that $\Delta\sigma \vdash E : \delta_E\sigma$ is derivable.

- E is a monomorphic abstraction $\lambda^m x.M$. Then $USUP(E, \Delta)$ contains the equality $\delta_E = \beta_x \to \delta_M$, and β_x is unknown (i.e., monomorphic) throughout $USUP(M, \Delta_M)$, where $\Delta_M = (\Delta, x : \beta_x)$.. Thus, for any solution σ of $USUP(E, \Delta)$, we have $\delta_E\sigma = \beta_x\sigma \to \delta_M\sigma$. By induction, the statement $\Delta_M\sigma \vdash M : \delta_M\sigma$ is derivable. Then by rule [m-abs], we can derive $\Delta\sigma \vdash \lambda^m x.M : \beta_x\sigma \to \delta_M\sigma$. Since $\delta_E\sigma = \beta_x\sigma \to \delta_M\sigma$, we obtain $\Delta\sigma \vdash E : \delta_E\sigma$.

- E is a redex $(\lambda^p x.M)N$. Let σ be a substitution that solves $USUP(E, \Delta)$. The instance $USUP(E, \Delta)$ contains the equalities $\delta_{\lambda^p x.M} = \delta_N \to \delta_E$ and $\delta_{\lambda^p x.M} = \beta_x \to \delta_M$, and β_x is variable (i.e., polymorphic) throughout $USUP(M, \Delta_M)$, where $\Delta_M = (\Delta, x : \forall.\beta_x\sigma)$. Thus, for any solution σ of $USUP(E, \Delta)$, we have $\delta_{\lambda^p x.M}\sigma = \delta_N\sigma \to \delta_E\sigma$ and $\delta_{\lambda^p x.M}\sigma = \beta_x\sigma \to \delta_M\sigma$. By induction, the statement $\Delta_M\sigma \vdash M : \delta_M\sigma$ is derivable, where for each $y \in FV(M)$, $(\Delta_M\sigma)(y) = \beta_y\sigma$ if y is monomorphic, and $(\Delta_M\sigma)(y) = \forall.\beta_y\sigma$ if y is polymorphic. Similarly, the statement $\Delta\sigma \vdash N : \delta_N\sigma$ is derivable, where for each $y \in FV(N)$, $(\Delta\sigma)(y) = \beta_y\sigma$ if

y is monomorphic, and $(\Delta\sigma)(y) = \forall.\beta_y\sigma$ if y is polymorphic. By rule [gen], the statement $\Delta\sigma \vdash N : \forall.\delta_N\sigma$ is derivable. Since $\delta_N\sigma \rightarrow \delta_E\sigma = \delta_{\lambda^p x.M}\sigma = \beta_x\sigma \rightarrow \delta_M\sigma$, we have $\beta_x\sigma = \delta_N\sigma$ and $\delta_M\sigma = \delta_E\sigma$. Then the statement $\Delta\sigma \vdash N : \forall.\delta_N\sigma$ is equivalent to $\Delta\sigma \vdash N : \forall.\beta_x\sigma$. Then by rule [redex], we can derive $\Delta\sigma \vdash (\lambda^p x.M)N : \delta_M\sigma$, which is equivalent to $\Delta\sigma \vdash (\lambda^p x.M)N : \delta_E\sigma$.

Having exhausted all cases, we conclude, by structural induction, that every solution σ of $USUP(E)$ yields a valid type derivation of E, in the manner laid out in the statement of the theorem. $\square$

In particular, the theorem implies that whenever $USUP(E)$ is solvable, then E is typable; moreover, if σ solves $USUP(E)$, then $\forall.\delta_E\sigma$ is a derivable type for E. This is the soundness result we sought.

7.2.2 Completeness of the Translation

Establishing completeness for our translation is a matter of showing that whenever an ML_0 program E is typable with type τ, then there is a substitution σ that solves $USUP(E)$, such that $\delta_E\sigma = \tau$.

We introduce the following notation:

Definition 17 *Let π be a polytype. We denote by $|\pi|$ the monotype obtained by stripping all quantifiers from π.*

We shall also need the following definition and technical result:

Definition 18 *Let $\Gamma = \{\tau_i \leq \mu_i\}_{i=1}^n$ be a SUP instance with solution σ. σ is called* canonical *if*

$$\mathrm{dom}(\sigma) \subseteq \bigcup_{i=1}^{n} \mathrm{Vars}(\mu_i) \ .$$

The following result shows that any SUP solution has a "canonical core" that is also a solution:

Theorem 8 (Canonical Solutions) *If a SUP instance $\Gamma = \{\tau_i \leq \mu_i\}_{i=1}^N$ has a solution σ, then σ can be written as $\sigma_C' \circ \sigma_C$, where σ_C is canonical,* $\mathrm{dom}\ \sigma_C \cap \mathrm{dom}\ \sigma_C' = \emptyset$, *and σ_C solves Γ. Moreover, for any $\alpha \in \mathrm{Vars}(\mu_i)$ for some i, $\alpha\sigma_C = \alpha\sigma$.*

Proof For each inequality $\tau_i \leq \mu_i$ in Γ, there is a substitution σ_i such that $\tau_i\sigma\sigma_i = \mu\sigma$. Let

$$V = \bigcup_{i=1}^{n} Vars(\mu_i) \ .$$

Let $\sigma_C = \sigma|_V$ (the restriction of σ to variables in V) and $\sigma_C' = \sigma \backslash \sigma_C$ (i.e., the restriction of σ to varia bles not in V). Then σ_C is canonical by construction, $\sigma = \sigma_C' \circ \sigma_C$, and dom $\sigma_C \cap$dom $sigma_C' = \emptyset$. It remains to show that σ_C solves Γ. We have

$$\tau_i\sigma\sigma_i = \mu\sigma \ .$$

Hence, since σ_C and σ_C' commute (their domains are disjoint),

$$\tau_i\sigma_C\sigma_C'\sigma_i = \mu\sigma_C'\sigma_C \ .$$

Sin ce dom $\sigma_C' \cap V = \emptyset$, we have

$$\tau_i\sigma_C\sigma_C'\sigma_i = \mu\sigma_C \ .$$

Grouping σ_C' and σ_i together as $\sigma_i \circ \sigma_C'$, we see that $sigma_C$ solves each inequality in Γ; hence, it solves Γ. The final claim, $\alpha\sigma_C = \alpha\sigma$, follows immediately from the construction $S_C = \sigma|_V$ and the fact that $\alpha \in V$. $\square$

The completeness theorem is as follows:

Theorem 9 *Let E be a labelled λ-term, in which each free variable is labelled as either monomorphic or polymorphic. Suppose there exists a type environment Δ whose domain contains $FV(E)$, such that each monomorphic variable in $FV(E)$ has a monomorphic binding in Δ, and each polymorphic variable in $FV(E)$ has a polymorphic binding in Δ. Furthermore, suppose there is a type π such that $\Delta \vdash E : \pi$ is derivable from the rules in Figure 4. Then there exists a substitution σ such that the following conditions hold:*

1. *σ is a solution of $USUP(E, \Delta)$;*
2. *σ is an identity on $FTV(\Delta)$;*
3. *$\delta_E\sigma = |\pi|$ (up to variable renaming);*
4. *for each variable $\alpha \in \text{Vars}(\delta_E\sigma)$: all occurrences of α on the left-hand sides of inequalities in $USUP(E, \Delta)\sigma$ (i.e., the result of reducing $USUP(E, \Delta)$) are unknown occurrences.*

Proof The proof is by induction on the type derivation for $\Delta \vdash E : \rho$. We dispatch based on the last rule applied in the derivation:

- Rule [var]. If rule [var] is at the root of the derivation, then the derivation must consist solely of the application of rule [var], and therefore, E is a variable x. Then by rule [var], $\Delta(x) = |\pi|$. $USUP(E, \Delta)$ contains the inequalities $\beta_x \leq \delta_E$ (since $E = x$, δ_E and δ_x are identical) and $\beta_x = |\pi|$. Then there are two possibilities:
  - β_x is a true variable. Then take $\sigma = [|\pi|'/\delta_E, |\pi|/\beta_x]$ (recall that $|\pi|'$ is the result of consistently replacing the variables in $|\pi|$ with fresh ones), and let σ' be the substitution that maps $|\pi|$ to $|\pi|'$. Then $\beta_x\sigma\sigma' = \delta_E\sigma$ and $\beta_x\sigma = |\pi|\sigma$; hence σ solves $USUP(E, \Delta)$, and since $\text{dom}(\sigma) = \{\beta_x, \delta_E\}$, σ is an identity on $FTV(\Delta)$. Furthermore, $\delta_E\sigma = |\pi|$, up to renaming, as required. Finally, since the variables in $|\pi|'$ are fresh, they do not occur on any left-hand sides.
  - β_x is an unknown. Then take $\sigma = [|\pi|/\delta_E, |\pi|/\beta_x]$, and take σ' to be the identity substitution. Then $\beta_x\sigma\sigma' = \delta_E\sigma$, $\beta_x\sigma = |\pi|\sigma$, and $\text{dom}(\sigma')$ contains no unknowns. Hence, σ solves $USUP(E, \Delta)$, $\text{dom}(\sigma) \cap FTV(\Delta) = \emptyset$, and $\delta_E\sigma = |\pi|$, as required. Finally, since β_x is an unknown, every identifier in τ is an unknown in $USUP(E, \Delta)\sigma$; hence, the final requirement of the theorem is satisfied vacuously.
- Rule [app]. If rule [app] is at the root of the derivation, then E is an application MN and M is not an abstraction. Then π is a monotype τ_2 and there is a monotype τ_1 such that $\Delta \vdash M : \tau_1 \to \tau_2$ and $\Delta \vdash N : \tau_2$. By induction, there are substitutions σ_M and σ_N that solve $USUP(M, \Delta)$ and $USUP(N, \Delta)$ respectively, are identity maps on $FTV(\Delta)$, and satisfy $\delta_M\sigma_M = |\tau_1 \to \tau_2| = \tau_1 \to \tau_2$ and $\delta_N\sigma_N = |\tau_2| = \tau_2$. By restricting their domains, if necessary, we may assume that $\text{dom}(\sigma_M) \subseteq \text{Vars}(USUP(M, \Delta))$ and $\text{dom}(\sigma_N) \subseteq \text{Vars}(USUP(N, \Delta))$. Since bound variables in E are assumed to be distinctly named, the only variables that occur in both M and N are those that are free in both M and N. If a variable x lies in $FV(M) \cap FV(N)$, then x is either globally free, or it is bound in an abstraction whose body contains both M and N. Either way, $x \in \text{dom}(\Delta)$, and therefore, $\beta_x\sigma_M = |\Delta(x)| = \beta_x\sigma_N$, i.e., σ_M and σ_N agree where their domains intersect. Then the set σ_E defined by $\sigma_E = \sigma_M \cup \sigma_N$ is a substitution, with $\sigma_E|_{dom(\sigma_M)} = \sigma_M$ and $\sigma_E|_{dom(\sigma_N)} = \sigma_N$, so that σ_E solves both $USUP(M, \Delta)$ and $USUP(N, \Delta)$ and is an identity on $FTV(\Delta)$. The instance $USUP(E, \Delta)$ is made up of both $USUP(M, \Delta)$ and $USUP(N, \Delta)$, as well as

