A Composite Domain for Freeness, Sharing, and Compoundness Analysis of Logic Programs

Maurice Bruynooghe*
Dept. of Comp. Sci.
K.U.Leuven
Belgium
maurice@cs.kuleuven.ac.be

Michael Codish
Dept. of Math. & Comp. Sci.
Ben-Gurion Univ.
Israel
codish@bengus.bgu.ac.il

Anne Mulkers†
Dept. of Comp. Sci.
K.U.Leuven
Belgium
anne@cs.kuleuven.ac.be

July 1994

Abstract

Accurate sharing and freeness properties of program variables have been inferred in the past by means of combined dataflow analyses of logic programs. Groundness, linearity, and structural information must be taken into account in order to obtain sufficient precision. Abstract equation systems are one of the formalisms previously proposed to represent all these properties of the runtime values of program variables. The present work is concerned with the specification of correct and practical operations on the domain of abstract equation systems, which constitute a prerequisite for the domain to be suited for an implementation in the framework of abstract interpretation. The main technical contribution of the paper is the presentation of a novel and powerful algorithm for resolving an abstract equation system, the formal proof of its correctness, and a study of the invariance of certain linearity and freeness properties under the application of relevant most general unifiers.

Keywords: Logic programming, abstract interpretation, mode analysis.

1 Introduction

Accurate possible sharing and definite freeness and compoundness information is of primary importance in a variety of dataflow analyses of logic programs; for instance, for automatic (restricted) AND-parallelisation of programs and for compiler optimisations improving the efficiency of unification and clause indexing in sequential logic programs. Many domains for variable sharing and freeness information have been proposed that can be used in static analyses based on the framework of abstract interpretation (e.g., [11, 23, 24, 30, 32]). They are mostly based on the 'set-sharing' approach for variable sharing analysis as developed by Jacobs and Langen [20, 21]. Verifying these analyses and evaluating their accuracy and performance has been a hard task and the issue of what might be the most appropriate domain is still not solved. The aim of the present effort to design yet another enhancement of the basic set-sharing domain for deriving possible variable sharing is to obtain a formally verified analysis that is precise and apt for an efficient implementation

*Supported by the Belgium National Fund for Scientific Research. Address: Celestijnenlaan 200 A, 3001 Herverlee, Belgium, Tel. ++32(0)16-20 10 15, Fax. ++32(0)16-20 53 08

†Supported by DPWB contract IT/IF/4, Belgium.

by using a concise representation and effective abstract rewrite rules. It is generally agreed that combined analyses potentially yield more accurate results than separate independent flow analyses, as was demonstrated in, e.g., [9, 12, 30]. An effective combination that takes most advantage of the independent parts, in general requires redefining the basic operations [12, 14]. Our starting point is the composite abstract domain introduced in [5], which consists of abstract equation systems augmented with a sharing component and an annotation mapping. The abstract equation system provides depth-bounded structural information together with definite equality of subterms and has quite similar expressive power as the generic pattern domain of [26, 27, 33]. The sharing component is based on the set-sharing domain of Jacobs and Langen [20, 21]; it captures possible sharing as well as covering information and provides a good propagation of groundness information. However, the sharing information is expressed at the level of so-called *abstract* variables, which are convenient to represent *strict* subterms of the runtime values of program variables, and hence are fit to represent the sharing information to a higher degree of accuracy than when expressed at the level of program variables. Abstract variables are also annotated to capture additional freeness and linearity information, which have been put forward in recent literature to be quite useful ingredients for improving the precision of the basic set-sharing domain. In the present paper, the annotation mapping is further extended to capture compoundness information ('definitely compound' means 'definitely not free'; 'not definitely free' is the same as 'possibly compound'). Compoundness information proved useful for optimizing the sharing analysis and can be exploited by a compiler to improve clause indexing. Among the technical contributions of the present paper are a definition of the preorder relation and the abstract solved form for this composite domain, and the presentation of a revised unification algorithm able to handle equations involving mixed terms in a uniform way, and its correctness proof. The basic strategy, also applied in [6, 7], is to systematically mimic each step in the Martelli-Montanari unification algorithm [29]. Equations on mixed terms are rewritten by the algorithm to a set of equations which are either in solved form or involve abstract terms only; the latter equations are then solved in the set-sharing domain augmented with the annotation mapping. Other abstract domains providing variable sharing and mode information could be augmented with depth-bounded structural and term-equality information using an abstract equations component in a similar way. By presenting the abstract unification algorithm as a high-level equation solving algorithm, its correctness proof can be split into verifying the rewriting system and proving the correctness of the underlying domain for sharing and mode analysis. Proving the soundness of the abstract sharing, freeness, and linearity analysis presented here is done by a careful study of the behavior of these properties under the application of concrete idempotent substitutions. Other properties and operations for abstract equation systems can be based on related properties of concrete terms and equation systems [17, 19].

Section 2 describes the notational conventions and basic definitions related to abstract equation systems. In Section 3, the transition system for abstract unification is presented. Section 4 is concerned with the soundness proof and Section 5 clarifies the relationship between the composite domain and a few other domains for sharing and freeness analysis that appeared in the recent literature. In Appendix A, preliminary specifications of the domain dependent operations are given and Appendix B shows a simple example.

2 Abstract Equation Systems and Their Semantics

We adopt the convention that X, X_i, Y, Z, etc. denote arbitrary *concrete variables* (Var denotes a denumerable set of concrete variables); as in [5], *abstract variables* A, A_i, A', etc. (from the denumerable set AVar) are introduced to represent terms whose internal structure is hidden by abstraction (we assume $\mathsf{AVar} \cap \mathsf{Var} = \emptyset$); t, t_i, τ, τ_i, denote *mixed terms* ($\in \mathsf{MTerm}$), i.e., terms over $\mathsf{Var} \cup \mathsf{AVar}$. The set of normal terms over the set of concrete variables Var is denoted Term. The sets of concrete variables and abstract variables occurring in a syntactic expression e are denoted by $\mathsf{Var}(e)$ and $\mathsf{AVar}(e)$, respectively. The power set of S is denoted $\wp(S)$ and by $\doteq$ we denote syntactic identity. By ∇ we denote symmetric set difference, i.e., $S \nabla R = (S \setminus R) \cup (R \setminus S)$.

An *equation* over a set T of terms is an object of the form $t_1 = t_2$ where $t_1, t_2 \in T$. An *equation system* over T is a finite set of equations over T. Given an equation system E and an equation e, we let $e :: E$ denote the set $\{e\} \cup E$. The set of equation systems over Term, denoted Eqs, is referred to as the set of *concrete equation systems*. Abstract equation systems involve equations over MTerm.

Definition 2.1 (Abstract equation system) *An abstract equation system* AEqs_V*, for a set of variables* V*, is a pair* $\langle \mathcal{E}; \Delta \rangle$ *such that* $\mathcal{E}$ *is an equation system over the set of mixed terms* MTerm*, with* $\mathsf{Var}(\mathcal{E}) \subseteq V$*,* $\Delta \subseteq \wp(\mathsf{AVar}(\mathcal{E}))$ *a sharing component, and* ann_{AEqs} *an annotation mapping,* $ann_{\mathsf{AEqs}} : \mathsf{AVar}(\mathsf{AEqs}_V) \to \{\{f, \ell\}, \{c, \ell\}, \{\ell\}, \{c\}, \emptyset\}$*, such that the abstract variables of* AEqs_V *that do not occur in* Δ *have annotation* $\{c, \ell\}$*.*

The set V ($\subseteq \mathsf{Var}$) of program variables that are of interest is omitted as a subscript in AEqs_V when it is not important or clear from the context. The set of abstract equation systems is denoted $\mathcal{A}Eqs$. We further assume that a bottom abstract equation sytem, $\bot$, is added to $\mathcal{A}Eqs$.

We follow Jacobs and Langen [21] in the representation of the sharing component. Each set $\{A_1, \ldots, A_n\}$ indicates that the terms represented by the abstract variables $A_1, \ldots, A_n$ can share one or more variables. Moreover, the annotations on abstract variables enrich the domain with linearity[1] information as captured by the sharing domain of Søndergaard [31] and with freeness and compoundness information. In particular, abstract variables from $\mathcal{E}$ that do not occur in Δ represent ground, and thus compound linear, terms. To state that a concrete term t is ground, compound, linear, or a free variable, we use the predicates $ground(t)$, $compound(t)$, $linear(t)$, and $free(t)$ respectively. The basic unification algorithm for the set-sharing domain [20, 21] exploits groundness information. Freeness information is used to improve the precision of the original algorithm in, e.g., [30]. The algorithm presented in Section 3 uses linearity and compoundness information to derive even more accurate sharing information.

Let $\mathsf{AEqs} = \langle \mathcal{E}; \Delta \rangle$ be an abstract equation system. We define the following predicates on the abstract variables occurring in AEqs:

$ground_{\mathsf{AEqs}}(A) \Leftrightarrow A \notin \mathsf{AVar}(\Delta)$; $\quad free_{\mathsf{AEqs}}(A) \Leftrightarrow ann_{\mathsf{AEqs}}(A) = \{f, \ell\}$;
$compound_{\mathsf{AEqs}}(A) \Leftrightarrow ann_{\mathsf{AEqs}}(A) \supseteq \{c\}$; $\quad share_{\mathsf{AEqs}}(A_1, A_2) \Leftrightarrow \exists S \in \Delta.\ \{A_1, A_2\} \subseteq S$.
$linear_{\mathsf{AEqs}}(A) \Leftrightarrow ann_{\mathsf{AEqs}}(A) \supseteq \{\ell\}$;

Basically, AEqs describes a concrete equation system Eqs if it is possible to map each of the abstract variables in $\mathcal{E}$ to a concrete term and obtain Eqs. However, the mapping is

[1] A term is linear if no variable occurs more than once in it.

required to preserve the information specified by Δ and ann_{AEqs}. The description relation $\propto$ on $\mathcal{A}Eqs \times Eqs$ is formalized similarly as in [5, 6].

