Freeness, Sharing, Linearity and Correctness — All at Once[*]

M. Bruynooghe [**] M. Codish [***]

Department of Computer Science
K.U. Leuven, Belgium
email: maurice@cs.kuleuven.ac.be

Abstract. The availability of freeness and sharing information for logic programs has proven useful in a wide variety of applications. However, deriving correct freeness and sharing information turns out to be a complex task. In a recent paper, Codish *et al.* introduce a domain of abstract equation systems and illustrate the systematic derivation of an abstract unification algorithm for sharing and freeness. This paper follows the same approach, and using a more detailed abstraction, develops a more powerful transition system. The resulting algorithm is more prone to efficient implementation as it computes a single abstract solved form which exhibits sharing, freeness, groundness and linearity information.

1 Introduction

The aim of possible sharing analysis of a logic program P is to identify independent variables in a clause of P. A set of variables is said to be *independent* if no computation binds them to terms which contain a common variable. Groundness, linearity and freeness analyses aim to identify program variables which are respectively bound to ground terms, linear terms (a term is linear if no variable occurs in it more than once) and variable terms by all computations of P. The availability of groundness and linearity information is useful for improving the precision of sharing analyses. The availability of sharing and freeness information has proven useful in a wide range of applications including parallelisation of programs, compiler optimisations as well as for improving the precision of other analyses.

We focus here on semantic based analyses such as those specified within the framework of abstract interpretation [5]. In this approach a program analysis is viewed as a non-standard semantics defined over a domain of data-descriptions. Analyses are constructed by replacing the basic operations on data in a suitable concrete semantics with corresponding abstract operations defined on data-descriptions. Formal justification is reduced to proving conditions on the relation between data and data-descriptions and on the elementary operations defined on the data-descriptions. This approach eases both the development and the justification of program analyses. In the case of logic programming languages, proving the correctness of an abstract unification function is the major step in justifying an analysis.

[*] Funded in part by the ESPRIT project 5246 PRINCE.
[**] Supported by the Belgian National Fund for Scientific Research.
[***] From Oct. 93 at Dept. of Math. & Comp. Sci., Ben Gurion Univ., Beer-Sheba, Israel.

In its short history, the design and formal justification of sharing and freeness analyses for logic programs has proven to be a surprisingly difficult task. The first formally justified abstract unification algorithm for the "pair-sharing" domain proposed by Søndergaard in 1986 [10] was not given until 1991 in [2]. Many of the combined sharing and freeness analyses proposed in the literature have since been found incorrect in various details. In a recent paper [1], Codish *et al.* introduce a novel domain of abstract equation systems and illustrate the derivation of an abstract unification algorithm for sharing and freeness together with its formal justification.

The basic strategy applied in [1] is to systematically mimic each step in the Martelli – Montanari unification algorithm [8]. However, since an abstract equation may describe different concrete equations, to which different concrete rules apply, the resulting abstract algorithm is non-deterministic. The abstract equation is reduced by different abstract rules to mimic each of the corresponding concrete rules. Consequently different abstract solved forms may be obtained, all of which must be considered: a variable is definitely free only if it is free in all solved forms. This makes it doubtful whether it can be the basis for a very efficient analysis. Moreover, it should be enhanced to consider linearity information to improve its precision.

In this paper, we take the systematic development of [2] and [1] one step further. Choosing a different abstraction of equations, which distinguishes between abstract equations and a sharing component, we formalise a transition system which computes a single solved form which captures possible sharing and definite freeness information together with groundness and linearity. In addition to concrete terms, abstract equations also involve abstract variables which are no more than symbols from a designated set. For the sharing component we adopt the domain of Jacobs and Langen [6] which captures possible sharing as well as covering information and provides a good propagation of groundness information. Abstract variables are annotated to capture additional freeness and linearity information.

Here we do not require the abstract unification algorithm to mimic all possible steps in the concrete algorithm. Instead, we exploit the confluence of concrete unification to mimic a particular strategy on the concrete level which is determined by the structure of the equations on the abstract level. Each abstract rewrite rule is associated with an equivalence preserving transformation on concrete equations. Correctness follows by showing that each abstract transition mimics the corresponding transformation. The abstract unification algorithm is deterministic in the sense that at most one rule applies to a given abstract equation in a system. While not confluent, correctness holds for any sequence of selected equations.

We assume the reader is familiar with the basic concepts of abstract interpretation of logic programs and understands that abstract unification is the key step in developing a semantic based analysis of logic programs.

2 Preliminaries

Let Σ and *Var* denote respectively a fixed set of function symbols and an enumerable set of variables. The term algebra over Σ and *Var* is denoted Term. We reserve the symbols T and S to denote elements of Term. Variables are typically denoted U, V, W, X, Y, Z. The predicates *nonvar*(T), *var*(T), *ground*(T) and *linear*(T) denote respectively that T is non-variable, variable, ground and linear. Note that a free variable and a ground term are always linear. The power set of S is denoted $\wp S$.

A (concrete) equation system is a set of equations of the form $\mathrm{T}_1 = \mathrm{T}_2$. Given an equation system Eqs and an equation e, we let e :: Eqs denote the set $\{e\} \cup$ Eqs. We fix a partial function *mgu* which maps an equation system Eqs to a solved form *mgu*(Eqs). A reference to *mgu*(Eqs) implicitly implies that Eqs is satisfiable. We do not distinguish between equations in solved form and idempotent substitutions as the correspondence between them is well known (see for example [7]). We often view *mgu*(Eqs) as an idempotent substitution and write $eq(\theta)$ to denote the set of equations corresponding to an idempotent substitution θ. Equation systems Eqs_1 and Eqs_2 are said to be *equivalent with respect to a set of variables* V, denoted $\mathsf{Eqs}_1 \approx_V \mathsf{Eqs}_2$, if there exist most general unifiers θ_1 and θ_2 of Eqs_1 and Eqs_2 such that $\theta_1 \restriction V = \theta_2 \restriction V$.

Example 1 Let $\mathsf{Eqs}_1 = \{X = f(Y)\}$, $\mathsf{Eqs}_2 = \{X = f(Z), Y = Z\}$ and $\mathsf{Eqs}_3 = \{X = f(Y), Z = Y\}$. Then $\mathsf{Eqs}_1 \approx_V \mathsf{Eqs}_2$ where $V = \{X, Y\}$. Indeed $\theta_1 \restriction V = \theta_2 \restriction V$ for $\theta_1 = \{X/f(Y)\}$ and $\theta_2 = \{X/f(Y), Z/Y\}$ which are most general unifiers of Eqs_1 and Eqs_2. Also $\mathsf{Eqs}_1 \approx_V \mathsf{Eqs}_3$ because θ_2 is also a most general unifier of Eqs_3.

In addition to concrete terms and variables, we assume a disjoint and enumerable set AVar of abstract variables. Intuitively an abstract variable represents a term from Term. We reserve the symbol A to denote abstract variables. The term algebras over Σ and AVar and over Σ and *Var* $\cup$ AVar are denoted respectively ATerm (abstract terms) and MTerm (mixed terms). The sets of variables and abstract variables occurring in a syntactic object s are respectively denoted $vars(s)$ and $avars(s)$.