the additional inequality $\delta_M = \delta_N \to \delta_E$. Since δ_E occurs in neither Vars($USUP(M, \Delta)$) nor Vars($USUP(N, \Delta)$), let $\sigma = [\tau_2/\delta_E] \circ \sigma_E$. Then σ also solves both $USUP(M, \Delta)$ and $USUP(N, \Delta)$, and is also an identity on $FTV(\Delta)$. We then have $\delta_M\sigma = \delta_M\sigma_M = \tau_1 \to \tau_2$, and $\delta_N\sigma = \delta_N\sigma_N = \tau_1$. Also, by construction, $\delta_E\sigma = \tau_2$. Putting these together, we obtain $\delta_M\sigma = \delta_N\sigma \to \delta_E\sigma$; therefore σ solves $USUP(E, \Delta)$, with $\delta_E\sigma = |\pi|$, as required.

It remains to show that the variables in τ_2 do not occur on any left-hand sides in $USUP(E, \Delta)\sigma$, except possibly where they are unknowns. By induction, the variables in $\tau_1 \to \tau_2$ do not occur in any left-hand sides (except for unknown occurrences) in $USUP(M, \Delta)\sigma_M$, and similarly for τ_1 and $USUP(N, \Delta)\sigma_N$. The non-rank-2 variables in $\tau_1 \to \tau_2$ form a superset of those in τ_2 (note that *all* variables in τ_1 are non-rank-2); hence, unknown occurrences aside, the non-rank-2 variables in τ_2 do not occur on left-hand sides in $USUP(M, \Delta)\sigma_M$, except for unknown occurrences. Let α be a variable in τ_2. If α has a non-unknown occurrence on a left-hand side in $USUP(N, \Delta)\sigma_N$ (indeed, any occurrence at all in $USUP(N, \Delta)\sigma_N$), then since α would then occur in *both* $USUP(M, \Delta)\sigma_M$ and $USUP(N, \Delta)\sigma_N$, α must originate higher in the term's parse tree than the entire application. Suppose, then, that α is a subexpression of some $\beta_x\sigma_M$ ($= \beta_x\sigma_N$), where x is a variable with a monomorphic binding occurrence. Then β_x is an unknown, and the result follows vacuously—α is then also an unknown throughout $USUP(M, \Delta)\sigma_M$ and $USUP(N, \Delta)\sigma_N$ and therefore has no occurrences of interest. If, instead, x has a polymorphic binding occurrence, then β_x only occurs on left-hand sides throughout $USUP(MN, \Delta)$, except for an equality $\beta_x = \tau$ that sets β_x by environment lookup. The variables in τ are distinct from all variables in $USUP(MN, \Delta)$. Let $\sigma_1 = [\tau/\beta_x]$. Then $USUP(MN, \Delta)$ is solvable with solution $\sigma_2 \circ \sigma_1$ for some σ_2 if and only if $USUP(MN, \Delta)\sigma_1$ is solvable with solution σ_2. By Theorem 8 (the Canonical Solutions Theorem), σ_2 can be assumed to be canonical without affecting its operation on any variable on a right-hand side. Hence, we can take $\sigma_N = \sigma_2 \circ \sigma_1$ with no effect on the value of $\delta_N\sigma_N$. Since the variables of τ are distinct from all those in the rest of $USUP(MN, \Delta)$, they occur only on left-hand sides in $USUP(MN, \Delta)\sigma_1$. Hence σ_N (and also σ_M, from previous reasoning) do not replace any variable in τ; hence apart from replacing β_x with τ, they do not replace β_x. In particular, then, $\delta_N\sigma$ does not contain β_x, or any other variable on a left-hand side throughout $USUP(MN, \Delta)$. Thus, variables in $|\pi|$ do not occur on any left-hand sides in $USUP(E, \Delta)\sigma_E$. Since σ only adds to σ_E a replacement of δ_E (which has exactly one occurrence—on a right-hand side) with $|\pi|$, the same holds for σ, as required.

- Rule [m-abs]. If rule [m-abs] is at the root of the derivation, then E is a monomorphic abstraction $\lambda^m x.M$ and x is monomorphic in M (i.e., β_x is unknown in $USUP(M, (\Delta, x : \tau))$ for any τ). Then there are monotypes τ_x and τ_M such that $|\pi| = \pi = \tau_x \to \tau_M$, and $\Delta, x : \tau_x \vdash M : \tau_M$ is derivable. Let Δ_M denote the type environment $(\Delta, x : \tau_x)$. By induction, there is a substitution σ_M that solves $USUP(M, \Delta_M)$, such that $\delta_M\sigma_M = \tau_M$, up to renaming, and σ_M is an identity on $FTV(\Delta_M)$. The instance $USUP(E, \Delta)$ contains all of $USUP(M, \Delta_M)$, as well as the equality $\delta_E = \beta_x \to \delta_M$. If x has at least one occurrence in M, then $USUP(M, \Delta_M)$ contains the equality $\beta_x = \tau_x$. Then $\beta_x\sigma_M = \tau_x\sigma_M$. Otherwise, β_x has no occurrences in $USUP(M, \Delta_M)$ and we can let $\sigma_{M,x} = [\tau_x/\beta_x] \circ \sigma_M$, and continue the argument using $\sigma_{M,x}$. For simplicity, and without loss of generality, however, we will simply assume that $\beta_x \in$ Vars($USUP(M, \Delta_M)$), and therefore, $\beta_x \in$ dom(σ_M). Since δ_E has no occurrences in $USUP(M, \Delta_M)$, define the substitution σ as $[|\pi|/\delta_E] \circ \sigma_M$. Then σ also solves $USUP(M, \Delta_M)$, and since dom(σ) = dom(σ_M) $\cup \{\delta_E\}$, σ is an identity on $FTV(\Delta_M)$, and therefore also on $FTV(\Delta)$. Since σ and σ_M are identities on $FTV(\Delta_M)$, we have $\tau_x\sigma = \tau_x\sigma_M = \tau_x$. Finally, we have

$$\begin{aligned} \delta_E\sigma &= \pi \\ &= \tau_x \to \tau_M \\ &= \beta_x\sigma_M \to \delta_M\sigma_M \\ &= \beta_x\sigma \to \delta_M\sigma \ , \end{aligned}$$

as required.

We now show that the variables in $|\pi| = \tau_x \to \tau_M$ do not occur on any left-hand sides in $USUP(E, \Delta)$, unknown occurrences aside. By induction, the variables in τ_M have only unknown occurrences on left-hand sides in $USUP(M, \Delta_M)\sigma_M$. Since σ only adds a replacement of δ_E (whose only occurrence is on a right-hand side) to σ_M, and since $USUP(E, \Delta)$ does not introduce onto any left-hand sides any variable whose only occurrences in $USUP(M, \Delta_M)$ (unknown occurrences aside) are on right-hand sides, the variables in τ_M do not occur on any left-hand sides in $USUP(E, \Delta)\sigma_E$. As for β_x, since E is a monomorphic abstraction, any occurrences of β_x in $USUP(M, \Delta_M)$ are unknown occurrences, which we may disregard. Further, β_x's only other occurrence in $USUP(E, \Delta)$ is within the equality $\delta_E = \beta_x \to \delta_M$, which is a right-hand side occurrence. Thus, $|\pi|$ contains no variables with left-hand side occurrences as true variables, as required.