Definition 2.2 (Abstract term replacement) *An* abstract term replacement μ *for an abstract equation system* $\mathsf{AEqs}_V = \langle \mathcal{E}; \Delta \rangle$ *is a mapping* $\mu : \mathsf{AVar}(\mathsf{AEqs}_V) \rightarrow \mathsf{Term}$ *such that* $\forall A \in \mathsf{AVar}(\mathsf{AEqs}_V).\ \mathsf{Var}(A\mu) \cap V = \emptyset$.[2] *We define* $\mathsf{Range}(\mu) = \cup_{A \in \mathsf{Dom}(\mu)} \mathsf{Var}(A\mu)$. *The abstract term replacement* μ *is said to be* Δ-preserving *if for all* $A \in \mathsf{AVar}(\mathsf{AEqs}_V)$:[3] *(1)* $ground_{\mathsf{AEqs}}(A) \Rightarrow ground(A\mu)$; *(2)* $compound_{\mathsf{AEqs}}(A) \Rightarrow compound(A\mu)$; *(3)* $linear_{\mathsf{AEqs}}(A) \Rightarrow linear(A\mu)$; *(4)* $free_{\mathsf{AEqs}}(A) \Rightarrow free(A\mu)$; *and (5) for every* $X \in \mathsf{Range}(\mu)$. $SG^{\mu}_{\mathsf{AEqs}}(X) \in \Delta$ *where* $SG^{\mu}_{\mathsf{AEqs}}(X) \stackrel{def}{=} \{A \in \mathsf{AVar}(\mathsf{AEqs}_V) \mid X \in \mathsf{Var}(A\mu)\}$. *The notion of* sharing group *is similar as in [21].*

The abstract term replacement μ *is extended to a mapping for mixed terms and equation systems over* $\mathsf{Var} \cup \mathsf{AVar}(\mathsf{AEqs}_V)$ *as follows:*

1. $X\mu = X$ *for* $X \in \mathsf{Var}$;
2. $f(\tau_1, \ldots, \tau_n)\mu = f(\tau_1\mu, \ldots, \tau_n\mu)$;
3. $(\tau_1 = \tau_2)\mu = (\tau_1\mu = \tau_2\mu)$;
4. $(\varepsilon :: \mathcal{E})\mu = (\varepsilon\mu :: \mathcal{E}\mu)$;

Definition 2.3 (Equation description) *An abstract equation system* $\mathsf{AEqs}_V = \langle \mathcal{E}; \Delta \rangle$ describes *a concrete equation system* Eqs *via* μ, *denoted* $\mathsf{AEqs}_V \propto_{\mu} \mathsf{Eqs}$, *if* μ *is a* Δ*-preserving abstract term replacement for* AEqs_V *such that* $\mathcal{E}\mu = \mathsf{Eqs}$. *We write* $\mathsf{AEqs}_V \propto \mathsf{Eqs}$ *to denote that there exists an abstract term replacement* μ *such that* $\mathsf{AEqs}_V \propto_{\mu} \mathsf{Eqs}$. *The abstract equation system* $\perp$ *describes no concrete equation systems; it corresponds to failure in a concrete derivation.*

Let $\mathsf{AEqs} = \langle \mathcal{E}; \Delta \rangle$ be an abstract equation system. In the examples below, we use an implicit representation of the associated annotation mapping ann_{AEqs} by annotating the occurrences of each abstract variable A in AEqs to indicate the value of $ann_{\mathsf{AEqs}}(A)$: A^f, $A^{c\ell}$, A^{ℓ}, A^c, and A denote respectively that $ann_{\mathsf{AEqs}}(A)$ is $\{f, \ell\}$, $\{c, \ell\}$, $\{\ell\}$, $\{c\}$, and $\emptyset$.

Example 2.1 *Let* $\mathsf{AEqs}_V = \langle \mathcal{E}; \Delta \rangle$ *with* $V = \{X, Y, Z\}$, $\mathcal{E} = \{X = A_1^{c\ell}, Y = g(A_2^f, A_3^{\ell}), Z = A_4\}$, *and* $\Delta = \{\{A_2^f, A_4\}, \{A_3^{\ell}, A_4\}, \{A_3^{\ell}\}\}$. *Let* $\mathsf{Eqs} = \{X = f(5), Y = g(U, g(W_1, W_2)), Z = h(W_1, U, U)\}$. *The mapping* $\mu : \{A_1, A_2, A_3, A_4\} \rightarrow \mathsf{Term}$, *defined by* $A_1 \mapsto f(5), A_2 \mapsto U, A_3 \mapsto g(W_1, W_2), A_4 \mapsto h(W_1, U, U)$ *is a* Δ*-preserving abstract term replacement for* AEqs_V *such that* $\mathsf{AEqs}_V \propto_{\mu} \mathsf{Eqs}$.

An abstract equation from AEqs_V corresponds to precisely one concrete equation in Eqs; this is unlike the abstraction used in [6] where one abstract equation can correspond to a set of concrete equations. On the other hand, different abstract equations can be mapped to the same concrete equation by a Δ-preserving abstract term replacement μ, e.g., consider $\mathcal{E} = \{X = A_1^{\ell}, X = A_2\}$, $\Delta = \{\{A_1^{\ell}, A_2\}\}$, and $\mu : A_1 \mapsto U, A_2 \mapsto U$. Moreover, a Δ-preserving abstract term replacement μ may result in an unsatisfiable concrete equation system $\mathcal{E}\mu$, e.g., take $\mu : A_1 \mapsto f(U), A_2 \mapsto g(U)$. An abstract equation system $\mathsf{AEqs}_V = \langle \mathcal{E}; \Delta \rangle$ is said to be *unifiable* if and only if there exists a Δ-preserving abstract term replacement μ such that $\mathcal{E}\mu$ is unifiable.

With the current definition of equation description, the abstract equation system $\mathsf{AEqs}_V = \langle \{X = A_1, Y = A_2\}; \{\{A_1, A_2\}\} \rangle$ with $V = \{X, Y\}$, describes the concrete equation system $\mathsf{Eqs}_1 = \{X = f(U), Y = U\}$ but not the equation system $\mathsf{Eqs}_2 = \{X = f(Y), W = h(U), U = Y\}$ which is equivalent with respect to the variables in V.

[2] Note that, similar as for normal substitutions, we use $A\mu$ to denote $\mu(A)$.

[3] If $\mathsf{AVar}(\mathsf{AEqs}_V)$ is the empty set, then μ is the empty mapping which is Δ-preserving.

Definition 2.4 (Equivalence of equation systems) *Two concrete 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* $\theta_1 \restriction V = \theta_2 \restriction V$*. All unsatisfiable equation systems (for which there does not exist a most general unifier) are defined to be equivalent with respect to* V*.*

For the example above, take $\theta_1 = \{X \leftarrow f(Y), U \leftarrow Y\}$ and $\theta_2 = \{X \leftarrow f(Y), W \leftarrow h(Y), U \leftarrow Y\}$. So, an abstract equation system AEqs_V describes (one or more) representatives of (a collection of) equivalence classes w.r.t. $\approx_V$. The concrete equation systems of an equivalence class have most general unifiers with identical freeness, structural, linearity, and sharing properties for the elements of V. The concrete unification algorithm transforms a representative of the class into another representative that is in solved form such that the satisfiability of the system is clear and a solution (an mgu) immediately derivable. Similarly, the abstract rewriting algorithm of Section 3 yields abstract equation systems that describe representatives of a collection of $\approx_V$-classes from which the satisfiability is immediate and an mgu directly derivable. However, the abstract unification algorithm may loose precision in the sense that the final abstract solved form may describe representatives of equivalence classes that are not represented by the initial abstract system AEqs.

Based on the definition of equivalence of concrete equation systems, we define a preorder and an equivalence relation for abstract equation systems $\mathsf{AEqs}^1_V, \mathsf{AEqs}^2_V \in \mathcal{A}Eqs$:

$$\begin{array}{lcl} \mathsf{AEqs}^1_V \leq \mathsf{AEqs}^2_V & \Leftrightarrow & \forall \mathsf{Eqs}^1 \in Eqs.\ \mathsf{Eqs}^1 \text{ is unifiable } \ \&\ \mathsf{AEqs}^1_V \propto \mathsf{Eqs}^1 \rightarrow \\ & & \exists \mathsf{Eqs}^2 \in Eqs.\ \mathsf{Eqs}^2 \text{ is unifiable } \ \&\ \mathsf{AEqs}^2_V \propto \mathsf{Eqs}^2 \ \&\ \mathsf{Eqs}^1 \approx_V \mathsf{Eqs}^2, \\ \mathsf{AEqs}^1_V \equiv \mathsf{AEqs}^2_V & \Leftrightarrow & \mathsf{AEqs}^1_V \leq \mathsf{AEqs}^2_V \ \&\ \mathsf{AEqs}^2_V \leq \mathsf{AEqs}^1_V. \end{array}$$

From these definitions, it follows that $\forall \mathsf{AEqs}.\ \bot \leq \mathsf{AEqs}$, and $\forall \mathsf{AEqs}.\ \mathsf{AEqs}$ is not unifiable $\leftrightarrow \bot \equiv \mathsf{AEqs}$. The $\equiv$-relation in fact lifts the $\approx_V$ relation on concrete equation systems to the domain of abstract equation systems. The relation $\leq$ is reflexive and transitive, thus a preorder on the domain of abstract equation systems $\mathcal{A}Eqs$, and induces a partial ordering on the set of equivalence classes w.r.t. $\equiv$. The resulting poset has no infinite (strict) ascending chains. A proof of the latter can be constructed by using a size relation as in [25] for terms or by defining a degree for substitutions as in [17][4], and is further based on the finiteness of the set of possible annotations for abstract variables. Consequently, when $\mathcal{A}Eqs/\equiv,\leq$ is used in a framework of abstract interpretation (e.g., [3, 4, 8, 15, 22, 28]) a terminating analysis is guaranteed.

Example 2.2 *Let* $\mathsf{AEqs}^1_V = \langle\{X = f(Y)\}; \emptyset\rangle$ *and* $\mathsf{AEqs}^2_V = \langle\{X = A_1, Y = A_2^\ell\}; \{\{A_1, A_2^\ell\}\}\rangle$ *with* $V = \{X, Y\}$*, then* $\mathsf{AEqs}^1_V \leq \mathsf{AEqs}^2_V$ *and* $\mathsf{AEqs}^2_V \not\leq \mathsf{AEqs}^1_V$*. On the other hand, let* $\mathsf{AEqs}^1_{V'} = \langle\{X = f(Y)\}; \emptyset\rangle$ *and* $\mathsf{AEqs}^2_{V'} = \langle\{X = A_1^f, Y = A_2^f, A_1^f = f(A_2^f), Z = A_3^f\}; \{\{A_1^f\}, \{A_2^f\}, \{A_3^f\}\}\rangle$ *with* $V' = \{X, Y, Z\}$*, then* $\mathsf{AEqs}^1_{V'} \equiv \mathsf{AEqs}^2_{V'}$*.*

We introduce some further notation and predicates that are convenient in the abstract unification algorithm of the next section. For $SS \subseteq \mathsf{AVar}$, we denote $\Delta(SS) = \{S \in \Delta | SS \cap S \neq \emptyset\}$ and $\bar{\Delta}(SS) = \{S \in \Delta | SS \cap S = \emptyset\}$.[5] If we want to indicate explicitly that

[4] Roughly stated, the degree of a substitution σ is the number of symbol occurrences in the image of the domain and range variables of σ minus the number of different variables appearing therein.

[5] In the notation of Jacobs and Langen [21], $\Delta(SS)$ corresponds to $\mathrm{rel}(SS, \Delta)$ and $\bar{\Delta}(SS)$ corresponds to $\Delta \setminus \mathrm{rel}(SS, \Delta)$. In the case that $SS = \emptyset$, we have $\Delta(\emptyset) = \emptyset$ and $\bar{\Delta}(\emptyset) = \Delta$.

a mixed term τ contains only the variables $\{X_1, \ldots, X_m, A_1, \ldots, A_n\}$, with $(m \geq 0, n \geq 0)$, and no other variables, τ is denoted by $\tau[X_1, \ldots, X_m, A_1, \ldots, A_n]$. We also use the predicate $compound(\tau)$ to state that the mixed term τ is not a concrete nor an abstract variable. We generalize the predicates defined above for abstract variables to mixed terms occurring in AEqs.