Definition 2 (abstract equation system) *An abstract equation system consists of a pair* $\langle \mathsf{AEqs}, \Delta \rangle$ *where* AEqs *is a set of abstract equations of the form* $\mathrm{T}_1 = \mathrm{T}_2$ *or of the form* $\mathrm{T} = \mathrm{A}$ *and* $\Delta \in \wp\wp\, avars(\mathsf{AEqs})$ *is a sharing component with an associated annotation mapping* $ann_\Delta : avars(\Delta) \rightarrow \{\{f, \ell\}, \{\ell\}, \emptyset\}$.

Notice that for the time being we avoid abstract equations of the forms $\mathrm{A}_1 = \mathrm{A}_2$, $\mathrm{A} = \mathrm{T}$ and abstract equations involving mixed terms. While these extensions can easily be given a meaning, they complicate the abstract unification algorithm.

We follow Jacobs and Langen [6] in the representation of the sharing component. Each set $\{\mathrm{A}_1, \ldots \mathrm{A}_n\}$ indicates that the terms represented by the abstract variables $\mathrm{A}_1, \ldots, \mathrm{A}_n$ can share one or more variables. Moreover, the annotations on abstract variables enrich the domain with linearity information as captured by the sharing domain of Søndergaard [10] and with freeness information. Let AS $= \langle \mathsf{AEqs}, \Delta \rangle$ be an abstract equation system. We define the following predicates on the abstract variables occurring in AS:

- $ground_\Delta(\mathrm{A}) \Leftrightarrow \mathrm{A} \notin avars(\Delta)$;
- $free_\Delta(\mathrm{A}) \Leftrightarrow ann_\Delta(\mathrm{A}) = \{f, \ell\}$;
- $linear_\Delta(\mathrm{A}) \Leftrightarrow ann_\Delta(\mathrm{A}) = \{\ell\}$;
- $share_\Delta(\mathrm{A}_1, \mathrm{A}_2) \Leftrightarrow \exists S \in \Delta.\ \{\mathrm{A}_1, \mathrm{A}_2\} \subseteq S$.

Basically, AS describes a concrete equation system Eqs if it is possible to map each of the abstract variables in AEqs to a concrete term and obtain Eqs. However, the mapping is required to preserve the information specified by Δ. We say that a mapping $\alpha : avars(\mathsf{AS}) \rightarrow$ Term is Δ *preserving* if: (1) $ground_\Delta(\mathrm{A}) \Rightarrow ground(\alpha(\mathrm{A}))$; (2) $linear_\Delta(\mathrm{A}) \Rightarrow linear(\alpha(\mathrm{A}))$; (3) $free_\Delta(\mathrm{A}) \Rightarrow free(\alpha(\mathrm{A}))$; and (4) for every $X \in$

Var. $\{A \in avars(\Delta) \mid X \in \alpha(A)\} \in \Delta$. A mapping α : AVar $\rightarrow$ Term is extended into a mapping α' : MTerm $\rightarrow$ Term as follows:

$$\alpha'(M) = \begin{cases} M & \text{if } M \in Var \\ \alpha(M) & \text{if } M \in \text{AVar} \\ f(\alpha'(M_1), \ldots, \alpha'(M_n)) & \text{if } M = f(M_1, \ldots, M_n). \end{cases}$$

Further extending α to apply to abstract equation systems is similar.

Definition 3 (description) *An abstract equation system* AS = $\langle$AEqs, $\Delta\rangle$ *describes a concrete system* Eqs *via* α, *denoted* AS $\propto_\alpha$ Eqs, *if there is a* Δ *preserving mapping* α : *avars*(AS) $\rightarrow$ Term *such that* α(AEqs) = Eqs *and for every* A $\in$ AVar, *vars*(α(A)) $\cap$ *vars*(AEqs) = $\emptyset$. *We write* AS $\propto$ Eqs *to denote that there exists a mapping* α *such that* AS $\propto_\alpha$ Eqs.

Let AS = $\langle$AEqs, $\Delta\rangle$ be an abstract equation system with associated annotation mapping ann_Δ and let A be an abstract variable occurring in Δ. We annotate the occurrences of A in Δ to indicate the value of $ann_\Delta(A)$: A^f, A^ℓ and A denote respectively that $ann_\Delta(A)$ is $\{f, \ell\}$, $\{\ell\}$ and $\emptyset$. We denote $\Delta(A) = \{S \in \Delta \mid A \in S\}$ and $\bar{\Delta}(A) = \{S \in \Delta \mid A \notin S\}$. Likewise $\bar{\Delta}(A_1, A_2) = \{S \in \Delta \mid A_1 \notin S,\ A_2 \notin S\}$.

Example 4 $\langle\{X = A_1, Y = A_2, U = f(V)\}, \{\{A_1^f, A_2\}\}\rangle$ describes $\{X = Z_1, Y = Z_2, U = f(V)\}$ as well as $\{X = Z_1, Y = f(g(Z_1, Z_1)), U = f(V)\}$ but not $\{X = V, Y = f(V, V), U = f(V)\}$. Denoting $\Delta = \{\{A_1^f, A_2\}\}$, $ann_\Delta(A_1) = \{f, \ell\}$ and $ann_\Delta(A_2) = \emptyset$. In addition, $\Delta(A_1) = \Delta(A_2) = \Delta$ and $\bar{\Delta}(A_1) = \bar{\Delta}(A_2) = \bar{\Delta}(A_1, A_2) = \emptyset$.

3 Abstract unification

A call pattern is a pair $(p(\bar{t}); \text{AS})$ where $p(\bar{t})$ is an atom and AS = $\langle$AEqs, $\Delta\rangle$ is an abstract equation system such that AEqs consists of an equation of the form $X = A$ for each variable X in $p(\bar{t})$. Using abstract equation systems is well suited to describe definite freeness and possible sharing information. A variable X is definitely free if there is an equation $X = A$ and $free_\Delta(A)$. Variables X and Y possibly share if $X = Y$ or if $X = A_1$ and $Y = A_2$ and $share_\Delta(A_1, A_2)$. Given a call pattern $(p(\bar{t}); \langle\text{AEqs}, \Delta\rangle)$ and a renamed clause of the form $p(\bar{s}) \leftarrow$ *body* we wish to reduce the abstract equation system $\langle \bar{t} = \bar{s} :: \text{AEqs}, \Delta\rangle$ to a *solved form* which describes the result of a corresponding concrete unification. More generally:

Definition 5 (problem specification) *Given an abstract equation system* AS, *derive an abstract equation system* AS′ *in solved form such that* AS $\propto$ Eqs $\Rightarrow$ AS′ $\propto$ Eqs′ *and* Eqs′ $\approx_{vars(\text{AS})}$ *mgu*(Eqs).

An algorithm which satisfies the above problem specification is called an *abstract unification algorithm*. We present an abstract unification algorithm consisting of two parts. The first part is a transition system which is applied to rewrite a given abstract system into a *pseudo solved form* of the form $\langle\{X_i = A_i\}_{i=1}^m :: \text{Eqs}, \Delta\rangle$ where $\{X_1, \ldots, X_m\}$ and $\{A_1, \ldots, A_m\}$ are sets of variables and abstract variables respectively, Eqs is a set of concrete equations in solved form, and the left side of an equation in Eqs is not equal to a variable X_i $(1 \leq i \leq m)$. The difference with a standard solved form is that the X_i can occur in the right side of an equation in Eqs. The second part of the algorithm derives from the pseudo solved form a solved form which expresses definite freeness, linearity and groundness information together with possible sharing information.