- Rule [redex]. If rule [redex] is at the root of the derivation, then E has the form $(\lambda^p x.M)N$ and x is polymorphic in M (i.e., β_x is variable in $USUP(M, (\Delta, x : \forall.\tau))$ for any τ). Then there is a polytype $\forall.\tau_N$ and a monotype τ_M such that $\pi = |\pi| = \tau_M$, and the statements $\Delta \vdash N : \tau_N$ and $\Delta, x : (\forall.\tau_N) \vdash M : \tau_M$ are derivable. Let Δ_M denote the type environment $(\Delta, x : \forall.\tau_N)$. By induction, there is a substitution σ_M that solves $USUP(M, \Delta_M)$, such that $\delta_M\sigma_M = \tau_M$, and σ_M is an identity on $FTV(\Delta_M)$. Also by induction, there is a substitution σ_N that solves $USUP(N, \Delta)$, such that $\delta_N\sigma_N = \tau_N$, and σ_M is an identity on $FTV(\Delta)$. The instance $USUP(E, \Delta)$ contains all of $USUP(M, \Delta_M)$ and $USUP(N, \Delta)$, plus the equalities $\delta_{\lambda^p x.M} = \beta_x \to \delta_M$ and $\delta_{\lambda^p x.M} = \delta_N \to \delta_E$. If x has at least one occurrence in M, then $USUP(M, \Delta_M))$ contains the equality $\beta_x = \tau_x$. Then $\beta_x\sigma_M = \tau_x\sigma_M$. Otherwise, β_x has no occurrences in $USUP(M, \Delta_M)$ and we can let $\sigma_{M,x} = [\tau_x/\beta_x] \circ \sigma_M$, and continue the argument using $\sigma_{M,x}$. For simplicity, and without loss of generality, however, we will simply assume that $\beta_x \in \mathrm{Vars}(USUP(M, \Delta_M))$, and therefore, $\beta_x \in \mathrm{dom}(\sigma_M)$. Since $\delta_{\lambda^p x.M}$, and δ_E have no occurrences in $USUP(M, \Delta_M)$ and $USUP(M, \Delta)$, define the substitution σ as $[\delta_M\sigma_M/\delta_E, \beta_x\sigma_M \to \delta_M\sigma_M/\delta_{\lambda^p x.M}] \circ (\sigma_M \cup \sigma_N)$ (see the case for rule [app] for proof that $\sigma_M \cup \sigma_N$ is a well-defined substitution). Then σ also solves $USUP(M, \Delta_M)$ and $USUP(N, \Delta)$, and since $\mathrm{dom}(\sigma) = \mathrm{dom}(\sigma_M) \cup \mathrm{dom}(\sigma_N) \cup \{\delta_E, \delta_{\lambda^p x.M}\}$, σ is an identity on $FTV(\Delta)$. By construction, $\delta_{\lambda^p x.M}\sigma = \beta_x\sigma_M \to \delta_M\sigma_M = \beta_x\sigma \to \delta_M\sigma$; hence σ solves the equality $\delta_{\lambda^p x.M} = \beta_x \to \delta_M$. Also by construction, $\delta_E\sigma = \delta_M\sigma_M$, and since $USUP(M, \Delta_M)$ contains the equality $\beta_x = \tau_x$, we have $\beta_x\sigma_M = \tau_x\sigma_M$, and therefore $\beta_x\sigma = \tau_x\sigma$. Thus, the equality $\delta_{\lambda^p x.M} = \delta_N \to \delta_E$ is solved as well. have $\tau_x\sigma = \tau_x\sigma_M = \tau_x$. Therefore, σ solves $USUP(E, \Delta)$, with $\delta_E\sigma = \delta_N\sigma_N = \tau_N = |\pi|$, as required.

  The argument that the variables in $|\pi|$ have only unknown occurrences on left-hand sides mirrors the arguments for rule [m-abs] (for $\lambda^p x.M$) and [app] (for the entire expression).

- Rule [spec]. If rule [spec] is at the root of the derivation, then there is a polytype π_E, a monotype τ, and a type variable α such that $|\pi| = |\pi_E|[\tau/\alpha]$ and $\Delta \vdash E : \forall\alpha.\pi_E$ is derivable. By induction, there is a substitution σ_E that solves $USUP(E, \Delta)$, such that $\delta_E\sigma =$

$|\pi_E|$, and σ_E is an identity on $FTV(\Delta)$ and α has no occurrences on any left-hand sides in $USUP(E,\Delta)\sigma_E$. We can write $USUP(E,\Delta) = \{\langle \tau_i \leq \mu_i, \vec{\alpha_i}\rangle\}_{i=1}N$ for some N, $\tau_1,\ldots,\tau_N$, $\mu_1,\ldots,\mu_N$. Then there exist substitutions $\sigma_{E_1},\ldots,\sigma_{E_N}$ such that

$$\begin{array}{rcl} \tau_1\sigma_E\sigma_{E_1} & = & \mu_1\sigma_E \\ & \cdots & \\ \tau_N\sigma_E\sigma_{E_N} & = & \mu_N\sigma_E \ . \end{array}$$

Therefore, we have

$$\begin{array}{rcl} \tau_1\sigma_E\sigma_{E_1}[\tau/\alpha] & = & \mu_1\sigma_E[\tau/\alpha] \\ & \cdots & \\ \tau_N\sigma_E\sigma_{E_N}[\tau/\alpha] & = & \mu_N\sigma_E[\tau/\alpha] \ . \end{array}$$

Since α does not occur on any left-hand sides in $USUP(E,\Delta)\sigma_E$, we can introduce $[\tau/\alpha]$ on each left-hand side without affecting the equalities:

$$\begin{array}{rcl} \tau_1\sigma_E[\tau/\alpha]\sigma_{E_1}[\tau/\alpha] & = & \mu_1\sigma_E[\tau/\alpha] \\ & \cdots & \\ \tau_N\sigma_E[\tau/\alpha]\sigma_{E_N}[\tau/\alpha] & = & \mu_N\sigma_E[\tau/\alpha] \ . \end{array}$$

But this system now says that $[\tau/\alpha]\circ\sigma_E$ solves $USUP(E,\Delta)$. Furthermore,

$$\delta_E([\tau/\alpha]\circ\sigma_E) \ = \ \delta_E\sigma_E[\tau/\alpha] \ = \ |\pi_E|[\tau/\alpha] \ = \ |\pi|.$$

Since α is a quantified variable in $\forall\alpha.\rho_E$, and quantification can only be introduced by application of rule [gen], it follows that α is not free in Δ. By induction, σ_E is an identity on $FTV(\Delta)$. Since σ only adds to σ_E a replacement of α, which is not free in Δ, it follows that σ is also an identity on $FTV(\Delta)$. Also by induction, each variable in $|\pi|$ has only unknown occurrences on left-hand sides in $USUP(E,\Delta)$. Any variables τ has in common with $|\pi|$ obviously have this property as well. For any variables in τ but not in $|\pi|$, either these variables, or the variables in $USUP(E,\Delta)$ can be renamed such that the two sets of variable names are disjoint. As a result, any variables in $|\pi|[\tau/\alpha]$ have only unknown occurrences on left-hand sides in $USUP(E,\Delta)\sigma_E[\tau/\alpha]$, which is the needed result.

- Rule [gen]. If rule [gen] is at the root of the derivation, then there is polytype π such that $\Delta \vdash E : \pi$ is derivable. By induction, there is a substitution σ_E that solves $USUP(E,\Delta)$, such that $\delta_E\sigma = |\pi|$, σ is an identity on $FTV(\Delta)$, and the variables in $Q^{-1}(\rho)$ have only unknown occurrences on left-hand sides in $USUP(E,\Delta)\sigma_E$. Then, since $|\forall\alpha.\pi| = |\pi|$, the same substitution σ_E can be used to satisfy the theorem for the statement $\Delta \vdash E : \forall\alpha.\pi$, and the result follows immediately.

Having exhausted the possible forms of a valid type derivation for an expression E, we conclude by induction that a solution σ exists for $USUP(E,\Delta)$, according to the conditions laid out in the statement of the theorem. $\square$

The statement and proof of the completeness theorem are long and technical—much of the technical awkwardness arises from the need to properly handle quantifiers and the [spec] and [gen] rules. However, despite its complexity, the completeness theorem implies a relatively straightforward result: any type π derivable for an expression E arises as a solution for $USUP(E)$, which is the result we sought.

7.3 Termination

Our syntax-directed translation from ML$_0$ to USUP is sound and complete. In order to establish USUP, together with the associated translation procedure, as a viable alternative to the original SUP translation, it remains to establish termination—that for any ML$_0$ program E, the USUP redex procedure terminates on input $USUP(E)$.

In Section 6.2, we showed that the USUP redex procedure terminates on all USUP instances that possess a solution; hence, whenever an ML$_0$ program E is typable, the USUP redex procedure will terminate on input $USUP(E)$, and we will obtain a valid type for E. For the general case, however, we need a more general result. In Section 6.2, we also showed that the USUP redex procedure terminates on all R-acyclic instances of USUP. Hence, establishing R-acyclicity for $G(USUP(E))$, for an arbitrary E, would give us the result we seek.