$ground_{\mathsf{AEqs}}(\tau) \Leftrightarrow \forall\Delta$-preserving abstract term replacement μ. $ground(\tau\mu)$;
$compound_{\mathsf{AEqs}}(\tau) \Leftrightarrow \forall\Delta$-preserving abstract term replacement μ. $compound(\tau\mu)$;
$linear_{\mathsf{AEqs}}(\tau) \Leftrightarrow \forall\Delta$-preserving abstract term replacement μ. $linear(\tau\mu)$;
$free_{\mathsf{AEqs}}(\tau) \Leftrightarrow \forall\Delta$-preserving abstract term replacement μ. $free(\tau\mu)$;
$share_{\mathsf{AEqs}}(\tau, \tau') \Leftrightarrow \exists\Delta$-preserving abstract term replacement μ. $\mathsf{Var}(\tau\mu) \cap \mathsf{Var}(\tau'\mu) \neq \emptyset$.

For the case that τ contains only abstract variables, alternative more pragmatic definitions are in Appendix A. They are expressed in terms of the properties of the abstract variables occurring in τ instead of in terms of Δ-preserving abstract term replacements.

Finally, it is important to note that the freeness, compoundness, and sharing information specified by an abstract equation system is not orthogonal. Consider $\mathsf{AEqs}_V = \langle \mathcal{E}; \Delta \rangle$ with $V = \{X, Y, Z\}$, $\mathcal{E} = \{X = A_1^f, Y = g(A_2^f), Z = g(A_3^\ell, A_4)\}$ and $\Delta = \{\{A_1^f, A_2^f, A_4\}, \{A_1^f, A_3^\ell\}\}$. Then every Δ-preserving abstract term replacement μ maps A_1^f and A_2^f to the same concrete variable; since $\{A_1^f, A_2^f, A_4\}$ is the only sharing group that contains the definitely free abstract variable A_2, it indicates in fact 'definite' sharing between A_1^f, A_2^f, and A_4, and the sharing group $\{A_1^f, A_3^\ell\}$ is redundant in the representation. The system AEqs_V is equivalent to $\mathsf{AEqs}'_V = \langle \mathcal{E}'; \Delta' \rangle$ where $\mathcal{E}' = \{X = A_1^f, Y = g(A_1^f), Z = g(A_3^{c\ell}, A_4)\}$ and $\Delta' = \{\{A_1^f, A_4\}\}$. If we consider the same equation set and sharing component, but with variable A_3 annotated to be definitely free instead of definitely linear, the abstract equation system would not be unifiable since there does not exist an abstract term replacement μ that maps A_1 and A_2 to the same concrete variable, and also A_1 and A_3, but not A_2 and A_3. An equivalence preserving transformation removing some of these redundancies is obtained by applying the function Reduce.

Definition 2.5 (Reduce) *Let* $\mathsf{AEqs} = \langle \mathcal{E}; \Delta \rangle$ *be an abstract equation system.*

$$
\mathsf{Reduce}(\mathsf{AEqs}) = \begin{cases}
\textit{if} & \neg\,(\exists A_1, A_2 \in \mathsf{AVar}(\mathsf{AEqs}).\ A_1 \not\equiv A_2\ \&\\
& free_{\mathsf{AEqs}}(A_1)\ \&\ free_{\mathsf{AEqs}}(A_2)\ \&\\
& (\Delta(\{A_1\}) \subseteq \Delta(\{A_2\}) \vee \Delta(\{A_2\}) \subseteq \Delta(\{A_1\}))\ )\\
\textit{then} & \langle \mathcal{E}; \Delta \rangle \textit{ with } ann_{\langle \mathcal{E};\Delta \rangle}(A) =\\
& \quad \begin{cases}
ann_{\mathsf{AEqs}}(A) \cup \{c\} & \textit{if } \exists A_1, A_2 \in \mathsf{AVar}(\mathsf{AEqs}).\ (A_1 \not\equiv A_2\\
& \quad \& \ free_{\mathsf{AEqs}}(A_1)\ \&\ free_{\mathsf{AEqs}}(A_2)\\
& \quad \& \ \neg share_{\mathsf{AEqs}}(A_1, A_2)\ \&\\
& \quad \Delta(\{A_1, A_2\}) \subseteq \Delta(\{A\})),\\
ann_{\mathsf{AEqs}}(A) & \textit{otherwise},
\end{cases}\\
\textit{else if} & \exists A_3.\ (free_{\mathsf{AEqs}}(A_3)\ \&\ \Delta(\{A_3\}) \subseteq (\Delta(\{A_1\}) \nabla \Delta(\{A_2\})))\\
\textit{then} & \bot\\
\textit{else} & \mathsf{Reduce}(\langle \mathcal{E}'; \Delta' \rangle) \textit{ where}\\
& \langle \mathcal{E}'; \Delta' \rangle = \langle \mathcal{E}[A_2 \leftarrow A_1]; \Delta''[A_2 \leftarrow A_1] \rangle,\\
& \Delta'' = \Delta \setminus (\Delta(\{A_1\}) \nabla \Delta(\{A_2\})),\\
& \textit{and } \forall A \in \mathsf{AVar}(\mathcal{E}').\\
& \quad ann_{\langle \mathcal{E}';\Delta' \rangle}(A) = \begin{cases}
ann_{\mathsf{AEqs}}(A) & \textit{if } A \in \mathsf{AVar}(\Delta'),\\
\{c, \ell\} & \textit{otherwise.}
\end{cases}
\end{cases}
$$

If the abstract unification algorithm of Section 3 starts from an abstract equation system that is not in reduced form, then it may introduce imprecision that is avoidable.

3 Abstract Unification Algorithm

By analogy with the Martelli-Montanari [29] algorithm, abstract unification is formalised as a transition system which computes a single solved form. For each of the rules in this well-known algorithm, there are basically two corresponding variant rules in the abstract algorithm (one for concrete variables, one for abstract variables). Since abstract variables are of a secondary nature (they serve to identify subterms with which useful information is associated from the analysis perspective), some rules are introduced to remove equations and/or abstract variables that no longer carry relevant information from the representation. The rules 1 to 10 can be seen as realizing some sort of pre-unification [31] preserving structural information as much as possible. Rule 11 prepares an abstract equation for application of the general rule 12, which solves an 'abstract' equation in the underlying domain used for sharing and mode information. In the composite domain developed here, rule 12 corresponds to the basic unification algorithm of Jacobs and Langen, but it takes advantage of the freeness and linearity information available in the domain through a careful case analysis, avoiding the "closure under union" operation whenever possible (see Section 4 for the correctness proof).

Algorithm 1 (Abstract unification) *Let* AEqs *be an abstract equation system. Non-deterministically choose an equation from* AEqs *to which a numbered step applies. The action taken by the algorithm is determined by the form of the equation. Let* AEqs' *denote the resulting abstract equation system. The associated annotation mapping,* $ann_{\mathsf{AEqs}'}$*, is implicitly defined to be* $ann_{\mathsf{AEqs}}{\restriction}\mathsf{AVar}(\mathsf{AEqs}')$ *unless stated otherwise. The algorithm terminates when there is no equation on which a step applies.*

1. $\langle X = X :: \mathcal{E}; \Delta\rangle \overset{\text{remove}}{\rightarrow} \langle \mathcal{E}; \Delta\rangle.$

2. $\langle A = A :: \mathcal{E}; \Delta\rangle \overset{\text{remove}}{\rightarrow} \begin{cases} \langle \mathcal{E};\ \{S \setminus \{A\} \mid S \in \Delta\} \setminus \{\emptyset\}\rangle & \text{if } A \notin \mathsf{AVar}(\mathcal{E}), \\ \langle \mathcal{E}; \Delta\rangle & \text{otherwise.} \end{cases}$

3. $\langle X = A :: \mathcal{E}; \Delta\rangle \overset{\text{remove}}{\rightarrow} \langle \mathcal{E}; \Delta \setminus \{\{A\}\}\rangle$ *if* $free_{\mathsf{AEqs}}(A)$ & $X \notin \mathsf{Var}(\mathcal{E})$ & $A \notin \mathsf{AVar}(\mathcal{E})$ & $\neg(\exists \hat{A} \in \mathsf{AVar}(\mathcal{E}).\ A \neq \hat{A}$ & $share_{\mathsf{AEqs}}(A, \hat{A}))$.

4. $\langle t = X :: \mathcal{E}; \Delta\rangle \overset{\text{switch}}{\rightarrow} \langle X = t :: \mathcal{E}; \Delta\rangle$ *if* $t \notin \mathsf{Var}$.

5. $\langle t = A :: \mathcal{E}; \Delta\rangle \overset{\text{switch}}{\rightarrow} \langle A = t :: \mathcal{E}; \Delta\rangle$ *if* $t \notin \mathsf{Var} \cup \mathsf{AVar}$.

6. $\langle f(s_1, \ldots, s_n) = g(t_1, \ldots, t_m) :: \mathcal{E}; \Delta\rangle$
$\overset{\text{peel}}{\rightarrow} \langle \{s_1 = t_1, \ldots, s_n = t_n\} \cup \mathcal{E}; \Delta\rangle$ *if* $f/n \doteq g/m$,
$\overset{\text{fail}}{\rightarrow} \bot$ *otherwise.*

7. $\langle X = t :: \mathcal{E}; \Delta\rangle \overset{\text{fail}}{\rightarrow} \bot$ *if* $X \in \mathsf{Var}(t)$ & $t \neq X$.

8. $\langle X = t :: \mathcal{E}; \Delta\rangle \overset{\text{subst}}{\rightarrow} \langle X = t :: \mathcal{E}[X \leftarrow t]; \Delta\rangle$ *if* $X \notin \mathsf{Var}(t)$ & $X \in \mathsf{Var}(\mathcal{E})$.

9. $\langle A = t :: \mathcal{E}; \Delta\rangle \overset{\text{fail}}{\rightarrow} \bot$ *if* $A \in \mathsf{AVar}(t)$ & $t \neq A$.

10. $\langle A = t :: \mathcal{E}; \Delta\rangle \overset{\text{subst}}{\rightarrow} \langle A = t :: \mathcal{E}[A \leftarrow t]; \Delta\rangle$ *if* $t \notin \mathsf{Var}$ & $A \notin \mathsf{AVar}(t)$ & $A \in \mathsf{AVar}(\mathcal{E})$.