In the following we introduce a set of rewrite rules which are applied to reduce an abstract equation system to a solved form. Correctness follows because each transition AS $\rightarrow$ AS$'$ is shown to mimic a corresponding equivalence preserving transformation on concrete equation systems. Namely, if AS $\propto$ Eqs and Eqs is satisfiable, then there is a corresponding sequence of concrete transformations Eqs $\rightarrow^*$ Eqs$'$ such that Eqs $\approx_{vars(\mathsf{AS})}$ Eqs$'$ and AS$'$ $\propto$ Eqs$'$. We illustrate the intuition of each rule AS $\rightarrow$ AS$'$ by examples of the form

$$\begin{tikzcd}\mathsf{AS} \arrow[d] \arrow[r, phantom, "\propto"] & \mathsf{Eqs} \arrow[d, phantom, "\wr\wr"] \\ \mathsf{AS}' \arrow[r, phantom, "\propto"] & \mathsf{Eqs}'\end{tikzcd}$$

indicating that AS $\propto$ Eqs and that there is a transformation (sequence) of the form Eqs $\rightarrow^*$ Eqs$'$ such that AS$'$ $\propto$ Eqs$'$ and Eqs $\approx_{vars(\mathsf{AS})}$ Eqs$'$. As a convention, the equations chosen for reduction are underlined. Figure 1 illustrates the concrete transformations needed to justify our abstract algorithm. A transformation from Eqs to Eqs$'$ is denoted Eqs $\rightarrow_V$ Eqs$'$ where the V indicates that the transformation preserves equivalence with respect to the variables in $V \subseteq Var$. Clearly, if Eqs $\rightarrow_V$ Eqs$'$ then also Eqs $\rightarrow_{V'}$ Eqs$'$ for any $V' \subseteq V$. When $V = vars(\mathsf{Eqs})$ the subscript V is omitted. Rules 1–4 consist of the rules of the Martelli – Montanari algorithm. Rule 5 takes an equation, *solves* it and applies the result to the rest of the equations. Rule 6 *shuffles* the terms in two equations with a common right side. Rule 7 removes an equation of the form $X = \mathrm{T}$ if X does not occur in Eqs. This rule preserves equivalence with respect to the variables in Eqs. Rule 8 gives a fresh name Z to a variable X in an equation e and introduces a new equation $X = Z$.

Reducing equations of the form $\mathrm{T}_1 = \mathrm{T}_2$

These rules are basically the same as for concrete unification [8].

1. $\langle X = X :: \mathsf{AEqs}, \Delta\rangle \overset{\text{remove}}{\longrightarrow} \langle \mathsf{AEqs}, \Delta\rangle$.
2. $\langle X = \mathrm{T} :: \mathsf{AEqs}, \Delta\rangle \overset{\text{substitute}}{\longrightarrow} \langle X = \mathrm{T} :: \mathsf{AEqs}[X/\mathrm{T}], \Delta\rangle$ if $X \notin vars(\mathrm{T})$.
3. $\langle f(\mathrm{T}_1, \ldots, \mathrm{T}_n) = X :: \mathsf{AEqs}, \Delta\rangle \overset{\text{switch}}{\longrightarrow} \langle X = f(\mathrm{T}_1, \ldots, \mathrm{T}_n) :: \mathsf{AEqs}, \Delta\rangle$.
4. $\langle f(\mathrm{T}_1, \ldots, \mathrm{T}_n) = f(\mathrm{S}_1, \ldots, \mathrm{S}_n) :: \mathsf{AEqs}, \Delta\rangle \overset{\text{peel}}{\longrightarrow} \langle \{\mathrm{T}_i = \mathrm{S}_i\}_{i=1}^{n} :: \mathsf{AEqs}, \Delta\rangle$.

Reducing equations of the form $\mathrm{T} = \mathrm{A}$ such that $nonvar(\mathrm{T})$

Let T be a non-variable term such that $vars(\mathrm{T}) = \{X_1, \ldots, X_n\}$ (if $n = 0$ then T is ground). The following set of transitions are applied under the restriction that A *does not occur elsewhere in the abstract system.* If A does occur elsewhere, then a transition which better exploits the structural information in T is applied (see below). We distinguish the following cases:

1. $X = X :: \mathsf{Eqs} \overset{\text{remove}}{\rightarrow} \mathsf{Eqs}$.
2. $f(T_1,\ldots,T_n) = X :: \mathsf{Eqs} \overset{\text{switch}}{\rightarrow} X = f(T_1,\ldots,T_n) :: \mathsf{Eqs}$.
3. $f(T_1,\ldots,T_n) = g(S_1,\ldots,S_m) :: \mathsf{Eqs} \overset{\text{peel}}{\rightarrow} \begin{cases} \{T_i = S_i\}_{i=1}^{n} :: \mathsf{Eqs} & \text{if } f/n \equiv g/m \\ \mathit{fail} & \text{otherwise.} \end{cases}$
4. $X = T :: \mathsf{Eqs} \overset{\text{subst}}{\rightarrow} \begin{cases} X = T :: \mathsf{Eqs}[X/T] & \text{if } X \notin \mathit{vars}(T) \\ \mathit{fail} & \text{otherwise.} \end{cases}$
5. $e :: \mathsf{Eqs} \overset{\text{solve}}{\rightarrow} \mathit{eq}(\theta) :: \mathsf{Eqs}\,\theta$ where $\theta = \mathit{mgu}(e)$.
6. $T_1 = T, T_2 = T :: \mathsf{Eqs} \overset{\text{shuffle}}{\rightarrow} T_1 = T_2, T_1 = T :: \mathsf{Eqs}$.
7. $X = T :: \mathsf{Eqs} \overset{\text{restrict}}{\rightarrow}_V \mathsf{Eqs}$ where $X \notin \mathit{vars}(\mathsf{Eqs})$ and $V \subseteq \mathit{vars}(\mathsf{Eqs})$.
8. $e :: \mathsf{Eqs} \overset{\text{fresh}}{\rightarrow}_V e[X/Z], X = Z :: \mathsf{Eqs}$ where $X \in \mathit{vars}(e)$, Z is a fresh variable, and $V \subseteq \mathit{vars}(e :: \mathsf{Eqs})$.