It turns out, however, that even for a simple, θ-normal, source term, this result (that is, R-acyclicity in the graph of the corresponding USUP instance) is false. Consider the following example:

$$E = (\lambda^p y.y)(\lambda^m z.z)\ .$$

Then $USUP(E)$ contains the following inequalities[2]:

$$\begin{aligned}
\alpha \to \alpha &\leq \delta_{\lambda^p y.y} \to (\delta_{\lambda^m z.z} \to \delta_E) \\
\alpha \to \alpha &\leq \delta_{\lambda^p y.y} \to (\beta_y \to \delta_y) \\
\beta_y &\leq \delta_y \\
\alpha \to \alpha &\leq \delta_{\lambda^m z.z} \to (\beta_z \to \delta_z) \\
\underline{\beta_z} &\leq \delta_z\ .
\end{aligned}$$

Consider now the graph $G = G(USUP(E))$. The relevant portion of G for our purposes is the subgraph containing the second and third inequalities. This subgraph appears in Figure 5. The

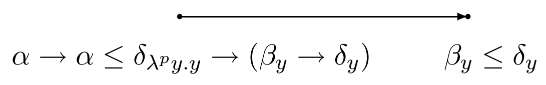
$\alpha \to \alpha \leq \delta_{\lambda^p y.y} \to (\beta_y \to \delta_y)$
$\beta_y \leq \delta_y$

Figure 5: Portions of the USUP graph for the expression $E = (\lambda^p y.y)(\lambda^m z.z)$.

edge in Figure 5 exists because the variable β_y occurs on the right-hand side of $\alpha \to \alpha \leq \delta_{\lambda^p y.y} \to (\beta_y \to \delta_y)$ and on the left-hand side of $\beta_y \leq \delta_y$. Hence every variable on the right-hand side of $\alpha \to \alpha \leq \delta_{\lambda^p y.y} \to (\beta_y \to \delta_y)$ is R' related to every variable on the right-hand side of $\beta_y \leq \delta_y$. In particular, $\delta_y R' \delta_y$. Since every variable is R-related to itself, via a path of length 0, we have $\delta_y R' \delta_y R \delta_y$, which is a violation of the condition for R-acyclicity. In general, whenever identifier occurs on the right-hand sides of two distinct inequalities that are joined by a directed path, there is an R-acyclicity violation.

The following propositions make explicit the idea that, under the right circumstances, an unknown is indistinguishable from a variable.

Proposition 1 *If $\underline{\gamma}$ is an unknown, then for every expression τ, the inequalities $\underline{\gamma} \leq \tau$ and $\underline{\gamma} = \tau$ have the same set of solutions.*

[2]For illustrative purposes, equalities are explicitly written here as inequalities.

Proof Let σ be a substitution, and suppose σ solves $\underline{\gamma} \leq \tau$. Then there is a substitution σ' such that $\gamma\sigma\sigma' = \tau\sigma$, where $\mathrm{dom}(\sigma')$ contains no unknowns. In the context of this inequality, then, we have $\mathrm{dom}(\sigma') \cap \mathrm{Vars}(\gamma\sigma) = \emptyset$. Hence, $\gamma\sigma\sigma' = \gamma\sigma$, and therefore, $\gamma\sigma = \tau\sigma$. The second inequality is, in fact, $\alpha \to \alpha \leq \gamma \to \tau$, where α is a fresh inequality (hence $\alpha \notin \mathrm{dom}(\sigma)$). Applying σ gives $\alpha\sigma \to \alpha\sigma \leq \gamma\sigma \to \tau\sigma$, which simplifies to $\alpha \to \alpha \leq \gamma\sigma \to \tau\sigma$. Taking $\sigma' = [\gamma\sigma/\alpha]$, we have $\alpha\sigma' \to \alpha\sigma' = \gamma\sigma \to \tau\sigma$; hence σ solves $\underline{\gamma} = \tau$. Conversely, if σ solves $\underline{\gamma} = \tau$, then there is a substitution σ' such that $\alpha\sigma\sigma' \to \alpha\sigma\sigma' = \gamma\sigma \to \tau\sigma$. Then we have $\gamma\sigma = \alpha\sigma\sigma' = \tau\sigma$. Applying σ to $\underline{\gamma} \leq \tau$ gives $\underline{\gamma}\sigma = \tau\sigma$, and since $\underline{\gamma}\sigma = \tau\sigma$, we can take $\sigma' = []$, and we have $\underline{\gamma}\sigma\sigma' = \tau\sigma$. Since $\mathrm{dom}(\sigma') = \emptyset$, it contains no unknowns. Hence, σ solves $\underline{\gamma} \leq \tau$. $\square$

Proposition 2 *If an unknown $\underline{\gamma}$ within an R-AUSUP instance Γ only occurs on right-hand sides within the inequalities in which it is an unknown, then $\underline{\gamma}$ may be simply treated as a variable, without affecting the principal solution (if one exists) of Γ; moreover $\underline{\gamma}$ is indistinguishable from an ordinary variable by the USUP redex procedure.*

Proof Let $\Gamma = \{\langle \tau_i \leq \mu_i, \vec{\alpha_i} \rangle\}_{i=1}^{N}$. Let Γ_S denote the R-ASUP instance derived from Γ by treating all unknowns as ordinary variables. Then a substitution σ solves Γ iff for all i, there is a substitution σ_i, such that $\tau_i\sigma\sigma_i = \mu_i\sigma$, and $\mathrm{dom}(\sigma_i) \cap \mathrm{Vars}(\vec{\alpha_i}\sigma) = \emptyset$, and a substitution σ_S solves Γ_S iff for all i, there is a substitution σ_{Si}, such that $\tau_i\sigma\sigma_{Si} = \mu_i\sigma$. We show that if σ_S is a principal solution for Γ_S, then σ_S solves Γ. Suppose, for some i, that $\underline{\gamma} \in \vec{\alpha_i}$, but $\underline{\gamma} \notin \mathrm{Vars}(\tau_i)$. If some variable $\alpha \in \mathrm{Vars}(\underline{\gamma}\sigma_S)$ has an occurrence in τ_i, then some variable $\beta \in \mathrm{Vars}(\tau_i)$ was at some point replaced, via redex reduction, by an expression containing α. Hence, $\beta R \alpha$ and $\alpha R \beta$. Also, we have $\underline{\gamma} R \alpha$ and $\alpha R \underline{\gamma}$. Now, if β has no occurrences on a right-hand side, then there is a solution σ'_S of Γ that does not replace β and is otherwise equivalent to σ in terms of its actions on right-hand sides. Assume, therefore, that β does have an occurrence on a right-hand side. Then $\beta R' \underline{\gamma}$, which gives $\beta R' \underline{\gamma} R \alpha R \beta$, which contradicts R-acyclicity. Hence, no variable in $\mathrm{Vars}(\underline{\gamma}\sigma_S)$ has an occurrence in τ_i. Therefore, σ_{Si} can be chosen such that its domain contains no occurrences of any variable in $\mathrm{Vars}(\underline{\gamma}\sigma)$, and we find that σ_S solves Γ. The reverse direction (i.e., that σ solves Γ_S) is trivial, as USUP is a superset of SUP.

For the second part of the claim, consider the actions of the four kinds of redex reduction on an unknown $\underline{\gamma}$ that only has occurrences on right-hand sides within inequalities in which it is an unknown. In the case of a redex-I, any occurrence $\underline{\gamma}$ on the right-hand side will be ignored, because it is not strictly a variable. Redex-II reductions do not distinguish between unknowns and variables; hence $\underline{\gamma}$ will be treated the same under redex-II reduction, whether or not it is labelled as an unknown. Redex-III reduction does not apply, as it only treats unknown occurrences on left-hand sides. Redex-IV reduction reduces an unknown occurrence on the right-hand side with a structural copy of the corresponding expression on the left-hand side—this is precisely the reduction performed under redex-I reduction for ordinary variables. Hence, redex-IV reduction makes up for the behaviour that is lacking in redex-I reduction, so that, indeed, the redex procedure treats γ in the same way as $\underline{\gamma}$. $\square$

Because of Proposition 2, for subexpressions E_i of the source program E, such that $\mathit{USUP}(E_i)$ only contains unknown occurrences on right-hand sides (in particular, after applying the convenience transformation afforded by Proposition 1), we can regard the USUP instance $\mathit{USUP}(E_i)$ as equivalent to the underlying SUP instance, and dispense with discussion of unknowns in the context of E_i. We call the simplifying assumptions afforded by Propositions 1 and 2 the *convenience assumptions*.

In the case of the edge in Figure 5, because β_y is *not* an unknown, the inequality $\beta_y \leq \delta_y$ is not equivalent to the equality $\beta_y = \delta_y$, and the R-acyclicity violation is not easily eliminated.

Note, however, that the R-acyclicity violation in the upper edge does not lead to non-termination. Instead it reduces as follows. Reduction of the redex-I in the last inequality (which then becomes solved) yields

$$\begin{array}{rcl} \alpha \to \alpha & \leq & \delta_{\lambda^p y.y} \to (\delta_{\lambda^m z.z} \to \delta_E) \\ \alpha \to \alpha & \leq & \delta_{\lambda^p y.y} \to (\beta_y \to \delta_y) \\ \beta_y & \leq & \delta_y \\ \alpha \to \alpha & \leq & \delta_{\lambda^m z.z} \to (\beta_z \to \beta_z) \ . \end{array}$$

Reduction of the redex-II in the fourth inequality (which then becomes solved) yields

$$\begin{array}{rcl} \alpha \to \alpha & \leq & \delta_{\lambda^p y.y} \to ((\beta_z \to \beta_z) \to \delta_E) \\ \alpha \to \alpha & \leq & \delta_{\lambda^p y.y} \to (\beta_y \to \delta_y) \\ \beta_y & \leq & \delta_y \ . \end{array}$$

Reduction of the redex-II in the second inequality (which then becomes solved) yields

$$\begin{array}{rcl} \alpha \to \alpha & \leq & (\beta_y \to \delta_y) \to ((\beta_z \to \beta_z) \to \delta_E) \\ \beta_y & \leq & \delta_y \ . \end{array}$$

Reduction of the redex-II in the first inequality (which then becomes solved) yields

$$\begin{array}{rcl} \alpha \to \alpha & \leq & ((\beta_z \to \beta_z) \to \delta_E) \to ((\beta_z \to \beta_z) \to \delta_E) \\ \beta_z \to \beta_z & \leq & \delta_E \ . \end{array}$$

Finally, a redex-I reduction yields

$$\begin{array}{rcl} \alpha \to \alpha & \leq & ((\beta_z \to \beta_z) \to (\beta \to \beta)) \to ((\beta_z \to \beta_z) \to (\beta \to \beta)) \\ \beta_z \to \beta_z & \leq & \beta \to \beta \ , \end{array}$$

at which point the entire instance is solved. The image of δ_E under the solution yields the final type, $\forall \beta . \beta \to \beta$, as expected. Thus, even though we do not have R-acyclicity for this problem instance, we do have termination.