11. $\langle A = t :: \mathcal{E};\ \Delta\rangle \overset{\text{fresh \& subst}}{\rightarrow} \langle \{X_1 = A'_1, \ldots, X_n = A'_n\} \cup (A = t :: \mathcal{E})[\{X_1 \leftarrow A'_1, \ldots, X_n \leftarrow A'_n\}];$[6] $\Delta \cup \{\{A'_1\}, \ldots, \{A'_n\}\}\rangle$

with $ann_{\mathsf{AEqs}'}(\hat{A}) = \begin{cases} \{f, \ell\}, & \text{if } \hat{A} \doteq A'_i, \text{ for } 1 \leq i \leq n, \\ ann_{\mathsf{AEqs}}(\hat{A}), & \text{otherwise,} \end{cases}$

if $t \notin \mathsf{Var}$ & $A \notin \mathsf{AVar}(\mathcal{E}) \cup \mathsf{AVar}(t)$ & $\mathsf{Var}(t) = \{X_1, \ldots, X_n\} \neq \emptyset$ & $A'_1, \ldots, A'_n$ *are new abstract variables not occurring in the system.*

12. $\langle A = \tau[A_1, \ldots, A_n] :: \mathcal{E}; \Delta\rangle$ *with* $(n \geq 0)$ *and* $A \notin \mathsf{AVar}(\mathcal{E}) \cup \{A_1, \ldots, A_n\}$

$\overset{\text{fail}}{\rightarrow} \bot$ *if* $(free_{\mathsf{AEqs}}(A)$ & $compound_{\mathsf{AEqs}}(\tau[A_1, \ldots, A_n]) \vee$ $compound_{\mathsf{AEqs}}(A)$ & $free_{\mathsf{AEqs}}(\tau[A_1, \ldots, A_n]))$ & $\exists \hat{A} \in \mathsf{AVar}(\mathcal{E}) \cup \{A, A_1, \ldots, A_n\}.\ (free_{\mathsf{AEqs}}(\hat{A})$ & $\Delta(\{\hat{A}\}) \subseteq \Delta(\{A\}) \cap \Delta(\{A_1, \ldots, A_n\}))$

$\overset{\text{solve}}{\rightarrow} \langle \mathcal{E}; \Delta'\rangle$ *otherwise,*

where,

(in the case that $free_{\mathsf{AEqs}}(A)$ & $compound_{\mathsf{AEqs}}(\tau[A_1, \ldots, A_n])$ *or* $compound_{\mathsf{AEqs}}(A)$ & $free_{\mathsf{AEqs}}(\tau[A_1, \ldots, A_n])$*, let the sharing component* Δ *be first reduced with the set* $\Delta(\{A\}) \cap \Delta(\{A_1, \ldots, A_n\})$ *in)*

$\Delta' =$

if $ground_{\mathsf{AEqs}}(\tau[A_1, \ldots, A_n]) \vee ground_{\mathsf{AEqs}}(A)$

then $\bar{\Delta}(\{A, A_1, \ldots, A_n\})$

else $\bar{\Delta}(\{A, A_1, \ldots, A_n\}) \cup$

$\{\ (S_\tau \cup S_A) \setminus (\{A, A_1, \ldots, A_n\} \setminus \mathsf{AVar}(\mathcal{E}))$

- *if* $free_{\mathsf{AEqs}}(A) \vee free_{\mathsf{AEqs}}(\tau[A_1, \ldots, A_n]) \vee (linear_{\mathsf{AEqs}}(A)$ & $\neg share_{\mathsf{AEqs}}(\tau[A_1, \ldots, A_n], A))$ *then* $S_\tau \in \Delta(\{A_1, \ldots, A_n\})$ *else* $S_\tau = \cup SS$ *where* $SS \subseteq \Delta(\{A_1, \ldots, A_n\})$ & $SS \neq \emptyset$
- *if* $free_{\mathsf{AEqs}}(A) \vee free_{\mathsf{AEqs}}(\tau[A_1, \ldots, A_n]) \vee (linear_{\mathsf{AEqs}}(\tau[A_1, \ldots, A_n])$ & $\neg share_{\mathsf{AEqs}}(\tau[A_1, \ldots, A_n], A))$ *then* $S_A \in \Delta(\{A\})$ *else* $S_A = \cup SS$ *where* $SS \subseteq \Delta(\{A\})$ & $SS \neq \emptyset$

$\}\ \setminus \{\emptyset\}$,

and for $\hat{A} \in \mathsf{AVar}(\mathcal{E}) \setminus \mathsf{AVar}(\Delta')$ *(ground variable) :* $ann_{\mathsf{AEqs}'}(\hat{A}) = \{c, \ell\}$, *for* $\hat{A} \in \mathsf{AVar}(\Delta')$ *such that* $\exists i.\ \hat{A} \doteq A_i$ *(variable from right-hand-side) :*

$ann_{\mathsf{AEqs}'}(\hat{A}) =$

[6] Note that rule 11 involves a substitution, which could alternatively be obtained by applications of rule 8; however, by successively selecting rule 11 without the immediate substitution of the newly introduced abstract variables, the algorithm would become non-terminating.

$$
\begin{array}{l}
\quad \textit{if} \quad \mathit{compound}_{\mathsf{AEqs}}(A) \ \&\ \neg \mathit{compound}_{\mathsf{AEqs}}(\tau[A_1,\ldots,A_n]) \\
\qquad \textit{then} \quad \mathit{ann}_{aux} \cup \{c\} \quad \textit{(since in this case } c \in \mathit{ann}_{\mathsf{AEqs}}(A) \textit{ and } \tau[A_1,\ldots,A_n] \doteq \hat{A}\textit{)} \\
\qquad \textit{else} \quad \mathit{ann}_{aux} \\
\quad \textit{where} \quad \mathit{ann}_{aux} = \\
\qquad \textit{if} \quad \mathit{free}_{\mathsf{AEqs}}(\hat{A}) \ \&\ \mathit{free}_{\mathsf{AEqs}}(A) \\
\qquad\quad \textit{then} \quad \mathit{ann}_{\mathsf{AEqs}}(\hat{A}) \quad \textit{(freeness and linearity preserved)} \\
\qquad \textit{else if} \quad \mathit{linear}_{\mathsf{AEqs}}(A) \ \&\ (\neg \mathit{share}_{\mathsf{AEqs}}(A,\hat{A}) \vee \mathit{free}_{\mathsf{AEqs}}(\tau[A_1,\ldots,A_n])) \\
\qquad\quad \textit{then} \quad \mathit{ann}_{\mathsf{AEqs}}(\hat{A}) \setminus \{f\} \quad \textit{(compoundness, linearity preserved)} \\
\qquad \textit{else} \quad \mathit{ann}_{\mathsf{AEqs}}(\hat{A}) \setminus \{f,\ell\}, \quad \textit{(compoundness preserved)}
\end{array}
$$

for $\hat{A} \in \mathsf{AVar}(\Delta')$ *such that* $\hat{A}$ *is* not *a variable from the right-hand-side :*

$$
\begin{array}{l}
\mathit{ann}_{\mathsf{AEqs}'}(\hat{A}) = \\
\quad \textit{if} \quad \mathit{free}_{\mathsf{AEqs}}(A) \ \&\ \mathit{compound}_{\mathsf{AEqs}}(\tau[A_1,\ldots,A_n]) \,\&\, \left(\begin{array}{l} \Delta(\{\hat{A}\}) \subseteq \Delta(\{A\}) \vee \\ \Delta(\{A\}) \subseteq \Delta(\{\hat{A}\}) \end{array} \right) \vee \\
\quad \mathit{compound}_{\mathsf{AEqs}}(A) \ \&\ \mathit{free}_{\mathsf{AEqs}}(\tau[A_1,\ldots,A_n]) \,\&\, \left(\begin{array}{l} \Delta(\{\hat{A}\}) \subseteq \Delta(\{A_1,\ldots,A_n\}) \vee \\ \Delta(\{A_1,\ldots,A_n\}) \subseteq \Delta(\{\hat{A}\}) \end{array} \right) \\
\qquad \textit{then} \quad \mathit{ann}_{aux} \cup \{c\} \ ^{7} \\
\qquad \textit{else} \quad \mathit{ann}_{aux} \\
\quad \textit{where} \quad \mathit{ann}_{aux} = \\
\qquad \textit{if} \quad (\neg \mathit{share}_{\mathsf{AEqs}}(\tau[A_1,\ldots,A_n],\hat{A}) \ \&\ \neg \mathit{share}_{\mathsf{AEqs}}(A,\hat{A})) \qquad (1) \\
\qquad \vee \quad (\mathit{free}_{\mathsf{AEqs}}(\tau[A_1,\ldots,A_n]) \ \&\ \mathit{free}_{\mathsf{AEqs}}(A) \ \&\ \mathit{free}_{\mathsf{AEqs}}(\hat{A})) \qquad (2) \\
\qquad \vee \quad (\mathit{free}_{\mathsf{AEqs}}(A) \ \&\ \neg \mathit{share}_{\mathsf{AEqs}}(A,\hat{A})) \qquad (3) \\
\qquad \vee \quad (\mathit{free}_{\mathsf{AEqs}}(\tau[A_1,\ldots,A_n]) \ \&\ \neg \mathit{share}_{\mathsf{AEqs}}(\tau[A_1,\ldots,A_n],\hat{A})) \qquad (4) \\
\qquad \textit{then} \quad \mathit{ann}_{\mathsf{AEqs}}(\hat{A}) \quad \textit{(freeness or compoundness, and linearity preserved)} \\
\quad \textit{else} \\
\qquad \textit{if} \quad (\mathit{free}_{\mathsf{AEqs}}(A) \ \&\ \mathit{linear}_{\mathsf{AEqs}}(\tau[A_1,\ldots,A_n]) \ \&\ \mathit{free}_{\mathsf{AEqs}}(\hat{A})) \qquad (5) \\
\qquad \vee \quad (\mathit{free}_{\mathsf{AEqs}}(\tau[A_1,\ldots,A_n]) \ \&\ \mathit{linear}_{\mathsf{AEqs}}(A) \ \&\ \mathit{free}_{\mathsf{AEqs}}(\hat{A})) \qquad (6) \\
\qquad \vee \quad (\mathit{linear}_{\mathsf{AEqs}}(A) \ \&\ \neg \mathit{share}_{\mathsf{AEqs}}(A,\hat{A})) \qquad (7) \\
\qquad \vee \quad (\mathit{linear}_{\mathsf{AEqs}}(\tau[A_1,\ldots,A_n]) \ \&\ \neg \mathit{share}_{\mathsf{AEqs}}(\tau[A_1,\ldots,A_n],\hat{A})) \qquad (8) \\
\qquad \textit{then} \quad \mathit{ann}_{\mathsf{AEqs}}(\hat{A}) \setminus \{f\} \quad \textit{(compoundness, linearity preserved)} \\
\quad \textit{else} \quad \mathit{ann}_{\mathsf{AEqs}}(\hat{A}) \setminus \{f,\ell\} \quad \textit{(compoundness preserved)}
\end{array}
$$

We do not need the annotation of A*, since* $A \notin \mathsf{AVar}(\langle \mathcal{E}; \Delta' \rangle)$*.*

The abstract unification algorithm is deterministic in the sense that at most one rule is applicable to the selected abstract equation. The algorithm always terminates and results either in $\bot$ or an *abstract equation system in solved form*, i.e., an abstract equation system that only contains equations of the form $X = t$, with $X \in \mathsf{Var}$ and $t \in \mathsf{MTerm}$, such that the variables of the left-hand-sides are all different and do not occur in the right-hand-sides.

[7] From $\mathit{free}_{\mathsf{AEqs}}(A)$ & $(\Delta(\{\hat{A}\}) \subseteq \Delta(\{A\}) \vee \Delta(\{A\}) \subseteq \Delta(\{\hat{A}\}))$, it follows that for any Δ-preserving abstract term replacement μ, $A\mu \in \mathsf{Var}(\hat{A}\mu)$ or $\hat{A}\mu$ is ground; thus, if $A\mu$ becomes compound, so is $\hat{A}\mu$. The other case is symmetric.