Fig. 1. Equivalence preserving transformations for equation systems

5. $\langle T = A :: \mathsf{AEqs}, \Delta\rangle \overset{\text{fresh,solve\&restrict}}{\rightarrow} \langle X_1 = A_1, \ldots, X_n = A_n :: \mathsf{AEqs}, \Delta'\rangle$ where $A_1, \ldots, A_n$ are fresh abstract variables, and
   (a) If $\mathit{ground}_\Delta(A)$ then $\Delta' = \Delta$ (note that this implies that $\mathit{ground}_{\Delta'}(A_i)$ $(1 \leq i \leq n)$) and for $A' \in \mathit{avars}(\Delta')$, $\mathit{ann}_{\Delta'}(A') = \mathit{ann}_\Delta(A')$;
   (b) Else if $\mathit{free}_\Delta(A)$ then

```
* $\Delta' = \bar{\Delta}(A) \cup \{\, (S \setminus \{A\}) \cup \{A_i\} \mid S \in \Delta(A),\ 1 \leq i \leq n \,\}$;
* $\mathit{ann}_{\Delta'}(A') =$ if $A' \in \{A_1,\ldots,A_n\}$ then $\{f, \ell\}$
                  else  if $\mathit{share}_\Delta(A, A')$ then
                           if $\mathit{linear}(T)$ then $\mathit{ann}_\Delta(A') \setminus \{f\}$
                           else $\mathit{ann}_\Delta(A') \setminus \{f, \ell\}$
                        else $\mathit{ann}_\Delta(A')$
```

   (c) Else $(\mathit{ann}_\Delta(A) \subseteq \{\ell\})$

```
* $\Delta' =$ if $\mathit{linear}(T)$ then
         $\bar{\Delta}(A) \ \cup\ \{\, (S \setminus \{A\}) \cup S' \mid S \in \Delta(A),\ S' \in SS' \,\}$
     else
         $\bar{\Delta}(A) \ \cup\ \left\{ (\cup SS \setminus \{A\}) \cup S' \;\middle|\; \begin{array}{l} SS \subseteq \Delta(A), SS \neq \emptyset, \\ S' \in SS' \end{array} \right\}$
  where $SS' =$ if $\mathit{linear}(A)$ then   $\{\, \{A_i\} \mid 1 \leq i \leq n \,\}$.
               else  $\{\, S \subseteq \{A_1,\ldots,A_n\} \mid S \neq \emptyset \,\}$
* $\mathit{ann}_{\Delta'}(A') =$ if $A' \in \{A_1,\ldots,A_n\}$ then
                    if $\mathit{linear}(A)$ then $\{\ell\}$else $\emptyset$
                  else
                    if $\mathit{share}_\Delta(A, A')$ then
                       if $\mathit{linear}(T)$ then $\mathit{ann}_\Delta(A') \setminus \{f\}$
                       else $\mathit{ann}_\Delta(A') \setminus \{\ell, f\}$
                    else $\mathit{ann}_\Delta(A')$
```

Example 6

1. **T is ground**

$$\begin{array}{ccc}
\left\langle \begin{array}{l} \left\{ \underline{a = \mathbf{A}_1}, X = \mathbf{A}_2, Y = \mathbf{A}_3 \right\}, \\ \left\{ \{\mathbf{A}_1, \mathbf{A}_2, \mathbf{A}_3\}, \{\mathbf{A}_3^f\} \right\} \end{array} \right\rangle & \propto & \left\{ a = Z_1, X = f(Z_1), Y = Z_1 \right\} \\
\downarrow & & \wr \\
\left\langle \left\{ X = \mathbf{A}_2, Y = \mathbf{A}_3 \right\}, \left\{ \{\mathbf{A}_3^\ell\} \right\} \right\rangle & \propto & \left\{ X = f(a), Y = a \right\}
\end{array}$$

2. **A is free, T is non-linear**

$$\begin{array}{ccc}
\left\langle \begin{array}{l} \left\{ \underline{f(X,X) = \mathbf{A}_1}, Y = \mathbf{A}_2, U = \mathbf{A}_3 \right\}, \\ \left\{ \{\mathbf{A}_1^f, \mathbf{A}_2^\ell, \mathbf{A}_3^f,\} \right\} \end{array} \right\rangle & \propto & \left\{ f(X,X) = Z, Y = g(Z), U = Z) \right\} \\
\downarrow & & \wr \\
\left\langle \begin{array}{l} \left\{ X = \mathbf{A}_{1,1}, Y = \mathbf{A}_2, U = \mathbf{A}_3 \right\}, \\ \left\{ \{\mathbf{A}_{1,1}^f, \mathbf{A}_2, \mathbf{A}_3\} \right\} \end{array} \right\rangle & \propto & \left\{ \begin{array}{l} X = Z_1, Y = g(f(Z_1, Z_1)), \\ U = f(Z_1, Z_1) \end{array} \right\}
\end{array}$$

3. **A is linear, T is non-linear (observe that the new abstract variables are linear and both linearity and freeness are lost in all abstract variables which possibly share with $\mathbf{A}_1$)**

$$\begin{array}{ccc}
\left\langle \begin{array}{l} \left\{ \underline{f(g(X,X),X) = \mathbf{A}_1}, V = \mathbf{A}_3, U = \mathbf{A}_2 \right\}, \\ \left\{ \{\mathbf{A}_1^\ell, \mathbf{A}_2^f\}, \{\mathbf{A}_1^\ell, \mathbf{A}_3^f\} \right\} \end{array} \right\rangle & \propto & \left\{ \begin{array}{l} f(g(X,X),X) = f(Z_1, h(Z_2)), \\ V = Z_2, U = Z_1 \end{array} \right\} \\
\downarrow & & \wr \\
\left\langle \begin{array}{l} \left\{ X = \mathbf{A}_{1,1}, V = \mathbf{A}_3, U = \mathbf{A}_2 \right\}, \\ \left\{ \{\mathbf{A}_{1,1}^\ell, \mathbf{A}_2\}, \{\mathbf{A}_{1,1}^\ell, \mathbf{A}_3\}, \{\mathbf{A}_{1,1}^\ell, \mathbf{A}_2, \mathbf{A}_3\} \right\} \end{array} \right\rangle & \propto & \left\{ \begin{array}{l} X = h(Z_2), V = Z_2, \\ U = g(h(Z_2), h(Z_2)) \end{array} \right\}
\end{array}$$

4. **A is 'any', T is non-linear**

$$\begin{array}{ccc}
\left\langle \begin{array}{l} \left\{ \underline{f(X,X) = \mathbf{A}_1}, U = \mathbf{A}_2, V = \mathbf{A}_3 \right\}, \\ \left\{ \{\mathbf{A}_1, \mathbf{A}_2^f\}, \{\mathbf{A}_1, \mathbf{A}_3^f,\} \right\} \end{array} \right\rangle & \propto & \left\{ \begin{array}{l} f(X,X) = f(Z_1, h(Z_2, Z_2)), \\ U = Z_1, V = Z_2) \end{array} \right\} \\
\downarrow & & \wr \\
\left\langle \begin{array}{l} \left\{ X = \mathbf{A}_{1,1}, U = \mathbf{A}_2, V = \mathbf{A}_3 \right\}, \\ \left\{ \{\mathbf{A}_{1,1}, \mathbf{A}_2\}, \{\mathbf{A}_{1,1}, \mathbf{A}_3\}, \{\mathbf{A}_{1,1}, \mathbf{A}_2, \mathbf{A}_3\} \right\} \end{array} \right\rangle & \propto & \left\{ \begin{array}{l} X = h(Z_2, Z_2), \\ U = h(Z_2, Z_2), V = Z_2 \end{array} \right\}
\end{array}$$