In general, any time a polymorphic abstraction $E = \lambda^p x.M$ actually makes use of its argument x, this kind of R-acyclicity violation will occur. If E makes use of x, then there will be at least one inequality $\beta_x \leq \delta_x$ in $\mathit{USUP}(E)$. But then M's type will in general be dependent upon x's type, so that δ_x and δ_M are R-related. Further, because of the equality $\delta_E = \beta_x \to \delta_M$, δ_M and δ_x will be R'-related, thus yielding an R-cycle comprising at least one non-trivial path. Our task, then, is to show that these violations can never lead to non-termination.

It is worth pointing out that the original ASUP translation does not suffer from this acyclicity violation, because it never actually represents the type of a polymorphic abstraction in the translation; for an expression

$$(\lambda^p y_1.(\cdots((\lambda^p y_n.E_{n+1})E_n)\cdots)E_2)E_1 \ ,$$

the ASUP translation only translates each subexpression E_i into ASUP, and then adds additional equalities to match up the parameter and argument types for each λ^p-abstraction. The final result type is given as the image of E_{n+1} under the translation, and the final parameter types are added on at the end by prepending the bindings for each x_j in a user-supplied environment. By never explicitly including the types of the λ^p-abstractions in the translation, the ASUP translation avoids the kind of acyclicity violation we have encountered above; had they explicitly translated the polymorphic abstractions, the acyclicity violations (which violate not only R-acyclicity, but, as a consequence, ASUP-acyclicity as well) would indeed have surfaced.

To begin our analysis, we will assume that the expression E to be typed is θ-normal (i.e., in the form presented above, in which no expression E_i contains a redex), and then relax this assumption as our analysis progresses. Our first observation is the following:

Proposition 3 *For an θ-normal expression*

$$E = (\lambda^p y_1.(\cdots((\lambda^p y_n.E_{n+1})E_n)\cdots)E_2)E_1$$

in some environment, each USUP(E_i) is R-acyclic. Further, the concatenation of all USUP(E_i) is also R-acyclic.

Proof The proof follows immediately from the R-acyclicity of the original R-ASUP translation (proof can be found in [7])—aside from the introduction of unknowns (which have no effect on an instance's graph structure) and some variable name changes, the two translation procedures produce identical (U)SUP instances for each E_i. $\square$

Thus, termination is assured for the parts of $USUP(E)$ corresponding to each E_i. Now, for each redex $(\lambda^p y_i.M)E_i$, in addition to the inequalities in $USUP(E_i)$ and $USUP(M)$, we also have the following two inequalities:

$$\begin{aligned}\delta_{\lambda^p y_i.M} &= \delta_{E_i} \rightarrow \delta_{(\lambda^p y_i.M)E_i}\\ \delta_{\lambda^p y_i.M} &= \beta_{y_i} \rightarrow \delta_M \ .\end{aligned}$$

All other occurrences of β_{y_i} are in inequalities of the form $\beta_{y_i} \leq \delta_{y_i,k}$. In particular, all other occurrences of β_{y_i} are on left-hand sides, and none of them are unknown occurrences, as $\lambda^p y_i.M$ is a polymorphic abstraction.

Now, we claim that, if this R-acyclicity violation leads to an infinite reduction sequence, then that reduction sequence must include substitutions that replace β_{y_i}. To see this, consider the replacement of β_{y_i} in $\delta_{\lambda^p y_i.M} = \beta_{y_i} \rightarrow \delta_M$ (or indeed in each of the $\beta_{y_i} \leq \delta_{y_i,k}$) by some fresh variable β_0, so that the edge created by β_{y_i} is broken. Under the assumption that E_i and M are already R-acyclic, the graph as a whole then becomes R-acyclic, and termination is then assured. The only difference between this USUP instance and the original is the ability of a substitution to propagate across the edge between $\delta_{\lambda^p y_i.M} = \beta_{y_i} \rightarrow \delta_M$ and $\beta_{y_i} \leq \delta_{y_i,k}$, which the former possess and the latter lacks. Hence an infinite reduction sequence must involve the propagation of a substitution along one such edge. In particular, an infinite reduction sequence must involve a replacement of some β_{y_i}. But the redex procedure only replaces variables that have an occurrence on a right-hand side. Hence, a replacement of β_y can only take place via the two inequalities presented above. The effect, however, of such reductions, is that (after solved vertices are removed) the R-acyclicity violation disappears, and along with it, the potential for non-termination.

We formalize the situation as follows:

Proposition 4 *Let*

$$E = (\lambda^p y_1.(\cdots((\lambda^p y_n.E_{n+1})E_n)\cdots)E_2)E_1$$

be a θ-normal, labelled λ-term. Then the USUP redex procedure terminates on input USUP(E).

Proof For each $i \in \{1, \ldots, n+1\}$, let $\Gamma_i = \mathit{USUP}(E_i)$, and let Γ'_i be the corresponding underlying SUP instance. By the convenience assumptions, we may regard each Γ_i as equivalent to the corresponding Γ'_i. Then each Γ_i is R-acyclic, and so is the concatenation $\Gamma_1 \cdots \Gamma_{n+1}$. Hence, termination is assured for the instance $\Gamma_1 \cdots \Gamma_{n+1}$. To this instance, for each i, the instance $\mathit{USUP}(E)$ adds the following inequalities:

$$\begin{aligned} \delta_{\lambda^p y_i . M_i} &= \beta_{y_i} \to \delta_{M_i} \\ \delta_{\lambda^p y_i . M_i} &= \delta_{E_{i+1}} \to \delta_{(\lambda^p y_i . M_i) E_{i+1}} \ , \end{aligned}$$

for each λ^p.-abstraction, where $M_n = E_{n+1}$ and $M_i = (\lambda^p y_{i+1} . M_{i+1}) E_{i+1}$. If the redex procedure never reduces these inequalities, then they are effectively not present, and the procedure terminates. Hence, non-termination implies that the procedure must reduce at least one of these inequalities. Whatever the choice of i, however, there is only one possible replacement:

- for some value of i, the reduction replaces $\delta_{\lambda^p y_i . M_i}$. This reduction can either come from the inequality $\delta_{\lambda^p y_i . M_i} = \beta_{y_i} \to \delta_{M_i}$ or the inequality $\delta_{\lambda^p y_i . M_i} = \delta_{E_{i+1}} \to \delta_{(\lambda^p y_i . M_i) E_{i+1}}$. In the former case, we obtain the substitution $[\beta_{y_i} \to \delta_{M_i} / \delta_{\lambda^p y_i . M_i}]$, and in the latter case, we obtain the substitution $[\delta_{E_{i+1}} \to \delta_{(\lambda^p y_i . M_i) E_{i+1}} / \delta_{\lambda^p y_i . M_i}]$. Since the only two occurrences of $\delta_{\lambda^p y_i . M_i}$ are within these two inequalities, reduction either way renders the originating inequality solved, and the other inequality becomes $\beta_{y_i} \to \delta_{M_i} = \delta_{E_{i+1}} \to \delta_{(\lambda^p y_i . M_i) E_{i+1}}$. All other inequalities in $\mathit{USUP}(E)$ remain unchanged. Thus, an infinite reduction in $\mathit{USUP}(E)$ is still impossible, unless further reduction of one of the added inequalities takes place. If, for some i', the additional reduction replaces $\delta_{\lambda^p y_{i'} . M_{i'}}$, then the reduction proceeds as outlined here, and still the remainder of the instance is unchanged. The additional substitution reduces the newly-established inequality, $\beta_{y_i} \to \delta_{M_i} = \delta_{E_{i+1}} \to \delta_{(\lambda^p y_i . M_i) E_{i+1}}$, thereby yielding $[\delta_{E_{i+1}} / \beta_{y_i}, \delta_{(\lambda^p y_i . M_i) E_{i+1}} / \delta_{M_i}]$. By the end of this substitution, however, both of the inequalities $\delta_{\lambda^p y_i . M_i} = \beta_{y_i} \to \delta_{M_i}$ and $\delta_{\lambda^p y_i . M_i} = \delta_{E_{i+1}} \to \delta_{(\lambda^p y_i . M_i) E_{i+1}}$ will now be solved, and thus of no further interest. Furthermore, in the remainder of the instance, we will have equated β_{y_i} with $\delta_{E_{i+1}}$ and $\delta_{(\lambda^p y_i . M_i) E_{i+1}}$ with δ_{M_i}. This, however, is precisely the way in which the original R-ASUP translation treats β-redexes. Hence by the termination property for that translation, this reduction cannot produce a non-terminating USUP instance.

In summary, none of the additional inequalities from $\mathit{USUP}(E)$, over and above those in $\Gamma_1 \cdots \Gamma_{n+1}$ can bring about an infinite reduction, without requiring more of the additional inequalities to be reduced as well. Since they are only finite in number, we conclude that an infinite reduction in $\mathit{USUP}(E)$ is simply not possible. $\square$

Proposition 4 establishes termination for the USUP redex procedure on input $\mathit{USUP}(E)$, for any θ-normal, labelled expression E. We now expand on this result to accommodate expressions that are *not* θ-normal.