Note that an abstract equation system in solved form only describes concrete equation systems (if any) that are in solved form (thus, unifiable equation systems).

Theorem 1 (Termination) *Algorithm 1 for abstract unification terminates for any input abstract equation system and results either in $\bot$ or an abstract equation system in solved form.*

Proof The proof is similar as the proof of termination of the concrete unification algorithm in [25]. Applications of rules 1, 2, 3, and 6 strictly diminish the total number of occurrences of variables and function symbols on the left-hand-sides of equations. Rules 4 and 5 can be applied only a finite number of times before applying another rule, and their application does not increase the number of variables and function symbols on the left-hand-sides. The application of the rules 7 and 9 leads to failure and termination. Application of rule 12 either leads to failure or strictly diminishes the total number of occurrences of variables and function symbols on the left-hand-sides of equations. The application of the rules 8 and 10 eliminates all occurrences of the variable of the left-hand-side in all other abstract equations. Consequently, for a given (concrete or abstract) variable these rules can be performed only once. Moreover, there are only a finite number of concrete variables in the system. Rule 11 introduces new abstract variables, however, for each concrete variable in the system at most one new abstract variable can be created, so also the number of abstract variables that occurs in the transition system is finite. It follows that the algorithm terminates for any input.

Assume that the algorithm has not failed, then all the left-hand-sides of equations are *concrete* variables since otherwise one of the rules 4, 5, or 6 could be further applied if the left-hand-side is a non-variable term, or one of the rules 2, 4, 9, 10, 11, or 12 if the left-hand-side is an abstract variable. All the concrete variables are different and do not appear on the right-hand-side of an equation, otherwise rule 8 could be further applied. So, the algorithm yields an equation system in abstract solved form. □

By an inspection of the algorithm, we see that the only rule that can cause imprecision is rule 12. Moreover, the abstract unification algorithm is not confluent as the following example illustrates.

Example 3.1 *For each step, the underlined equation is the equation selected by the transformation rule(s) of which the number(s) is(are) indicated above the arrow. The set of variables of interest is $V = \{X, Y, Z, U\}$.*

$$\langle \left\{ \begin{array}{l} A_5^f = X,\ Z = h(A_4^\ell), \\ A_1^f = f(A_2^\ell, U, Z), \\ \underline{g(h(A_3^{c\ell}), Y, A_7) = g(A_2^\ell, A_6^f, A_7)} \end{array} \right\} ; \left\{ \begin{array}{l} \{A_6^f\}, \{A_7, A_6^f\}, \\ \{A_2^\ell, A_4^\ell\}, \{A_1^f\}, \\ \{A_1^f, A_5^f\}, \{A_5^f\} \end{array} \right\} \rangle \xrightarrow{6}$$

$$\langle \left\{ \begin{array}{l} A_5^f = X, \\ Z = h(A_4^\ell), \\ A_1^f = f(A_2^\ell, U, Z), \\ h(A_3^{c\ell}) = A_2^\ell, \\ Y = A_6^f,\ \underline{A_7 = A_7} \end{array} \right\} ; \left\{ \begin{array}{l} \{A_6^f\}, \{A_7, A_6^f\}, \\ \{A_2^\ell, A_4^\ell\}, \{A_1^f\}, \\ \{A_1^f, A_5^f\}, \{A_5^f\} \end{array} \right\} \rangle \xrightarrow{2} \langle \left\{ \begin{array}{l} \underline{A_5^f = X}, \\ \underline{Z = h(A_4^\ell)}, \\ A_1^f = f(A_2^\ell, U, Z), \\ h(A_3^{c\ell}) = A_2^\ell, \\ Y = A_6^f \end{array} \right\} ; \left\{ \begin{array}{l} \{A_6^f\}, \\ \{A_2^\ell, A_4^\ell\}, \\ \{A_1^f\}, \{A_5^f\}, \\ \{A_1^f, A_5^f\} \end{array} \right\} \rangle$$

$$\xrightarrow{4} \langle \left\{ \begin{array}{l} X = A_5^f,\ Z = h(A_4^\ell), \\ A_1^f = f(A_2^\ell, U, Z), \\ h(A_3^{c\ell}) = A_2^\ell,\ \underline{Y = A_6^f} \end{array} \right\} ; \left\{ \begin{array}{l} \{A_6^f\}, \\ \{A_2^\ell, A_4^\ell\}, \{A_1^f\}, \\ \{A_1^f, A_5^f\}, \{A_5^f\} \end{array} \right\} \rangle \xrightarrow{3}$$

$$\langle \left\{ \begin{array}{l} X = A_5^f,\ \underline{Z = h(A_4^\ell)}, \\ A_1^f = f(A_2^\ell, U, Z), \\ h(A_3^{c\ell}) = A_2^\ell \end{array} \right\}; \left\{ \begin{array}{l} \{A_2^\ell, A_4^\ell\}, \\ \{A_1^f, A_5^f\}, \\ \{A_1^f\}, \{A_5^f\} \end{array} \right\} \rangle \stackrel{8}{\rightarrow} \langle \left\{ \begin{array}{l} X = A_5^f,\ Z = h(A_4^\ell), \\ \underline{A_1^f = f(A_2^\ell, U, h(A_4^\ell))}, \\ \underline{h(A_3^{c\ell}) = A_2^\ell} \end{array} \right\}; \left\{ \begin{array}{l} \{A_2^\ell, A_4^\ell\}, \\ \{A_1^f, A_5^f\}, \\ \{A_1^f\}, \{A_5^f\} \end{array} \right\} \rangle$$

E.g., we can now continue in either of the following two ways:

$$\stackrel{11}{\rightarrow} \langle \left\{ \begin{array}{l} X = A_5^f,\ Z = h(A_4^\ell),\ U = A_6^f, \\ \underline{A_1^f = f(A_2^\ell, A_6^f, h(A_4^\ell))},\ h(A_3^{c\ell}) = A_2^\ell \end{array} \right\}; \left\{ \begin{array}{l} \{A_2^\ell, A_4^\ell\}, \{A_1^f\}, \\ \{A_1^f, A_5^f\}, \{A_5^f\}, \{A_6^f\} \end{array} \right\} \rangle$$

$$\stackrel{12}{\rightarrow} \langle \left\{ \begin{array}{l} X = A_5,\ Z = h(A_4^\ell),\ U = A_6^f, \\ \underline{h(A_3^{c\ell}) = A_2^\ell} \end{array} \right\}; \left\{ \begin{array}{l} \{A_2^\ell, A_4^\ell, A_5\}, \{A_2^\ell, A_4^\ell\}, \\ \{A_5, A_6^f\}, \{A_5\}, \{A_6^f\} \end{array} \right\} \rangle$$

$$\stackrel{5}{\rightarrow}\stackrel{12}{\rightarrow} \langle \{\ X = A_5,\ Z = h(A_4^{c\ell}),\ U = A_6^f\ \}; \{\ \{A_5, A_6^f\}, \{A_5\}, \{A_6^f\}\ \} \rangle$$

or

$$\stackrel{11}{\rightarrow} \langle \left\{ \begin{array}{l} X = A_5^f,\ Z = h(A_4^\ell),\ U = A_6^f, \\ A_1^f = f(A_2^\ell, A_6^f, h(A_4^\ell)),\ \underline{h(A_3^{c\ell}) = A_2^\ell} \end{array} \right\}; \left\{ \begin{array}{l} \{A_2^\ell, A_4^\ell\}, \{A_1^f\}, \\ \{A_1^f, A_5^f\}, \{A_5^f\}, \{A_6^f\} \end{array} \right\} \rangle$$

$$\stackrel{5}{\rightarrow}\stackrel{10}{\rightarrow} \langle \left\{ \begin{array}{l} X = A_5^f,\ Z = h(A_4^\ell),\ U = A_6^f, \\ A_1^f = f(h(A_3^{c\ell}), A_6^f, h(A_4^\ell)),\ \underline{A_2^\ell = h(A_3^{c\ell})} \end{array} \right\}; \left\{ \begin{array}{l} \{A_2^\ell, A_4^\ell\}, \{A_1^f\}, \\ \{A_1^f, A_5^f\}, \{A_5^f\}, \{A_6^f\} \end{array} \right\} \rangle$$

$$\stackrel{12}{\rightarrow} \langle \left\{ \begin{array}{l} X = A_5^f,\ Z = h(A_4^{c\ell}),\ U = A_6^f, \\ \underline{A_1^f = f(h(A_3^{c\ell}), A_6^f, h(A_4^{c\ell}))} \end{array} \right\}; \left\{ \begin{array}{l} \{A_1^f, A_5^f\}, \{A_1^f\}, \\ \{A_5^f\}, \{A_6^f\} \end{array} \right\} \rangle$$

$$\stackrel{12}{\rightarrow} \langle \{\ X = A_5^\ell,\ Z = h(A_4^{c\ell}),\ U = A_6^f\ \}; \{\ \{A_5^\ell, A_6^f\}, \{A_5^\ell\}, \{A_6^f\}\ \} \rangle$$

In both cases, it is derived that A_4 is definitely ground and that A_6 is definitely free. The difference is in the annotation of the abstract variable A_5. The order in which the equations are processed thus affects the accuracy of the result. A heuristic rule for achieving optimal precision might be to propagate groundness information first, or, put in another way, delay the application of rule 12 until no other rules are applicable. If rule 12 is applicable to several equations, a suitable order still has to be determined.

The solved form resulting from the above algorithm does not always have the most concise representation possible. For instance, $\mathsf{AEqs}_1 = \langle \{X = A^f, Y = h(A^f, c)\}; \{\{A^f\}\} \rangle$ and $\mathsf{AEqs}_2 = \langle \{Y = h(X, c)\}; \{\} \rangle$ are two *equivalent abstract equation systems* in solved form. A variant of rule 3, which preserves equivalence of abstract equation systems, can be used to further compress an abstract equation system in solved form.[8]

13. $\langle X = A :: \mathcal{E}; \Delta \rangle \stackrel{\text{remove}}{\rightarrow} \langle \mathcal{E}[A \leftarrow X]; \Delta \backslash \{\{A\}\} \rangle$ if $free_{\mathsf{AEqs}}(A)$ & $\neg(\exists \hat{A} \in \mathsf{AVar}(\mathcal{E}).\ A \neq \hat{A}$ & $share_{\mathsf{AEqs}}(A, \hat{A}))$.

If the rule is applied to AEqs_1, we obtain AEqs_2. The abstract unification algorithm may also introduce redundancies in the sense formalized in Section 2.