5. **A is 'any', T is linear**

$$\begin{array}{ccc}
\left\langle \begin{array}{l} \left\{ \underline{f(X,Y) = \mathbf{A}_1}, U = \mathbf{A}_2 \right\}, \\ \left\{ \{\mathbf{A}_1, \mathbf{A}_2^f\} \right\} \end{array} \right\rangle & \propto & \left\{ \begin{array}{l} f(X,Y) = f(g(Z), g(Z)), \\ U = Z \end{array} \right\} \\
\downarrow & & \wr \\
\left\langle \begin{array}{l} \left\{ X = \mathbf{A}_{1,1}, Y = \mathbf{A}_{1,2}, U = \mathbf{A}_2 \right\}, \\ \left\{ \{\mathbf{A}_{1,1}, \mathbf{A}_2^\ell\}, \{\mathbf{A}_{1,2}, \mathbf{A}_2^\ell\}, \{\mathbf{A}_{1,1}, \mathbf{A}_{1,2}, \mathbf{A}_2^\ell\} \right\} \end{array} \right\rangle & \propto & \left\{ X = g(Z), Y = g(Z), U = Z \right\}
\end{array}$$

Reducing equations of the form $X = \mathbf{A}$

Equations of the form $X = \mathbf{A}$ are maintained in the pseudo solved form as long as X does not appear elsewhere as a left hand side. The following rule is applied to reduce the number of occurrences of a variable X occurring in the left side of an equation. The abstract variables participating in the reduction must have single occurrences in the equations of the system.

6. $\langle X = \mathbf{A}_1, X = \mathbf{A}_2 :: \mathsf{AEqs}, \Delta\rangle \overset{\textbf{shuffle,solve\&restrict}}{\longrightarrow} \langle X = \mathbf{A}_1 :: \mathsf{AEqs}, \Delta'\rangle$ where

   - $\Delta' =$ if $ground_\Delta(\mathbf{A}_1) \vee ground_\Delta(\mathbf{A}_2)$ then $\bar{\Delta}(\mathbf{A}_1, \mathbf{A}_2)$
     else
     $$\bar{\Delta}(\mathbf{A}_1,\mathbf{A}_2) \cup \left\{ (S_{\mathbf{A}_1} \cup S_{\mathbf{A}_2}) \setminus \{\mathbf{A}_2\} \;\middle|\; \begin{array}{l} i,j \in \{1,2\},\ i \neq j, \\ \text{if } \ linear_\Delta(\mathbf{A}_i) \wedge \\ \qquad\qquad \neg share_\Delta(\mathbf{A}_i, \mathbf{A}_j) \\ \text{then } \ S_{\mathbf{A}_j} \in \Delta(\mathbf{A}_j) \\ \text{else } \ S_{\mathbf{A}_j} = \cup\, SS \text{ where} \\ \qquad SS \subseteq \Delta(\mathbf{A}_j), SS \neq \emptyset \end{array} \right\}$$
   - for $\mathbf{A}' \in avars(\Delta')$:

```
$ann_{\Delta'}(\mathbf{A}') =$
  if $\mathbf{A}' = \mathbf{A}_1$ then
    if $free_\Delta(\mathbf{A}_1) \wedge free_\Delta(\mathbf{A}_2)$ then $\{f, \ell\}$
    else if $linear_\Delta(\mathbf{A}_1) \wedge linear_\Delta(\mathbf{A}_2) \wedge \neg share_\Delta(\mathbf{A}_1, \mathbf{A}_2)$ then $\{\ell\}$
    else $\emptyset$
  else if
    $(\neg share_\Delta(\mathbf{A}_1, \mathbf{A}') \wedge \neg share_\Delta(\mathbf{A}_2, \mathbf{A}'))\vee$
    $(free_\Delta(\mathbf{A}_1) \wedge free_\Delta(\mathbf{A}_2) \wedge$
        $(free_\Delta(\mathbf{A}') \vee \neg share_\Delta(\mathbf{A}_1, \mathbf{A}') \vee \neg share_\Delta(\mathbf{A}_2, \mathbf{A}')))\vee$
    $(free_\Delta(\mathbf{A}_2) \wedge \neg share_\Delta(\mathbf{A}_2, \mathbf{A}')) \vee (free_\Delta(\mathbf{A}_1) \wedge \neg share_\Delta(\mathbf{A}_1, \mathbf{A}'))$
      then $ann_\Delta(\mathbf{A}')$
  else if
    $(linear_\Delta(\mathbf{A}_2) \wedge \neg share_\Delta(\mathbf{A}_2, \mathbf{A}')) \vee (linear_\Delta(\mathbf{A}_1) \wedge \neg share_\Delta(\mathbf{A}_1, \mathbf{A}'))\vee$
    $(free_\Delta(\mathbf{A}') \wedge (free_\Delta(\mathbf{A}_2) \wedge linear_\Delta(\mathbf{A}_1))\vee$
      $(free_\Delta(\mathbf{A}_1) \wedge linear_\Delta(\mathbf{A}_2)))$ then $ann_\Delta(\mathbf{A}') \setminus \{f\}$
  else $\emptyset$
```

Example 7

$$\begin{array}{ccc}
\left\langle \left\{ \begin{array}{l} \underline{X = \mathbf{A}_1}, \underline{X = \mathbf{A}_2}, \\ Y_1 = \mathbf{A}_3, Y_2 = \mathbf{A}_4, \\ U_1 = \mathbf{A}_5, U_2 = \mathbf{A}_6 \end{array} \right\}, \left\{ \begin{array}{l} \{\mathbf{A}_1, \mathbf{A}_3^f\}, \{\mathbf{A}_1, \mathbf{A}_4^\ell\}, \\ \{\mathbf{A}_2^\ell, \mathbf{A}_5^f\}, \{\mathbf{A}_2^\ell, \mathbf{A}_6^\ell\} \end{array} \right\} \right\rangle & \propto & \left\{ \begin{array}{l} X = f(g(Z_1, Z_1), Z_2), \\ X = f(Z_3, h(Z_4)), \\ Y_1 = Z_2, Y_2 = h(Z_1), \\ U_1 = Z_3, U_2 = h(Z_4) \end{array} \right\} \\
\downarrow & & \wr\wr \\
\left\langle \left\{ \begin{array}{l} X = \mathbf{A}_1, \\ Y_1 = \mathbf{A}_3, Y_2 = \mathbf{A}_4, \\ U_1 = \mathbf{A}_5, U_2 = \mathbf{A}_6 \end{array} \right\}, \left\{ \begin{array}{l} \{\mathbf{A}_1, \mathbf{A}_3^\ell, \mathbf{A}_5\}, \\ \{\mathbf{A}_1, \mathbf{A}_3^\ell, \mathbf{A}_6\}, \\ \{\mathbf{A}_1, \mathbf{A}_3^\ell, \mathbf{A}_5, \mathbf{A}_6\}, \\ \{\mathbf{A}_1, \mathbf{A}_4^\ell, \mathbf{A}_5\}, \\ \{\mathbf{A}_1, \mathbf{A}_4^\ell, \mathbf{A}_6\}, \\ \{\mathbf{A}_1, \mathbf{A}_4^\ell, \mathbf{A}_5, \mathbf{A}_6\} \end{array} \right\} \right\rangle & \propto & \left\{ \begin{array}{l} X = f(g(Z_1, Z_1), h(Z_4)), \\ Y_1 = h(Z_4), Y_2 = h(Z_1), \\ U_1 = g(Z_1, Z_1), U_2 = h(Z_4) \end{array} \right\}
\end{array}$$

While linearity is preserved for $\mathbf{A}_3$ and $\mathbf{A}_4$, it is lost for $\mathbf{A}_5$ and $\mathbf{A}_6$. Taking $f(g(Z_1, Z_2), Z_1)$ for $\mathbf{A}_1$ in the original concrete equation illustrates the need for $\{\mathbf{A}_1, \mathbf{A}_4^\ell, \mathbf{A}_5, \mathbf{A}_6\}$ in the transformed sharing component.