Proposition 5 *Let E be a labelled λ-term such that the USUP redex procedure terminates on input $\mathit{USUP}(E)$. Suppose $E' \to_{\theta_3} E$. Then the USUP redex procedure terminates on input $\mathit{USUP}(E')$.*

Proof There exists a context C, with a single hole, and subexpressions N and P, such that $E = C[(\lambda^p y.NP)Q]$, and $E' = C[N((\lambda^p y.P)Q)]$. The context being the same in both cases, it is sufficient to show termination for $(\lambda^p y.NP)Q$. Let us assume instead, therefore, that $E = (\lambda^p y.NP)Q$, and $E' = N((\lambda^p y.P)Q)$. Let $\mathit{USUP}_E(N)$ and $\mathit{USUP}_{E'}(N)$ denote, respectively, the USUP translation of N in the context of E and E', and similarly for P and Q. Since P and Q have the same scope in both E and E', we have $USUP_E(P) = USUP_{E'}(P)$ and $USUP_E(Q) = USUP_{E'}(Q)$. As for N, it lies within y's scope in E, but not in E'. Since E' is the source term, however, by our unique naming assumption, N can contain no occurrences of y; hence $USUP_E(N)$ contains no occurrences of an inequality of the form $\beta_y \leq \delta_{y_i}$ for some occurrence y_i of y. Thus, $USUP_{E'}(N) = USUP_E(N)$ as well. By termination for $USUP(E)$, the concatenated instance $USUP_E(N)\,USUP_E(P)\,USUP_E(Q)$, which is common to both $USUP(E)$ and $USUP(E')$, cannot give rise to an infinite reduction sequence. The following inequalities are unique to $USUP(E)$:

$$\begin{aligned}
\delta_{\lambda^p y.NP} &= \delta_Q \rightarrow \delta_{(\lambda^p y.NP)Q} \\
\delta_{\lambda^p y.NP} &= \beta_y \rightarrow \delta_{NP} \\
\delta_N &= \delta_P \rightarrow \delta_{NP}\ ,
\end{aligned}$$

and the following inequalities are unique to $USUP(E')$:

$$\begin{aligned}
\delta_N &= \delta_{(\lambda^p y.P)Q} \rightarrow \delta_{N((\lambda^p y.P)Q)} \\
\delta_{\lambda^p y.P} &= \delta_Q \rightarrow \delta_{(\lambda^p y.P)Q} \\
\delta_{\lambda^p y.P} &= \beta_y \rightarrow \delta_P\ .
\end{aligned}$$

Any reduction in $USUP(E')$ that gives rise to an infinite reduction sequence must occur as a result of reducing at least one of these three inequalities. Because the variable $\delta_{\lambda^p y.P}$ has no occurrences other than above, we can replace it and thereby simplify the remaining inequalities:

$$\begin{aligned}
\delta_N &= \delta_{(\lambda^p y.P)Q} \rightarrow \delta_{N((\lambda^p y.P)Q)} \\
\beta_y \rightarrow \delta_P &= \delta_Q \rightarrow \delta_{(\lambda^p y.P)Q}\ .
\end{aligned}$$

If we reduce the second inequality, we obtain the replacement $[\delta_Q/\beta_y, \delta_{(\lambda^p y.P)Q}/\delta_P]$. The replacement $[\delta_Q/\beta_y]$ is available in $USUP(E)$, by reducing the first two inequalities presented above; hence, this replacement cannot lead to non-termination. The variable $\delta_{(\lambda^p y.P)Q}$ only has occurrences within the above inequalities; hence replacing it can neither single-handedly cause nor prevent non-termination in the expression as a whole. The replacement yields a single remaining unsolved inequality:

$$\delta_N = \delta_P \rightarrow \delta_{N((\lambda^p y.P)Q)}\ .$$

If there is to be a non-terminating reduction sequence in $USUP(E')$, then it must be a reduction of this inequality—in particular, the replacement $[\delta_P \rightarrow \delta_{N((\lambda^p y.P)Q)}/\delta_N]$—that creates it. Within $USUP(E)$, the inequality $\delta_N = \delta_P \rightarrow \delta_{NP}$ gives rise to the similar replacement, $[\delta_P \rightarrow \delta_{NP}/\delta_N]$. The difference between these is merely that the replacement for E' has an occurrence of $\delta_{N((\lambda^p y.P)Q}$, where the replacement for E has an occurrence of δ_{NP}. This difference is not surprising, as E does not contain $N((\lambda^p y.P)Q)$ as a subexpression, and E' does not contain NP as a subexpression. However, these two variables actually denote the same type. As we have seen via the replacement $[\delta_{(\lambda^p y.P)Q}/\delta_P]$, the variables $\delta_{(\lambda^p y.P)Q}$ and δ_P have the same value in $USUP(E')$. Further, every reduction we have performed in E' has also been available in E; thus, if σ represents the substitutions performed so far, then $\delta_P\sigma$ has the same value in both $USUP(E)$ and $USUP(E')$. Therefore, $\delta_P\sigma$ in $USUP(E)$ is equal to $\delta_{(\lambda^p y.P)Q}\sigma$ in $USUP(E')$. Similarly, $\delta_N\sigma$ has the same value in both $USUP(E)$ and $USUP(E')$ (before reduction of this last inequality). Therefore, $N((\lambda^p y.P)Q)$ has the same type in E' as NP has in E, and $\delta_{N((\lambda^p y.P)Q)}$ in $USUP(E')$ is equal to δ_{NP} in $USUP(E)$. Hence, the reduction $[\delta_P \rightarrow \delta_{N((\lambda^p y.P)Q}/\delta_N]$ is performed in E, though using different names, and therefore, by termination in $USUP(E)$, the instance $USUP(E')$ must terminate as well. $\square$

Thus, an expression that is some number of θ_3-reductions away from θ-normal form still gives rise to a terminating USUP instance. We continue in a similar vein for the other forms of θ-reduction.

Proposition 6 *Let E be a labelled λ-term such that the USUP redex procedure terminates on input $USUP(E)$. Suppose $E' \rightarrow_{\theta_1} E$. Then the USUP redex procedure terminates on input $USUP(E')$.*

Proof As in Proposition 5, we can ignore the surrounding context, and simply assume that $E = (\lambda^p y.NQ)P$, and $E' = ((\lambda^p y.N)P)Q$. Let $USUP_E(N)$ and $USUP_{E'}(N)$ denote, respectively, the USUP translation of N in the context of E and E', and similarly for P and Q. Since N and P have the same scope in both E and E', we have $USUP_E(N) = USUP_{E'}(N)$ and $USUP_E(P) = USUP_{E'}(P)$. As for Q, it lies within y's scope in E, but not in E'. Since E' is the source term, however, by our unique naming assumption, Q can contain no occurrences of y; hence $USUP_E(Q)$ contains no occurrences of an inequality of the form $\beta_y \leq \delta_{y_i}$ for some occurrence y_i of y. Thus, $USUP_{E'}(Q) = USUP_E(Q)$ as well. By termination for $USUP(E)$, the concatenated instance $USUP_E(N)\, USUP_E(P)\, USUP_E(Q)$, which is common to both $USUP(E)$ and $USUP(E')$, cannot give rise to an infinite reduction sequence. The following inequalities are unique to $USUP(E)$:

$$\begin{aligned}
\delta_{\lambda^p y.NQ} &= \delta_P \rightarrow \delta_{(\lambda^p y.NQ)P} \\
\delta_{\lambda^p y.NQ} &= \beta_y \rightarrow \delta_{NQ} \\
\delta_N &= \delta_Q \rightarrow \delta_{NQ}\ .
\end{aligned}$$

and the following inequalities are unique to $USUP(E')$:

$$\begin{aligned}
\delta_{(\lambda^p y.N)P} &= \delta_Q \rightarrow \delta_{((\lambda^p y.N)P)Q} \\
\delta_{\lambda^p y.N} &= \delta_P \rightarrow \delta_{(\lambda^p y.N)P} \\
\delta_{\lambda^p y.N} &= \beta_y \rightarrow \delta_N\ .
\end{aligned}$$

Any reduction in $USUP(E')$ that gives rise to an infinite reduction sequence must occur as a result of reducing at least one of these three inequalities. As before, because the variables $\delta_{\lambda^p y.N}$ and $\delta_{(\lambda^p y.N)P}$ have no occurrences other than above, we can replace them and thereby simplify the remaining inequalities:

$$\beta_y \rightarrow \delta_N \quad = \quad \delta_P \rightarrow (\delta_Q \rightarrow \delta_{((\lambda^p y.N)P)Q})\ .$$

If there is to be a non-terminating reduction sequence in $USUP(E')$, therefore, it must be a reduction of this inequality—in particular, the replacement $[\delta_P/\beta_y, \delta_Q \rightarrow \delta_{((\lambda^p y.N)P)Q}/\delta_N]$—that creates it. The replacement $[\delta_P/\beta_y]$ arises from reducing the first two inequalities particular to $USUP(E)$, presented above. Hence, by termination for $USUP(E)$, this replacement cannot, on its own, lead to non-termination. Thus, non-termination can only come from the replacement $[\delta_Q \rightarrow \delta_{((\lambda^p y.N)P)Q}/\delta_N]$. Within $USUP(E)$, there is the similar replacement $[\delta_Q \rightarrow \delta_{NQ}/\delta_N]$, arising from reduction of the third inequality. From the inequalities for $USUP(E')$, we can derive $\delta_N = \delta_{(\lambda^p y.N)P}$. Then by the same reasoning as in the proof of Proposition 5, the reduced value of $\delta_{((\lambda^p y.N)P)Q}$ in $USUP(E')$ is equal to that of δ_{NQ} in $USUP(E)$. Therefore, a replacement equivalent to $[\delta_Q \rightarrow \delta_{((\lambda^p y.N)P)Q}/\delta_N]$ is performed in $USUP(E)$. Hence, by the assumed termination for $USUP(E)$, we conclude that the redex procedure must terminate on input $USUP(E')$. □