Example 3.2 *Consider the abstract equation system* $\mathsf{AEqs} = \langle \{X = f(A_3^f, A_4^f, A_2^f), A_1^f = f(A_2^f)\}; \{\{A_1^f\}, \{A_2^f\}, \{A_1^f, A_2^f, A_3^f\}, \{A_3^f, A_4^f\}, \{A_4^f\}\} \rangle$*, which is in reduced form (i.e.,* $\mathsf{AEqs} = \mathsf{Reduce}(\mathsf{AEqs})$*). The abstract unification algorithm applies rule 12 to the second equation and results in the system* $\mathsf{AEqs}' = \langle \{X = f(A_3^f, A_4^f, A_2^f)\}; \{\{A_2^f\}, \{A_3^f, A_4^f\}, \{A_4^f\}\} \rangle$*, which is not in reduced form. An application of the* Reduce *function results in the equivalent more compact system* $\mathsf{AEqs}'' = \langle \{X = f(A_3^f, A_3^f, A_2^f)\}; \{\{A_2^f\}, \{A_3^f\}\} \rangle$.

[8] The rule could replace rule 3, provided that the rules 11 and 12 are combined into one single rule, since otherwise the algorithm becomes non-terminating. Note that for an abstract equation system in solved form, it always holds that $X \notin \mathsf{Var}(\mathcal{E})$.

By solving an equation system imprecision may be introduced, but on the other hand non-unifiable concrete equation systems described by the original abstract system are no longer represented.

Example 3.3 *Consider the following derivation for* $\mathsf{AEqs} = \langle \mathcal{E}; \Delta \rangle$ *and* $V = \{X, Y\}$*:*

$$\mathsf{AEqs} = \langle \left\{ \begin{array}{l} A_1^f = f(A_2^c, A_3^f),\ \underline{A_6^f = f(A_3^f)}, \\ X = g(A_5^f, A_6^f),\ Y = A_6^f \end{array} \right\} ; \left\{ \begin{array}{l} \{A_3^f, A_6^f\}, \{A_6^f\}, \{A_2^c, A_3^f\}, \\ \{A_1^f\}, \{A_1^f, A_5^f\}, \{A_5^f\} \end{array} \right\} \rangle$$

$$\stackrel{10}{\rightarrow} \langle \left\{ \begin{array}{l} A_1^f = f(A_2^c, A_3^f),\ \underline{A_6^f = f(A_3^f)}, \\ X = g(A_5^f, f(A_3^f)),\ Y = f(A_3^f) \end{array} \right\} ; \left\{ \begin{array}{l} \{A_3^f, A_6^f\}, \{A_6^f\}, \{A_2^c, A_3^f\}, \\ \{A_1^f\}, \{A_1^f, A_5^f\}, \{A_5^f\} \end{array} \right\} \rangle$$

$$\stackrel{12}{\rightarrow} \langle \left\{ \begin{array}{l} \underline{A_1^f = f(A_2^c, A_3^f)}, \\ \underline{X = g(A_5^f, f(A_3^f))},\ Y = f(A_3^f) \end{array} \right\} ; \left\{ \begin{array}{l} \{A_2^c, A_3^f\}, \\ \{A_1^f\}, \{A_1^f, A_5^f\}, \{A_5^f\} \end{array} \right\} \rangle$$

$$\stackrel{12}{\rightarrow} \langle \{\ X = g(A_5, f(A_3^f)),\ Y = f(A_3^f)\ \} ; \{\ \{A_3^f\}, \{A_3^f, A_5\}, \{A_5\}\ \} \rangle = \mathsf{AEqs}'$$

For the final abstract equation system, $\mathsf{AEqs}' = \langle \mathcal{E}'; \Delta' \rangle$*, which is in solved form,* $\mu' : A_3^f \mapsto Z_1, A_5 \mapsto g(Z_1, Z_1)$ *is a* Δ'*-preserving abstract term replacement such that* AEqs' *describes* $\mathsf{Eqs}' = \{X = g(g(Z_1, Z_1), f(Z_1)), Y = f(Z_1)\}$*. There does not exist an equivalent (with respect to* V*) concrete equation system described by* AEqs*, since a* Δ*-preserving abstract term replacement either maps* A_5 *to a free variable independent of all other variables in the system, or to a free variable that shares with the free variable that* A_1 *is mapped to. So, the most general unifier of a (unifiable) concrete equation system* Eqs *described by* AEqs *assigns to* X *a term with principal functor* $g/2$*, the first argument of which is either a free variable or a term with principal functor* $f/2$*. Thus,* Eqs *and* Eqs' *are not equivalent (with respect to* V*). A loss of (disjunctive) structural information by the last application of rule 12 is the cause. On the other hand,* $\mu : A_1^f \mapsto Z_1, A_2^c \mapsto g(a), A_3^f \mapsto Z_2, A_5^f \mapsto Z_1, A_6^f \mapsto Z_2$ *is a* Δ*-preserving abstract term replacement such that* AEqs *describes* $\mathsf{Eqs} = \{Z_1 = f(g(a), Z_2), Z_2 = f(Z_2), X = g(Z_1, Z_2), Y = Z_2\}$*. This is an unsatisfiable equation system, hence there does not exist an equivalent equation system described by* AEqs'*.*

4 Soundness of the Abstract Unification Algorithm

Let AEqs_V be an abstract equation system describing the concrete unifiable equation system Eqs (i.e., $\mathsf{AEqs}_V \propto \mathsf{Eqs}$). To prove the soundness of the abstract unification algorithm, we have to show that, independent of the order in which the abstract equations are processed, the abstract solved form obtained for AEqs_V describes a concrete system Eqs' that is equivalent to Eqs on the set V of the program variables of interest. We consider the unification algorithm of Martelli-Montanari [29] extended with a rule for shuffling subterms and a rule for introducing fresh variables. The important property is that every rule preserves the unifiers restricted to the set of variables V. This is sufficient for the analysis to be sound since two concrete equation systems that are equivalent with respect to a set of variables V, have most general unifiers with identical freeness, structural, linearity, and sharing properties for the elements of V.

Algorithm 2 (Concrete unification) *Nondeterministically choose from the set of equations an equation of a form below and perform the associated action.*

1. $X = X :: \mathsf{Eqs} \stackrel{\text{remove}}{\rightarrow} \mathsf{Eqs}$.
2. $f(t_1, \ldots, t_n) = X :: \mathsf{Eqs} \stackrel{\text{switch}}{\rightarrow} X = f(t_1, \ldots, t_n) :: \mathsf{Eqs}$.

3. $f(t_1,\ldots,t_n) = g(s_1,\ldots,s_m) :: \mathsf{Eqs} \overset{\text{peel}}{\rightarrow} \begin{cases} \{t_i = s_i\}_{i=1}^n \cup \mathsf{Eqs} & \text{if } f/n \doteq g/m, \\ \textit{fail} & \textit{otherwise.} \end{cases}$

4. $X = t :: \mathsf{Eqs} \overset{\text{subst}}{\rightarrow} \begin{cases} X = t :: \mathsf{Eqs}[X \leftarrow t] & \text{if } X \notin \mathsf{Var}(t) \ \&\ X \in \mathsf{Var}(\mathsf{Eqs}), \\ \textit{fail} & \text{if } X \in \mathsf{Var}(t) \ \&\ t \neq X. \end{cases}$

5. $s = t :: e(s) :: \mathsf{Eqs} \overset{\text{shuffle}}{\rightarrow} s = t :: e(t) :: \mathsf{Eqs}$ *where* $s, t \notin \mathsf{Var}$ *and* $e(s)$ *is an equation with* s *as a subterm in the lhs or rhs.*

6. $e :: \mathsf{Eqs} \overset{\text{fresh \& subst}}{\rightarrow} \{X = Z\} \cup (e :: \mathsf{Eqs})[X \leftarrow Z]$ *where* $X \in \mathsf{Var}(e)$, Z *is a fresh variable (i.e.,* $Z \notin \mathsf{Var}(e :: \mathsf{Eqs}) \cup V$*).*

Rule 5 is added in order to have an equivalence (with respect to V) preserving concrete transformation rule that corresponds (possibly after repeated applications) to the abstract rule 10. For the purpose of reaching a solved form, the rule may seem counterproductive since term t can be more general (less instantiated) then term s; however no information is lost. Rule 5 can be seen as a generalization of the *Merge* rule of the unification algorithm presented in [16] or the *Confrontation* rule of the unification algorithm for rational trees presented in [10]. Rule 6 is added in order to have an equivalence (with respect to V) preserving concrete transformation rule that corresponds (possibly after repeated applications) to the abstract rule 11. The extended algorithm is not terminating (since rules 5 and 6 can be applied infinitely many times) nor confluent; however, the solved forms obtained after finitely many applications of rules 5 and 6 are all equivalent modulo $\approx_V$. Moreover, since Algorithm 2 is only introduced as a basis for correctness, termination is not essential, given that the abstract Algorithm 1 has been proven to terminate.

In the sequel unless stated otherwise, we denote by $\theta = \mathsf{mgu}(\mathsf{Eqs})$ a most general unifier derived from one of the solved forms of Eqs obtained by the above concrete unification algorithm, involving only finitely many applications of rules 5 and 6. Such a most general unifier is idempotent but not necessarily relevant [1, 2] since it may introduce new variables (no elements of V however). In the proof below we will nevertheless rely on the basic property that for any two unifiable terms, there exist an idempotent and relevant most general unifier (see e.g. [1]) derivable from the solved form that is obtained with zero applications of rules 5 and 6. This most general unifier is unique up to variable renaming.

Theorem 2 (Soundness of abstract unification) *Let* $\mathsf{AEqs}_V = \langle \mathcal{E}; \Delta \rangle$ *be an abstract equation system describing the concrete unifiable equation system* Eqs *(i.e.,* $\mathsf{mgu}(\mathsf{Eqs}) \neq \textit{fail}$*), then for every abstract equation system* AEqs'_V *obtained from* AEqs_V *by applying some rules of Algorithm 1, there exists* Eqs' *derivable from* Eqs *using the rules of Algorithm 2 such that* $\mathsf{AEqs}'_V \propto \mathsf{Eqs}'$ *and* $\mathsf{Eqs}' \approx_V \mathsf{Eqs}$.

The theorem is proven by showing that each abstract transformation rule of Algorithm 1 correctly mimics (one or more) equivalence preserving transformation steps on corresponding concrete equations. In the sequel, the most general instance of two terms t and s is denoted by $\mathsf{mgi}(t, s)$. The most general unifier is denoted by $\mathsf{mgu}(t, s)$. We call a substitution θ *linear on a set* V *of variables* if and only if for all $Z_1, Z_2 \in V$, $Z_1\theta$ and $Z_2\theta$ are linear, and if $Z_1 \neq Z_2$, then $Z_1\theta$ and $Z_2\theta$ are independent (i.e., do not share) [2]. The domain of a substitution θ is defined as $\mathsf{Dom}(\theta) = \{x \in \mathsf{Var} \mid x\theta \neq x\}$ and is assumed to be finite. The range is $\mathsf{Range}(\theta) = \{x \in \mathsf{Var}(y\theta) \mid y \in \mathsf{Dom}(\theta)\}$.

We need a collection of properties describing whether independence, freeness, and linearity properties are preserved under the application of the *idempotent and relevant*

most general unifier of two concrete terms. In the case where the two terms being unified are non-linear, obviously no freeness nor linearity properties hold for their most general instance. We just have the following general result which is true for any two terms being unified.