Reducing equations of the form T = A (multiple occurrences of A)

The previous set of rules did not apply to the case in which the abstract variable occurring in an abstract equation system has multiple occurrences. The following rule reduces the number of occurrences of an abstract variable:

7. $\langle \mathbf{T}_1 = \mathbf{A}, \mathbf{T}_2 = \mathbf{A} :: \mathsf{AEqs}, \Delta \rangle \overset{\text{shuffle}}{\rightarrow} \langle \mathbf{T}_1 = \mathbf{T}_2, \mathbf{T}_1 = \mathbf{A} :: \mathsf{AEqs}, \Delta \rangle$.

Example 8

$$\begin{array}{ccc}
\left\langle \left\{ \begin{array}{l} \underline{f(a,X) = \mathbf{A}_1}, U = \mathbf{A}_2, \\ \underline{f(Y,b) = \mathbf{A}_1}, V = \mathbf{A}_3 \end{array} \right\}, \left\{ \begin{array}{l} \{\mathbf{A}_1, \mathbf{A}_2^f\}, \\ \{\mathbf{A}_1, \mathbf{A}_3^f\} \end{array} \right\} \right\rangle & \propto & \left\{ \begin{array}{l} f(a,X) = f(Z_1,Z_2), U = Z_1, \\ f(Y,b) = f(Z_1,Z_2), V = Z_2 \end{array} \right\} \\
\downarrow & & \wr \\
\left\langle \left\{ \begin{array}{l} f(a,X) = f(Y,b), U = \mathbf{A}_2, \\ f(a,X) = \mathbf{A}_1, V = \mathbf{A}_3 \end{array} \right\}, \left\{ \begin{array}{l} \{\mathbf{A}_1, \mathbf{A}_2^f\}, \\ \{\mathbf{A}_1, \mathbf{A}_3^f\} \end{array} \right\} \right\rangle & \propto & \left\{ \begin{array}{l} f(a,X) = f(Y,b), U = Z_1, \\ f(a,X) = f(Z_1,Z_2), V = Z_2 \end{array} \right\}
\end{array}$$

Psuedo solved form

Repeated application of rules 1 – 7 provides an abstract equation system of the form $\langle \{X_i = \mathbf{A}_i\}_{i=1}^m :: \mathsf{Eqs}, \Delta \rangle$ where $\{X_1, \ldots, X_m\}$ and $\{\mathbf{A}_1, \ldots, \mathbf{A}_m\}$ are sets of variables and abstract variables respectively, Eqs is a set of <u>concrete</u> equations in solved form, and the left side of an equation in Eqs is not equal to a variable X_i $(1 \leq i \leq m)$. An abstract equation system of this form is said to be in *pseudo solved form.*

Reducing the pseudo solved form

The following rule is applied to reduce an abstract system in pseudo solved form to an abstract system in *solved form*. Namely, an abstract system which describes only concrete systems which are in solved form.

8. $\langle \{X_i = \mathbf{A}_i\}_{i=1}^m :: \mathsf{Eqs}, \Delta \rangle \overset{\text{abstract-subst}}{\rightarrow} \langle \{X_i = \mathbf{A}_i\}_{i=1}^m :: \mathsf{Eqs}\,\sigma, \Delta \rangle$
   where $\sigma = \{X_1/\mathbf{A}_1, \ldots, X_m/\mathbf{A}_m\}$

Strictly speaking, the result of this transition is not an abstract equation system as it may potentially involve equations with mixed terms. However, the definition of description is applicable. Moreover the following rules remove mixed terms.

Example 9

$$\begin{array}{ccc}
\langle \{ \underline{X = \mathbf{A}_1}, Y = g(X,U,X) \}, \{ \{\mathbf{A}_1^f\} \} \rangle & \propto & \{ X = f(Z_1), Y = g(X,U,X) \} \\
\downarrow & & \wr \\
\langle \{ X = \mathbf{A}_1, Y = g(\mathbf{A}_1,U,\mathbf{A}_1) \}, \{ \{\mathbf{A}_1^f\} \} \rangle & \propto & \{ X = f(Z_1), Y = g(f(Z_1),U,f(Z_1)) \}
\end{array}$$

Abstracting the solved form

Recall that a mixed term is a term which involves variables as well as abstract variables. The following transition is applied to remove variables occurring in mixed terms.

9. $\langle \mathsf{AEqs}, \Delta \rangle \overset{\text{remove-var}}{\rightarrow} \langle X = \mathbf{A} :: \mathsf{AEqs}[X/\mathbf{A}], \Delta \cup \{\{\mathbf{A}^f\}\} \rangle$
   if X occurs in a mixed term (which is not a term) in some equation of AEqs and $\mathbf{A}$ is a fresh abstract variable.

Repeated application of Rule 9 replaces all variables in mixed terms by abstract variables hence transforming mixed terms to abstract terms. From here on, we assume without loss of generality that mixed terms do not occur in abstract equation systems.

Example 10

$$\begin{array}{ccc}
\left\langle \begin{Bmatrix} X = \mathbf{A}_1, \underline{Y = g(\mathbf{A}_1, U, \mathbf{A}_1)}, \\ \{\mathbf{A}_1^{\ell}\} \end{Bmatrix} \right\rangle & \propto & \begin{Bmatrix} X = f(Z_1), \\ Y = g(f(Z_1), U, f(Z_1)) \end{Bmatrix} \\
\downarrow & & \| \\
\left\langle \begin{Bmatrix} U = \mathbf{A}_2, X = \mathbf{A}_1, \underline{Y = g(\mathbf{A}_1, \mathbf{A}_2, \mathbf{A}_1)}, \\ \{\mathbf{A}_1^{\ell}\}, \{\mathbf{A}_2^{f}\} \end{Bmatrix} \right\rangle & \propto & \begin{Bmatrix} U = Z_2, X = f(Z_1), \\ Y = g(f(Z_1), Z_2, f(Z_1)) \end{Bmatrix}
\end{array}$$

Let $\mathcal{T}$ be an abstract term occurring in an abstract equation system $\langle \mathsf{AEqs}, \Delta \rangle$. We denote: $\mathit{free}_{\Delta}(\mathcal{T})$, if $\mathcal{T} = \mathbf{A}$ and $\mathit{free}_{\Delta}(\mathbf{A})$; and $\mathit{linear}_{\Delta}(\mathcal{T})$, if $\forall \mathbf{A}, \mathbf{A}' \in \mathit{avars}(\mathcal{T})$, (i) $\mathit{linear}_{\Delta}(\mathbf{A})$, (ii) if $\mathbf{A}$ occurs more than once in $\mathcal{T}$ then $\mathit{ground}_{\Delta}(\mathbf{A})$, and (iii) $\neg \mathit{share}_{\Delta}(\mathbf{A}, \mathbf{A}')$. An abstract system involving abstract terms is abstracted using the following rule