Proposition 7 *Let E be a labelled λ-term such that the USUP redex procedure terminates on input $USUP(E)$. Suppose $E' \rightarrow_{\theta_2} E$. Then the USUP redex procedure terminates on input $USUP(E')$.*

Proof As in Proposition 5, we can ignore the surrounding context, and simply assume that $E = (\lambda^p v.\lambda^m z.N')(\lambda^m w.P')$, and $E' = \lambda^m z.(\lambda^p y.N)P$, where $N' = N[vz/y]$ and $P' = P[w/z]$. Let $USUP_E(N')$ and $USUP_{E'}(N)$ denote, respectively, the USUP translation of N' in the context of E and N in the context of E', and similarly for P and P'. In the context of E', P lies within the scope of the monomorphic variable z; hence the variable β_z is unknown throughout $USUP_{E'}(P)$. In the context of E, P lies within the scope of the monomorphic variable w, so that the variable β_w is unknown throughout $USUP_E(P)$. Thus, since $P' = P[w/z]$, we have that $USUP_E(P')$ and $USUP_{E'}(P)$ are the same, except that every occurrence of β_z in $USUP_{E'}(P)$ becomes β_w in $USUP_E(P')$, and similarly for each δ_{z_i} and δ_{w_i}, accounting for each respective occurrence z_i of z in E' and w_i of w in E. Thus, modulo these variable renamings, the result of running the USUP redex procedure on input $USUP_{E'}(P)$ is identical to that of running it on input $USUP_E(P')$. More importantly, termination for E implies that the USUP redex procedure will not fall into an infinite reduction on input $USUP_{E'}(P)$. Similarly, $USUP_{E'}(N)$ and $USUP_E(N')$ are the same, except that for each occurrence y_i of y in N, the corresponding inequality $\beta_y \leq \delta_{y_i}$ in $USUP_{E'}(N)$ becomes the trio of inequalities

$$\begin{array}{rcl} \delta_{v_i} & = & \delta_{z_i} \rightarrow \delta_{v_i z_i} \\ \beta_v & \leq & \delta_{v_i} \\ \underline{\beta_z} & \leq & \delta_{z_i} \end{array}$$

in $USUP_E(N')$. (Note that β_z is unknown throughout $USUP_{E'}(N)$.) Further, any remaining occurrences of δ_{y_i} in $USUP_{E'}(N)$ become $\delta_{v_i z_i}$ in $USUP_E(N)$. Thus, if any reduction in $USUP_{E'}(N)$ is to cause non-termination, it can only be the inequality $\beta_y \leq \delta_{y_i}$, as this is the only inequality particular to $USUP_{E'}(N)$. However, this inequality, as it currently exists, contains no redexes. Further, as N lies entirely within y's scope, not reduction in $USUP_{E'}(N)$ will cause a replacement of β_y; hence a redex will not be induced in $\beta_y \leq \delta_{y_i}$ as a result of reduction in $USUP_{E'}(N)$. Hence, the USUP redex procedure terminates on $USUP_{E'}(N)$.

Now, in addition, the following inequalities are particular to $USUP(E)$:

$$\begin{array}{rcl} \delta_{\lambda^p v.\lambda^m z.N'} & = & \delta_{\lambda^m w.P} \rightarrow \delta_E \\ \delta_{\lambda^p v.\lambda^m z.N'} & = & \beta_v \rightarrow \delta_{\lambda^m z.N'} \\ \delta_{\lambda^m z.N'} & = & \beta_z \rightarrow \delta_{N'} \\ \delta_{\lambda^m w.P'} & = & \beta_w \rightarrow \delta_{P'} \ , \end{array}$$

and the following inequalities are particular to $USUP(E')$:

$$\begin{array}{rcl} \delta_{E'} & = & \beta_z \rightarrow \delta_{(\lambda^p y.N)P} \\ \delta_{\lambda^p y.N} & = & \delta_P \rightarrow \delta_{(\lambda^p y.N)P} \\ \delta_{\lambda^p y.N} & = & \beta_y \rightarrow \delta_N \ . \end{array}$$

Thus, if non-termination is to arise, it must come as a result of reducing one or more of these three latter inequalities. The variable $\delta_{\lambda^p y.N}$ has no occurrences outside these three inequalities; hence reducing it cannot cause non-termination without further reduction. We therefore replace it in the above inequalities to obtain the simpler system

$$\begin{array}{rcl} \delta_{E'} & = & \beta_z \rightarrow \delta_{(\lambda^p y.N)P} \\ \beta_y \rightarrow \delta_N & = & \delta_P \rightarrow \delta_{(\lambda^p y.N)P} \ . \end{array}$$

The variable $\delta_{E'}$ has no other occurrences in $USUP(E')$; hence, replacing it via the reduction in the first inequality has no effect on the rest of the instance, and in particular, no effect on termination. Thus, if non-termination is to result, it must come as a result of the replacement that arises from reducing the inequality $\beta_y \to \delta_N = \delta_P \to \delta_{(\lambda^p y.N)P}$—namely, $[\delta_P/\beta_y, \delta_{(\lambda^p y.N)P}/\delta_N]$. The effect of the replacement $[\delta_P/\beta_y]$ is to replace every inequality $\beta_y \leq \delta_{y_i}$ in $USUP(E')$ with $\delta_P \leq \delta_{y_i}$; more precisely, if σ denotes the replacements performed so far, then the inequalities become $\delta_P\sigma \leq \delta_{y_i}\sigma$. The effect of this replacement is the possible creation of additional redex-I's in these inequalities. Within $USUP(E)$, reducing the first two inequalities presented above gives $\beta_v = \delta_{\lambda^m w.P}$. Performing this replacement has an effect on the inequalities

$$\begin{array}{rcl} \delta_{v_i} & = & \delta_{z_i} \to \delta_{v_i z_i} \\ \beta_v & \leq & \delta_{v_i} \\ \underline{\beta_z} & \leq & \delta_{z_i} \end{array}$$

for each occurrence v_i of v. Specifically, if σ denotes the replacements performed so far, then each inequality $\beta_v \leq \delta_{v_i}$ becomes $\delta_{\lambda^m w.P'}\sigma \leq \delta_{v_i}\sigma$. Then, since $\delta_{\lambda^m w.P'}$ reduces to $\beta_w \to \delta_{P'}$, we obtain $\beta_w\sigma \to \delta_{P'}\sigma \leq \delta_{v_i}\sigma$. Then by the first inequality ($\delta_{v_i} = \delta_{z_i} \to \delta_{v_i z_i}$), we get $\beta_w\sigma \to \delta_{P'}\sigma \leq \delta_{z_i}\sigma \to \delta_{v_i z_i}\sigma$. Since β_w replaces β_y in $USUP(E)$, and $\delta_{v_i z_i}$ replaces δ_{y_i} in $USUP(E)$, the replacement of what stands for β_y with what stands for δ_P (i.e., $\delta_{P'}$) is reflected in $USUP(E)$. Note, however, that $\underline{\beta}_z$ is an unknown in $USUP(E')$; hence, any occurrence of δ_{z_i} in $USUP(E')$, representing the corresponding occurrence z_i of z in E', is also unknown. Hence, after the replacement $[\delta_P/\beta_y]$, the subexpressions in each $\delta_{y_i}\sigma$ corresponding to unknowns in $\delta_P\sigma$ themselves become unknown. Within $USUP(E)$, the occurrences of δ_{z_i} in $USUP(E')$ correspond to occurrences of δ_{w_i}, which are also unknown. After the replacement $[\delta_{\lambda^m w.P'}/\beta_v]$, we eventually obtain $\beta_w\sigma \to \delta_{P'}\sigma \leq \underline{\beta_z}\sigma \to \delta_{v_i z_i}\sigma$. However, the occurrences of $\delta_{z_i}\sigma$ in $\delta_{P'}\sigma$, which were unknown within $USUP_{E'}(P)$, are no longer unknown at this higher scope. On the other hand, each occurrence of $\delta_{v_i}\sigma$ is matched with $\delta_{P'}\sigma$ and then applied to the monomorphic variable z, whose occurrences δ_{z_i} in $USUP(E)$ are all unknown. In particular, the expression

$$(\lambda^p v.\lambda^m z.N[vz/y])(\lambda^m w.P[w/z])$$

gives rise to the following inequalities involving the variable β_v:

$$\begin{array}{rcl} \beta_v & = & \beta_w \to \delta_{P[w/z]} \\ \beta_v & = & \underline{\beta_z} \to \delta_{N[vz/y]} \ . \end{array}$$