Lemma 4.1 *Let $t[X_1, \ldots, X_n]$ and $s[Y_1, \ldots, Y_m]$ be concrete unifiable terms. It then holds for an idempotent and relevant $\theta = \mathsf{mgu}(t, s)$ that $\mathsf{Range}(\theta) \cup \mathsf{Dom}(\theta) \subseteq \mathsf{Var}(t) \cup \mathsf{Var}(s)$. Or alternatively, $\forall i \in \{1, \ldots, n\}$, $\mathsf{Var}(X_i\theta) \subseteq \mathsf{Var}(t) \cup \mathsf{Var}(s)$ and similarly, $\forall j \in \{1, \ldots, m\}$, $\mathsf{Var}(Y_j\theta) \subseteq \mathsf{Var}(t) \cup \mathsf{Var}(s)$.*

The corollary describes the effect of the unification of two arbitrary terms t and s in the context of a set of terms $t_1, \ldots, t_q$: it specifies whether and where the variables of t and s occur after the most general unifier has been applied.

Corollary 4.2 *If $X_i (1 \leq i \leq n)$ has other occurrences in the concrete terms $t^i_1, \ldots, t^i_{k_i}$ and $Y_j (1 \leq j \leq m)$ has other occurrences in the concrete terms $s^j_1, \ldots, s^j_{\ell_j}$, then after unifying the terms $t[X_1, \ldots, X_n]$ and $s[Y_1, \ldots, Y_m]$, yielding an idempotent and relevant $\theta = \mathsf{mgu}(t, s)$, it holds for each X_i that either X_i has disappeared or $\exists \{i_1, \ldots, i_r\} \subseteq \{1, \ldots, n\}$, and $\exists \{j_1, \ldots, j_q\} \subseteq \{1, \ldots, m\}$ such that $(r \geq 0, q \geq 1)$ and X_i occurs in all of $\mathsf{mgi}(t, s), t^i_1\theta, \ldots, t^i_{k_i}\theta, t^{i_1}_1\theta, \ldots, t^{i_1}_{k_{i_1}}\theta, \ldots, t^{i_r}_1\theta, \ldots, t^{i_r}_{k_{i_r}}\theta$, and $s^{j_1}_1\theta, \ldots, s^{j_1}_{\ell_{j_1}}\theta, \ldots, s^{j_q}_1\theta, \ldots, s^{j_q}_{\ell_{j_q}}\theta$. A similar symmetric property holds for Y_j $(1 \leq j \leq m)$.*

Variables that do not occur in the terms t and s are not affected by the unification.

Corollary 4.3 *If Z occurs in the concrete terms $t^z_1, \ldots, t^z_{k_z}$ and $Z \notin \{X_1, \ldots, X_n\} \cup \{Y_1, \ldots, Y_m\}$, then after unifying the terms $t[X_1, \ldots, X_n]$ and $s[Y_1, \ldots, Y_m]$, yielding an idempotent and relevant $\theta = \mathsf{mgu}(t, s)$, Z occurs in the concrete terms $t^z_1\theta, \ldots, t^z_{k_z}\theta$.*

Next, we consider the case that both t and s are linear and independent (i.e., not sharing). The lemma describes properties of the variables of t and s, after the most general unifier has been applied.

Lemma 4.4 *Let $t[X_1, \ldots, X_n]$ and $s[Y_1, \ldots, Y_m]$ be linear concrete terms such that the terms are unifiable and $\{X_1, \ldots, X_n\} \cap \{Y_1, \ldots, Y_m\} = \emptyset$ (i.e., the terms do not share). It then holds for an idempotent and relevant $\theta = \mathsf{mgu}(t, s)$ that*

- *$\forall i \in \{1, \ldots, n\}$ either $X_i\theta$ is a* linear *subterm $s'[Y_{i_1}, \ldots, Y_{i_r}]$ (where $r \geq 0$) or $X_i\theta = X_i \in \mathsf{Var}(\mathsf{mgi}(t, s))$ & $\exists!$[9]$j \in \{1, \ldots, m\} : Y_j\theta = t'[X_i, X_{i_1}, \ldots, X_{i_s}]$ (where $s \geq 0$) is a linear subterm,*
- *A similar (symmetric) property holds for $Y_j\theta$ for all $j \in \{1, \ldots, m\}$,*

and θ is a linear substitution on both $\mathsf{Var}(t)$ and $\mathsf{Var}(s)$ (thus, $t\theta = s\theta = \mathsf{mgi}(t, s)$ is linear).

The corollary describes properties about the occurrences of the variables of t and s in the terms $t_1, \ldots, t_q$ after the application of $\theta = \mathsf{mgu}(t, s)$.

[9] The quantifier $\exists!$ stands for 'there exists precisely one'.

Corollary 4.5 *If $X_i (1 \leq i \leq n)$ has other occurrences in the concrete terms $t_1^i, \ldots, t_{k_i}^i$ and $Y_j (1 \leq j \leq m)$ has other occurrences in the concrete terms $s_1^j, \ldots, s_{\ell_j}^j$, then after unifying the linear and independent terms $t[X_1, \ldots, X_n]$ and $s[Y_1, \ldots, Y_m]$, yielding an idempotent and relevant $\theta = \mathsf{mgu}(t, s)$, it holds for each X_i that either X_i has disappeared or $\exists! j (1 \leq j \leq m)$ such that X_i occurs in all of $\mathsf{mgi}(t, s), t_1^i\theta, \ldots, t_{k_i}^i\theta, s_1^j\theta, \ldots, s_{\ell_j}^j\theta$. A similar (symmetric) property holds for Y_j $(1 \leq j \leq m)$.*

Similar but weaker properties hold in the case that only one of the independent terms being unified is linear. The properties specified by the next lemma partially overlap with those of Lemma 5 in [23] (which slightly improved Lemma 2.2 in [7]) and Lemma 5.7 in [2].

Lemma 4.6 *Let $t[X_1, \ldots, X_n]$ be a linear concrete term and $s[Y_1, \ldots, Y_m]$ a non-linear concrete term such that the terms are unifiable and $\{X_1, \ldots, X_n\} \cap \{Y_1, \ldots, Y_m\} = \emptyset$ (i.e., the terms do not share). It then holds for an idempotent and relevant $\theta = \mathsf{mgu}(t, s)$ that*

- $\forall j \in \{1, \ldots, m\}$ *either $Y_j\theta$ is a* linear *term $t'[X_{i_1}, \ldots, X_{i_r}]$ $(r \geq 0)$*
  or $Y_j\theta = Y_j \in \mathsf{Var}(\mathsf{mgi}(t, s))$ & $\exists i \in \{1, \ldots, n\}$:
  $X_i\theta = s'[Y_j, Y_{j_1}, \ldots, Y_{j_r}, X_{i_1}, \ldots, X_{i_q}]$ $(r \geq 0, q \geq 0)$,
- $\forall i \in \{1, \ldots, n\}$ *either $X_i\theta$ is a term $s'[Y_{j_1}, \ldots, Y_{j_r}, X_{i_1}, \ldots, X_{i_q}]$ $(r \geq 0, q \geq 0)$*
  or $X_i\theta = X_i \in \mathsf{Var}(\mathsf{mgi}(t, s))$ & $\exists! j \in \{1, \ldots, m\}$:
  $Y_j\theta = t'[X_i, X_{i_1}, \ldots, X_{i_r}]$ is a linear term *$(r \geq 0)$,*

and θ is a linear substitution on $\mathsf{Var}(s)$.

Corollary 4.7 *If $X_i (1 \leq i \leq n)$ has other occurrences in the concrete terms $t_1^i, \ldots, t_{k_i}^i$ and $Y_j (1 \leq j \leq m)$ has other occurrences in the concrete terms $s_1^j, \ldots, s_{\ell_j}^j$, then after unifying the linear term $t[X_1, \ldots, X_n]$ with the independent non-linear term $s[Y_1, \ldots, Y_m]$, yielding an idempotent and relevant $\theta = \mathsf{mgu}(t, s)$, it holds*

- *for each X_i that either X_i has disappeared or $\exists! j (1 \leq j \leq m)$ such that X_i occurs in all of $\mathsf{mgi}(t, s), t_1^i\theta, \ldots, t_{k_i}^i\theta, s_1^j\theta, \ldots, s_{\ell_j}^j\theta$ and in $t_1^{i_1}\theta, \ldots, t_{k_{i_1}}^{i_1}\theta, \ldots, t_1^{i_\ell}\theta, \ldots, t_{k_{i_\ell}}^{i_\ell}\theta$ (with $\ell \geq 0$). The latter follows because there exist zero or more j (say $i_1, \ldots, i_\ell$) such that $X_i \in \mathsf{Var}(X_j\theta)$.*
- *for each Y_j that either Y_j has disappeared or Y_j occurs in all of $\mathsf{mgi}(t, s), s_1^j\theta, \ldots, s_{\ell_j}^j\theta$ and $t_1^{i_1}\theta, \ldots, t_{k_{i_1}}^{i_1}\theta, \ldots, t_1^{i_\ell}\theta, \ldots, t_{k_{i_\ell}}^{i_\ell}\theta$ (with $\ell \geq 1$). The latter follows because there exist one or more i (say $i_1, \ldots, i_\ell$) such that $Y_j \in \mathsf{Var}(X_i\theta)$.*

If a linear term is unified with another *dependent* (linear or non-linear) term, the next linearity property holds for $\theta = \mathsf{mgu}(t, s)$.

Lemma 4.8 *Let $t[X_1, \ldots, X_n]$ be a linear concrete term and $s[Y_1, \ldots, Y_m]$ an arbitrary concrete term such that the terms are unifiable and $\{X_1, \ldots, X_n\} \cap \{Y_1, \ldots, Y_m\} \neq \emptyset$ (i.e., the terms* do *share). It then holds for an idempotent and relevant $\theta = \mathsf{mgu}(t, s)$ that*

- $\forall j \in \{1, \ldots, m\}$ *such that $Y_j \notin \{X_1, \ldots, X_n\}$*
  either $Y_j\theta$ is a linear *term $t'[X_{i_1}, \ldots, X_{i_r}]$ $(r \geq 0)$*
  or $Y_j\theta = Y_j \in \mathsf{Var}(\mathsf{mgi}(t, s))$ & $\exists i \in \{1, \ldots, n\}$:
  $X_i\theta = s'[Y_j, Y_{j_1}, \ldots, Y_{j_q}, X_{i_1}, \ldots, X_{i_r}]$ $(r \geq 0, q \geq 0)$,

- $\forall i \in \{1,\ldots,n\}$ *either* $X_i\theta$ *is a term* $s'[Y_{j_1},\ldots,Y_{j_q},X_{i_1},\ldots,X_{i_r}]$ *(*$r \geq 0, q \geq 0$*)*
  or $X_i\theta = X_i \in \mathsf{Var}(\mathsf{mgi}(t,s))$ & $\exists j \in \{1,\ldots,m\}$:
  $Y_j\theta = t'[X_i,Y_{j_1},\ldots,Y_{j_q},X_{i_1},\ldots,X_{i_r}]$ *(*$r \geq 0, q \geq 0$*),*

and θ *is a linear substitution on* $\mathsf{Var}(s) \setminus \mathsf{Var}(t)$.