10. $\langle \mathsf{AEqs}, \Delta \rangle \overset{\text{abstract}}{\longrightarrow} \langle \mathsf{AEqs}', \Delta' \rangle$
    where $\{\mathcal{T}_1, \ldots, \mathcal{T}_n\}$ is the <u>set</u> of non variable abstract terms occurring (on the right sides of the equations) in AEqs $\mathbf{A}_1, \ldots, \mathbf{A}_n$ are fresh abstract variables and
    - AEqs$'$ is obtained by replacing each $\mathcal{T}_i$ in AEqs by the corresponding $\mathbf{A}_i$;
    - $\Delta' = \{\, S \cup \{\, \mathbf{A}_i \mid \mathit{avars}(\mathcal{T}_i) \cap S \neq \emptyset,\ 1 \leq i \leq n \,\} \mid S \in \Delta \,\}$; and
    - $\mathit{ann}_{\Delta'}(\mathbf{A}) =$ **if** $\mathbf{A} \in \{\mathbf{A}_1, \ldots, \mathbf{A}_n\}$ **then**

```
                 if $free_{\Delta}(\mathcal{T}_i)$ then $\{f, \ell\}$
                 else if $linear_{\Delta}(\mathcal{T}_i)$ then $\{\ell\}$
                 else $\emptyset$
             else $ann_{\Delta}(\mathbf{A})$
```

Example 11

$$\begin{array}{ccc}
\left\langle \begin{Bmatrix} U = \mathbf{A}_2, X = \mathbf{A}_1, \underline{Y = g(\mathbf{A}_1, \mathbf{A}_2, \mathbf{A}_1)}, \\ \{\mathbf{A}_1^{\ell}\}, \{\mathbf{A}_2^{f}\} \end{Bmatrix} \right\rangle & \propto & \begin{Bmatrix} U = Z_2, X = f(Z_1), \\ Y = g(f(Z_1), Z_2, f(Z_1)) \end{Bmatrix} \\
\downarrow & & \| \\
\left\langle \begin{Bmatrix} U = \mathbf{A}_2, X = \mathbf{A}_1, Y = \mathbf{A}_3, \\ \{\mathbf{A}_1^{\ell}, \mathbf{A}_3\}, \{\mathbf{A}_2^{f}, \mathbf{A}_3\} \end{Bmatrix} \right\rangle & \propto & \begin{Bmatrix} U = Z_2, X = f(Z_1), \\ Y = g(f(Z_1), Z_2, f(Z_1)) \end{Bmatrix}
\end{array}$$

4 Examples

The following examples are adapted from [9].

Example 12

$$\left\langle \begin{array}{l} \left\{ \begin{array}{l} X_1 = \mathbf{A}_1, X_2 = \mathbf{A}_2, X_3 = \mathbf{A}_3, X_4 = \mathbf{A}_4, X_5 = \mathbf{A}_5, X_6 = \mathbf{A}_6, \underline{f(X_1, X_2) = Y_1}, \\ \underline{f(X_3, X_4) = Y_1}, \underline{X_3 = a}, \underline{X_5 = Y_2}, \underline{X_6 = Y_2}, \underline{X_6 = f(Y_1, Y_3)} \end{array} \right\}, \\ \{\{\mathbf{A}_1^{f}, \mathbf{A}_2\}, \{\mathbf{A}_2\}, \{\mathbf{A}_3^{f}\}, \{\mathbf{A}_5^{f}\}, \{\mathbf{A}_6^{f}\}\} \end{array} \right\rangle \rightarrow^{*}$$

[rules 1–4; the sharing component is not effected.]

$$\left\langle \begin{array}{l} \left\{ \begin{array}{l} X_1 = a, X_3 = a, X_4 = X_2, X_5 = f(f(a, X_2), Y_3), X_6 = f(f(a, X_2), Y_3), \\ Y_1 = f(a, X_2), Y_2 = f(f(a, X_2), Y_3), \underline{a = \mathbf{A}_1}, \underline{a = \mathbf{A}_3}, X_2 = \mathbf{A}_4, X_2 = \mathbf{A}_2, \\ f(f(a, X_2), Y_3) = \mathbf{A}_5, f(f(a, X_2), Y_3) = \mathbf{A}_6 \end{array} \right\}, \\ \{\{\mathbf{A}_1^{f}, \mathbf{A}_2\}, \{\mathbf{A}_2\}, \{\mathbf{A}_3^{f}\}, \{\mathbf{A}_5^{f}\}, \{\mathbf{A}_6^{f}\}\} \end{array} \right\rangle \rightarrow^{*}$$

[rule 5; $\mathbf{A}_1$ and $\mathbf{A}_3$ become ground.]

$$\left\langle \begin{array}{l} \left\{ \begin{array}{l} X_1 = a, X_3 = a, X_4 = X_2, X_5 = f(f(a,X_2),Y_3), X_6 = f(f(a,X_2),Y_3), \\ Y_1 = f(a,X_2), Y_2 = f(f(a,X_2),Y_3), \underline{X_2 = \mathbf{A}_2}, \underline{X_2 = \mathbf{A}_4}, \\ f(f(a,X_2),Y_3) = \mathbf{A}_5, f(f(a,X_2),Y_3) = \mathbf{A}_6 \end{array} \right\}, \\ \{ \{\mathbf{A}_2\}, \{\mathbf{A}_5^f\}, \{\mathbf{A}_6^f\} \} \end{array} \right\rangle \rightarrow$$

[rule 6; $\mathbf{A}_2$ becomes ground.]

$$\left\langle \begin{array}{l} \left\{ \begin{array}{l} X_1 = a, X_3 = a, X_4 = X_2, X_5 = f(f(a,X_2),Y_3), X_6 = f(f(a,X_2),Y_3), \\ Y_1 = f(a,X_2), Y_2 = f(f(a,X_2),Y_3), \\ X_2 = \mathbf{A}_4, \underline{f(f(a,X_2),Y_3) = \mathbf{A}_5}, \underline{f(f(a,X_2),Y_3) = \mathbf{A}_6} \end{array} \right\}, \\ \{ \{\mathbf{A}_5^f\}, \{\mathbf{A}_6^f\} \} \end{array} \right\rangle \rightarrow^*$$

[rules 5 and 6; $\mathbf{A}_{5,1}$ and $\mathbf{A}_{6,1}$ remain free.]

$$\left\langle \begin{array}{l} \left\{ \begin{array}{l} X_1 = a, X_3 = a, X_4 = X_2, X_5 = f(f(a,X_2),Y_3), X_6 = f(f(a,X_2),Y_3), \\ Y_1 = f(a,X_2), Y_2 = f(f(a,X_2),Y_3), \underline{X_2 = \mathbf{A}_4}, \underline{X_2 = \mathbf{A}_{5,1}}, Y_3 = \mathbf{A}_{5,2} \end{array} \right\}, \\ \{ \{\mathbf{A}_{5,1}^f\}, \{\mathbf{A}_{5,2}^f\} \} \end{array} \right\rangle \rightarrow$$