Thus, we obtain $\beta_w = \underline{\beta_z}$, so that β_w becomes an unknown. Then for each occurrence w_i of w, we obtain the inequality

$$\underline{\beta_w} \leq \delta_{w_i} \ ,$$

from which each δ_{w_i} becomes an unknown, and in particular, each δ_{w_i} is equal to $\underline{\beta_w}$. Hence, the occurrences of $\delta_{w_i}\sigma$ get matched with unknowns and become unknowns once again. Then occurrences of $\delta_{v_i z_i}$ corresponding to these unknowns in $\delta_{P'}$ are also unknown. Thus, the behaviour of the reduction $[\delta_P/\beta_y]$ in $USUP(E')$ is exactly reflected in $USUP(E)$. Hence, by termination for $USUP(E)$, this replacement cannot result in non-termination for $USUP(E')$. The remaining possibility is the replacement $[\delta_{(\lambda^p y.N)P}/\delta_N]$. The variable $\delta_{(\lambda^p y.N)P}$, however, only has occurrences in the three inequalities particular to $USUP(E')$, and the variable δ_N only has a single occurrence in $USUP_{E'}(N)$. Furthermore, this single occurrence is on a right-hand side. Therefore, it can only induce a redex if if occurs in a position corresponding to an unknown on the left-hand side. Since the single inequality in which δ_N occurs within $USUP_{E'}(N)$ is an *equality*, however, the identifiers on the left-hand side are actual variables, and therefore, no redex in $USUP_{E'}(N)$ is induced by the replacement of δ_N by $\delta_{(\lambda^p y.N)P}$, or vice versa. Thus, this replacement cannot bring about non-termination either, and we conclude, from termination in $USUP(E)$, that termination in $USUP(E')$ is guaranteed as well. $\square$

What we have shown, by Propositions 5, 6, and 7, is that if we start with any expression E for which $USUP(E)$ is guaranteed to make the USUP redex procedure terminate, then any step of θ-expansion will yield an expression E', for which $USUP(E')$ also makes the USUP redex procedure terminates. By iterating these results over several steps of θ-expansion, we have the following result:

Theorem 10 (Termination) *For any labelled λ-term E, the USUP redex procedure will terminate on input $USUP(E)$.*

Proof Every labelled λ-term E is a finite number of θ-reductions from a θ-normal form E_θ. By Proposition 4, the USUP redex procedure terminates on input $USUP(E_\theta)$. Then by Propositions 5, 6, and 7, each θ-expansion of E_θ yields a USUP instance for which the redex procedure terminates. Further θ-expansions then yield further terminating USUP instances. Eventually, θ-expansion produces the original term E, at which point we conclude that, indeed, the USUP redex procedure terminates on input $USUP(E)$. $\square$

Theorem 10 is the main result we sought in this section—that, like the ASUP and R-ASUP translation procedures we discussed previously, the USUP translation procedure also produces problem instances (now USUP problem instances) for which the redex procedure (now the USUP redex procedure) is guaranteed to terminate.

8 Implementation

An implementation of USUP can be found at `http://plg.uwaterloo.ca/~bmlushma/usup/`.

9 Related Work

Constraint solving has become a popular mechanism for implementing type inference in ML-like languages. An overview of constraint-solving techniques and their advantages can be found in Pottier and Rémy [16]. Comon [2] gives a survey of earlier work on constraint systems. We outline some well-known and recent contributions here.

Odersky and Läufer's annotation-based type inference system for System F [11] is based on translation to a system of constraints (based on unification under a mixed prefix [9]). The recent adoption into the Glasgow Haskell Compiler of programmer-assisted higher-rank type inference is based, in part, on their ideas [14].

Odersky, Sulzmann, and Wehr's HM(X) [12] is a well-known, generalized constraint system that, in its simplest formulation, is equivalent to the type system of ML, but is easily extended to accommodate richer type systems. The language of constraints in HM(X) resembles that of predicate logic, and obey a rather complex set of algebraic equivalences. HM(X) has been extended and generalized in several works over the years (e.g., [17]).

Pierce and Turner's system of local type inference [15] is a System for eliminating certain type annotations from higher-order, typed languages; it is based on accumulating type information in localized areas of a program as constraints, which are then passed to a constraint solver to complete the type inference process. This system is extended in later work to accommodate a richer set of types [4] and non-local propagation of partial type information [13].

Historically, the association between semiunification and type inference has often been in negative contexts. Polymorphic recursion, as found in the Milner-Mycroft calculus [10] was shown to be equivalent to SUP, which was subsequently shown to be undecidable. Similarly, type inference for System F was proved undecidable via its connections to semiunification. A subset of System F that corresponds to a modest generalization of ML was shown to be equivalent to ASUP [6], and therefore admitted decidable type inference. However, technical limitations (some of which are relieved by this work) hindered the use of this system in practice. Semiunification has, however, appeared in other program analysis contexts. For example, Birkedal and Tofte [1] employ a constraint system for region inference that is similar to semiunification.

Perhaps for this reason, the use of semiunification as the foundational constraint system in polymorphic type inference for ML and related languages remains largely unexplored. However, semiunification offers certain clear advantages over many of the other systems discussed here. Chief among these is simplicity. The correspondence between a source program and its corresponding ASUP or USUP instance is clear, and SUP-like problems are solved via the application of only a small number of rules. Furthermore, with the advent of USUP, as presented in this paper, the severe technical limitations of the original SUP translation for ML programs (in particular, non-syntax-directedness and the need for θ-reduction) no longer exist.

It remains to be seen whether SUP and USUP are flexible enough to be easily extended to accommodate more sophisticated type systems, as is the case with HM(X). This question represents a definite avenue for further research.

10 Conclusion

Motivated by the desire to formulate a truly syntax-directed translation from ML typability to a SUP-like problem, we found in this paper that SUP on its own is inadequate for expressing the difference in behaviour between variables with monomorphic binding occurrences and those with polymorphic binding occurrences. Rather, the original SUP-based translation relies heavily on an expression being translated to θ-normal form before the translation begins. Once a term is in this form, the differences in behaviour between these two classes of variables do not manifest themselves and the SUP translation suffices.

If we truly want a syntax-directed translation, however, we need a SUP-like problem to act as the target of the translation, in which there is some accommodation for monomorphic type variables. In this paper, we presented USUP, which is an extension of SUP to include a new class of type variable, which we called *unknown*, whose purpose is precisely to capture the behaviour of monomorphic variables. We presented a solution semi-procedure for USUP, analogous to the redex procedure for SUP, and showed it to be sound and complete (when it terminates) with respect to the definition of a USUP solution. We showed termination for all solvable instances, and for the USUP analogue of R-acyclicity, which we called R-AUSUP.

By making USUP the target of a new translation procedure from typability, we obtain a procedure that is truly syntax-directed, and affords us several other simplifications over the original procedure as well. The new translation procedure is sound and complete with respect to the type rules for ML, and always produces terminating USUP instances.

By virtue of being syntax-directed, the USUP translation procedure provides simply one rule for every element of abstract syntax. The translation procedure now has an elegant, highly compact presentation, as presented in Figure 3. Compared to the original, the new translation is certainly easier to understand, and to state.

Furthermore, by virtue of being syntax-directed, there is a reverse mapping between individual inequalities in a USUP instance $USUP(E)$ and the elements of syntax in E that generated them. Thus, if an error is found in some inequality (say, an occurs-check violation) that renders the instance unsolvable, then the error can be traced directly back to an element of syntax in the original expression and then reported to the user. A similar error-reporting mechanism for the original SUP translation, while not technically impossible, would require a significant amount of bookkeeping, and be far from straightforward.

References

[1] Lars Birkedal and Mads Tofte. A constraint-based region inference algorithm. *Theoretical Computer Science*, 258:299–392, 2001.

[2] Hubert Comon. Constraints in term algebras (short survey). In *Conference on Algebraic Methodology and Software Technology (AMAST)*, Workshops in Computing, pages 97–108. Springer-Verlag, 1994, June 1994.

[3] Luis Damas and Robin Milner. Principal type-schemes for functional programs. In *Proceedings of the 9th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 207–212. ACM Press, 1982.

[4] Joshua Dunfield and Frank Pfenning. Tridirectional typechecking. In *Proceedings of the 31st ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 281–292. Association for Computing Machinery, ACM Press, 2004.

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

[6] 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.

[7] Brad Lushman and Gordon V. Cormack. A *more* direct algorithm for type inference in the rank-2 fragmentof the second-order λ-calculus. Technical Report CS2006-08, University of Waterloo Cheriton School of Computer Science, March 2006.

[8] Brad Lushman and Gordon V. Cormack. A larger decidable semiunification problem. In *Proceedings of the 9th International ACM SIGPLAN Symposium on Principles and Practice of Declarative Programming (PPDP '07)*, pages 143–152. ACM Press, 2007.

[9] Dale Miller. Unification under a mixed prefix. *Journal of Symbolic Computation*, 14:321–358, 1992.

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

[11] Martin Odersky and Konstantin Läufer. Putting type annotations to work. In *Proceedings of the 23rd ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 54–67. ACM Press, 1996.

[12] Martin Odersky, Martin Sulzmann, and Martin Wehr. Type inference with constrained types. *Theory and Practice of Object Systems*, 5(1):35–55, January/March 1999.

[13] Martin Odersky, Chritoph Zenger, and Matthias Zenger. Colored local type inference. In *ACM Symposium on Principles of Programming Languages*, pages 41–53, New York, NY, USA, 2001. ACM Press.

[14] Simon Peyton Jones and Mark Shields. Practical type inference for arbitrary-rank types. Submitted to *Journal of Functional Programming*, April 2004.

[15] Benjamin C. Pierce and David N. Turner. Local type inference. In *Proceedings of the 25th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 252–265. Association for Computing Machinery, ACM Press, January 1998.

[16] François Pottier and Didier Rémy. The essence of ML type inference. In Benjamin C. Pierce, editor, *Advanced Topics in Types and Programming Languages*, chapter 10, pages 389–489. The MIT Press, Cambridge, Massachusetts, 2005.

[17] Vincent Simonet and François Pottier. A constraint-based approach to guarded algebraic data types. *ACM Transactions on Programming Languages and Systems*, 29(1):1–56, January 2007.