[rule 6; $\mathbf{A}_{5,1}$ becomes ground.]

$$\left\langle \begin{array}{l} \left\{ \begin{array}{l} X_1 = a, X_3 = a, X_4 = X_2, X_5 = f(f(a,X_2),Y_3), X_6 = f(f(a,X_2),Y_3), \\ Y_1 = f(a,X_2), Y_2 = f(f(a,X_2),Y_3), \underline{X_2 = \mathbf{A}_4}, \underline{Y_3 = \mathbf{A}_{5,2}} \end{array} \right\}, \\ \{ \{\mathbf{A}_{5,2}^f\} \} \end{array} \right\rangle \rightarrow^*$$

[pseudo solved form.] [rule 8.]

$$\left\langle \begin{array}{l} \left\{ \begin{array}{l} X_1 = a, X_3 = a, X_4 = \mathbf{A}_4, \underline{X_5 = f(f(a,\mathbf{A}_4),\mathbf{A}_{5,2})}, \underline{X_6 = f(f(a,\mathbf{A}_4,),\mathbf{A}_{5,2})}, \\ \underline{Y_1 = f(a,\mathbf{A}_4)}, \underline{Y_2 = f(f(a,\mathbf{A}_4),\mathbf{A}_{5,2})}, X_2 = \mathbf{A}_4, Y_3 = \mathbf{A}_{5,2} \end{array} \right\}, \\ \{ \{\mathbf{A}_{5,2}^f\} \} \end{array} \right\rangle \rightarrow$$

[solved form (mixed terms).] [rule 10.]

$$\langle \{ X_1 = a, X_3 = a, X_4 = \mathbf{A}_4, X_5 = \mathbf{A}_7, X_6 = \mathbf{A}_7, Y_1 = \mathbf{A}_8, Y_2 = \mathbf{A}_7, X_2 = \mathbf{A}_4, Y_3 = \mathbf{A}_{5,2} \}, \{ \{\mathbf{A}_{5,2}^f, \mathbf{A}_7^f\} \} \rangle$$

[solved form (no mixed terms).]

This shows that X_1 and X_3 are ground (and equal), that X_2 and X_4 are ground (and equal), that Y_1 is ground , that X_5, X_6 and Y_2 are not free (and equal), that Y_3 is free and that X_5, X_6, and Y_2 are linear terms which can share Y_3.

Example 13

$$\langle \left\{ \begin{array}{l} \underline{X_1 = \mathbf{A}_1}, X_2 = \mathbf{A}_2, X_3 = \mathbf{A}_3, \\ X_4 = \mathbf{A}_4, \underline{X_1 = \mathbf{A}_5}, X_2 = \mathbf{A}_6 \end{array} \right\}, \{ \{\mathbf{A}_1^f\}, \{\mathbf{A}_3^f\}, \{\mathbf{A}_2^f, \mathbf{A}_4^f\}, \{\mathbf{A}_5\}, \{\mathbf{A}_6\} \} \rangle \overset{6}{\rightarrow}$$

$$\langle \left\{ \begin{array}{l} X_1 = \mathbf{A}_1, \underline{X_2 = \mathbf{A}_2}, X_3 = \mathbf{A}_3, \\ X_4 = \mathbf{A}_4, \underline{X_2 = \mathbf{A}_6} \end{array} \right\}, \{ \{\mathbf{A}_1\}, \{\mathbf{A}_3^f\}, \{\mathbf{A}_2^f, \mathbf{A}_4^f\}, \{\mathbf{A}_2\}, \{\mathbf{A}_6\} \} \rangle \overset{6}{\rightarrow}$$

$$\langle \{ X_1 = \mathbf{A}_1, X_2 = \mathbf{A}_2, X_3 = \mathbf{A}_3, X_4 = \mathbf{A}_4 \}, \{ \{\mathbf{A}_1\}, \{\mathbf{A}_3^f\}, \{\mathbf{A}_2, \mathbf{A}_4\} \} \rangle$$

This correctly shows that only X_3 remains free and that X_2 and X_4 can be non-linear.

5 Conclusion

We have presented a powerful abstract unification algorithm for deriving sharing and freeness information together with groundness and linearity. The algorithm appears to be at least as precise as other previously proposed algorithms such as [1, 3, 4, 9, 11]. The algorithm is formalised as a transition system which reduces a set of abstract equations to an abstract solved form. In this approach each transition can be analysed and justified seperately. A formal proof of correctness, an implementation of the algorithm and an experimental evaluation of its precision are underway.

Acknowledgements We acknowledge the comments of M. García de la Banda, Anne Mulkers and the anonymous referees of a previous version of this paper. We thank Joost for typing the first draft.

References

1. M. Codish, D. Dams, G. File, M. Bruynooghe. Freeness Analysis for Logic Programs - And Correctness? *Proc. of Tenth Int. Conf. on Logic Programming*, Budapest, 1993.
2. M. Codish, D. Dams, and E. Yardeni. Derivation and safety of an abstract algorithm for groundness and aliasing analysis. In K. Furukawa, editor, *Proc. Eighth Int. Conf. on Logic Programming*, pages 79- 93. MIT Press, 1991.
3. M. Codish, A. Mulkers, M. Bruynooghe, M. García de la Banda and M. Hermenegildo. Improving abstract interpretations by combining domains. In *Proc. ACM Symposium on Partial Evaluation and Semantics Based Program Manipulation.* 1993.
4. A. Cortesi and G. Filé. Abstract interpretation of logic programs: an abstract domain for groundness, sharing, freeness and compoundness analysis. In P. Hudak and N. Jones, editors, *Proc. ACM Symposium on Partial Evaluation and Semantics Based Program Manipulation.* SIGPLAN NOTICES vol. 26, n.11, 1991.
5. P. Cousot and R. Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Proc. Fourth ACM symp. on Principles of Programming Languages*, pages 238–252, Los Angeles, California, 1977.
6. D. Jacobs and A. Langen. Static analysis of logic programs for independent and-parallelism. *Journal of Logic Programming*, 13(2 and 3):291–314, July 1992.
7. J.-L. Lassez, M.J. Maher, and K. Marriott. Unification revisited. In J. Minker, editor, *Foundations of Deductive Databases and Logic Programming.* Morgan Kauffmann, 1987.
8. A. Martelli and U. Montanari. An efficient unification algorithm. *ACM Transactions on Programming Languages and Systems*, 4(2):258–282, April 1982.
9. K. Muthukumar and M. Hermenegildo. Combined determination of sharing and freeness of program variables through abstract interpretation. In K. Furukawa, editor, *Proc. Eighth International Conference on Logic Programming*, pages 49–63. MIT Press, 1991.
10. H. Søndergaard. An application of abstract interpretation of logic programs: occur check reduction. In B. Robinet and R. Wilhelm, editors, *ESOP'86 Proc. European Symposium on Programming*, LNCS 213, pages 327–338. Springer-Verlag, 1986.
11. R. Sundararajan and J. Conery. An abstract interpretation scheme for groundness, freeness, and sharing analysis of logic programs. *Proc. Twelfth FST & TCS Conf.*, New Delhi, Dec. 1992.