Here, no information that is more constrained than in the general case (described by Corollary 4.2) can be stated about the occurrences of the variables of t and s in the terms $t_1,\ldots,t_q$ after application of $\theta = \mathsf{mgu}(t,s)$.

If one of the terms being unified is a free variable, essentially one new binding results.

Lemma 4.9 *Let* X *be a free variable and* $t[X_1,\ldots,X_n]$ *a concrete term that is unifiable with* X. *It then holds for an idempotent and relevant* $\theta = \mathsf{mgu}(X,t[X_1,\ldots,X_n])$ *that either* $X_i\theta = X_i$ *for all* $i \in \{1,\ldots,n\}$ *and* $X\theta = t[X_1,\ldots,X_n]$ *or* $X\theta = X$ *and* $\neg compound(t[X_1,\ldots,X_n])$, *i.e.,* $n \doteq 1$ & $t[X_1,\ldots,X_n] \doteq X_1$, *and* $X_1\theta = X$.

Corollary 4.10 *If* X *has other occurrences in the concrete terms* $t_1,\ldots,t_k$ *and* $X_j (1 \leq j \leq n)$ *has other occurrences in the concrete terms* $s^j_1,\ldots,s^j_{\ell_j}$, *then after unifying* X *with the term* $t[X_1,\ldots,X_n]$, *yielding an idempotent and relevant* $\theta = \mathsf{mgu}(X,t[X_1,\ldots,X_n])$, *it holds*

- *for* X *that either* X *has disappeared or* $\neg compound(t[X_1,\ldots,X_n])$, $n \doteq 1$ & $t[X_1,\ldots,X_n] \doteq X_1$, *and* X *occurs in all of* $\mathsf{mgi}(t,s), t_1\theta,\ldots,t_k\theta$ *and* $s^1_1\theta,\ldots,s^1_{\ell_1}\theta$.
- *for each* X_j *that either* X_j *has disappeared or* X_j *occurs in all of* $\mathsf{mgi}(t,s), t_1\theta,\ldots,t_k\theta$ *and* $s^j_1\theta,\ldots,s^j_{\ell_j}\theta$.

To summarize, the next lemma specifies sufficient conditions for the freeness and linearity properties of a term r to be preserved under application of the most general unifier of two concrete terms t and s.

Lemma 4.11 *Let* t *and* s *be unifiable terms,* $\theta = \mathsf{mgu}(t,s)$ *idempotent and relevant, and* r *an arbitrary term (*r *may be a subterm of* t *or* s*). It holds that*

1. if $\mathsf{Var}(r) \cap (\mathsf{Var}(t) \cup \mathsf{Var}(s)) = \emptyset$ then $r\theta = r$,
2. if $free(t)$ & $free(s)$ then $free(r) \rightarrow free(r\theta)$,
3. if $free(t)$ & $\mathsf{Var}(t) \cap \mathsf{Var}(r) = \emptyset$ then $free(r) \rightarrow free(r\theta)$ and $linear(r) \rightarrow linear(r\theta)$,
4. if $linear(t)$ & $\mathsf{Var}(t) \cap \mathsf{Var}(r) = \emptyset$ then $linear(r) \rightarrow linear(r\theta)$,
5. if $free(t)$ & $linear(s)$ & $free(r)$ then $linear(r) \rightarrow linear(r\theta)$.

Proof
Case 1. The only free variables that can get bound are those involved in the unification $t = s$ (see Lemma 4.1). Since r shares with neither of them, applying θ leaves r unchanged.
Case 2. When unifying two free variables, no free variables get bound to a non-free term (see Lemma 4.9).
Case 3. From Lemma 4.9, it follows that either s does not get instantiated by $\theta = \mathsf{mgu}(t,s)$ (so $r\theta = r$) or s is a free variable too and gets bound to the free variable t. Because r only possibly shares with s, it follows that $r\theta$ is free resp. linear whenever r is.

Case 4. Because r only possibly shares with s, it follows from Lemma 4.4 in the case that s is linear and independent from t, from Lemma 4.6 in the case that s is non-linear and independent from t, and from Lemma 4.8 in the case that s is dependent from t, that $r\theta$ is linear whenever r is.
Case 5. If s is also a free variable, then Case 5. follows from Case 2. If s is compound, then the free variable r shares with at most one of t and s, since the unification succeeds by assumption. If it shares with one side, the other side is linear, hence it follows similar as in Case 4. that $r\theta$ is linear whenever r is. (Moreover, since r is a free variable, it is also linear.)

□

Proof of Theorem 2.
Let $\mathsf{AEqs}_V = \langle e :: \mathcal{E}; \Delta \rangle$, with e the abstract equation selected for applying one of the rules of Algorithm 1, and $\mathsf{AEqs}'_V = \langle \mathcal{E}'; \Delta' \rangle$ the resulting abstract equation system. Let μ be a Δ-preserving abstract term replacement for AEqs_V such that $\mathsf{AEqs}_V \propto_\mu e\mu :: \mathcal{E}\mu$ and $e\mu :: \mathcal{E}\mu$ is unifiable. It is sufficient to show that there exists a Δ'-preserving abstract term replacement μ' for AEqs'_V such that $\mathsf{AEqs}'_V \propto_{\mu'} \mathcal{E}'\mu'$, where $\mathcal{E}'\mu' \approx_V e\mu :: \mathcal{E}\mu$.

For the rules 1 through 10, take $\mu' = \mu \restriction \mathsf{AVar}(\mathsf{AEqs}'_V)$. For rule 11, take $\mu' = \mu\,\{A'_1 \leftarrow Z_1, \ldots, A'_n \leftarrow Z_n\}$, where $Z_1, \ldots, Z_n$ are fresh variables. By construction, μ' clearly is a Δ'-preserving abstract term replacement for AEqs'_V. For each of these abstract rules a corresponding (finite sequence of) equivalence preserving concrete transition(s), transforming $e\mu :: \mathcal{E}\mu$ into $\mathcal{E}'\mu'$ is easily determined; for the rules 1 through 9, rules 5 and 6 need not be applied; rule 10 corresponds to a (repeated) application of rule 5; rule 11 corresponds to a (repeated) application of rule 6. Hence, it follows that $\mathcal{E}'\mu' \approx_V e\mu :: \mathcal{E}\mu$.

We only present a detailed proof of the nontrivial general case formalised in rule 12. Let $e = (A = \tau[A_1, \ldots, A_n])$ be such that rule 12 applies and take an idempotent and relevant $\theta = \mathsf{mgu}(\{e\mu\})$. We show that $\mu' = (\mu\theta) \restriction \mathsf{AVar}(\mathsf{AEqs}'_V)$ is a Δ'-preserving abstract term replacement for AEqs'_V such that $(\mathcal{E}'\mu' =)\mathcal{E}\mu' \approx_V e\mu :: \mathcal{E}\mu$. We have to prove that

1. $\forall \hat{A} \in \mathsf{AVar}(\mathsf{AEqs}'_V).\ \mathsf{Var}(\hat{A}\mu') \cap V = \emptyset$;
2. $\forall \hat{A} \in \mathsf{AVar}(\mathsf{AEqs}'_V).\ ground_{\mathsf{AEqs}'}(\hat{A}) \Rightarrow ground(\hat{A}\mu')$;
3. $\forall \hat{A} \in \mathsf{AVar}(\mathsf{AEqs}'_V).\ compound_{\mathsf{AEqs}'}(\hat{A}) \Rightarrow compound(\hat{A}\mu')$;
4. $\forall \hat{A} \in \mathsf{AVar}(\mathsf{AEqs}'_V).\ free_{\mathsf{AEqs}'}(\hat{A}) \Rightarrow free(\hat{A}\mu')$;
5. $\forall \hat{A} \in \mathsf{AVar}(\mathsf{AEqs}'_V).\ linear_{\mathsf{AEqs}'}(\hat{A}) \Rightarrow linear(\hat{A}\mu')$;
6. $\forall X \in \mathsf{Range}(\mu').\ SG^{\mu'}_{\mathsf{AEqs}'}(X) \in \Delta'$.

By assumption, $\forall \hat{A} \in \mathsf{AVar}(\mathsf{AEqs}_V).\ \mathsf{Var}(\hat{A}\mu) \cap V = \emptyset$, and $\theta = \mathsf{mgu}(A\mu, \tau[A_1, \ldots, A_n]\mu)$ is idempotent and relevant. So, $\mathsf{Var}(\theta) \cap V = \emptyset$ and 1. follows from the construction of μ'.

The validity of 2. will follow immediately from the correctness of 6.

We give an indirect proof of 3. Assume $compound_{\mathsf{AEqs}'}(\hat{A})$ but $free(\hat{A}\mu')$. We then know that $free(\hat{A}\mu)$ by the construction of μ', and $\neg compound_{\mathsf{AEqs}}(\hat{A})$ since μ is Δ-preserving. Next, we distinguish two cases in correspondence with the algorithm: (a) $\hat{A} = A_i$ for some $i \in \{1, \ldots, n\}$; (b) $\hat{A} \notin \{A, A_1, \ldots, A_n\}$. In case (a), by an inspection of the algorithm, we see that the only situation where c is added to the annotation is if $compound_{\mathsf{AEqs}}(A)$ & $\neg compound_{\mathsf{AEqs}}(\tau[A_1, \ldots, A_n])$. This means that $\tau[A_1, \ldots, A_n] \doteq \hat{A}$ and $c \notin ann_{\mathsf{AEqs}}(\hat{A})$. So $\hat{A}\mu$, which is a free variable by assumption, is unified with the term

$A\mu$ which is not free (because μ is Δ-preserving). Hence, $\hat{A}\mu\theta = \hat{A}\mu'$ will be compound, contrary to our assumption. In case (b), by an inspection of the algorithm, the only situation that c is added to the annotation is when either $free_{\mathsf{AEqs}}(A)$ & $compound_{\mathsf{AEqs}}(\tau[A_1, \ldots, A_n])$& $(\Delta(\{\hat{A}\}) \subseteq \Delta(\{A\}) \vee \Delta(\{A\}) \subseteq \Delta(\{\hat{A}\}))$ or $compound_{\mathsf{AEqs}}(A)$& $free_{\mathsf{AEqs}}(\tau[A_1, \ldots, A_n])$ & $(\Delta(\{\hat{A}\}) \subseteq \Delta(\{A_1, \ldots, A_n\}) \vee \Delta(\{A_1, \ldots, A_n\}) \subseteq \Delta(\{\hat{A}\}))$. In the first case, since μ is Δ-preserving, it follows that the free variable $\hat{A}\mu$ shares with the free variable $A\mu$ which is unified with a compound term; hence, $\hat{A}\mu\theta = \hat{A}\mu'$ will be compound, contrary to our assumption. The second case is proved similarly.

To prove 4., we distinguish for $\hat{A} \in \mathsf{AVar}(\Delta')$ two cases in correspondence with the algorithm: (a) $\hat{A} \doteq A_i$ for some $i \in \{1, \ldots, n\}$; (b) $\hat{A} \notin \{A, A_1, \ldots, A_n\}$. In both cases it is clear from the algorithm that $free_{\mathsf{AEqs}'}(\hat{A})$ only if $free_{\mathsf{AEqs}}(\hat{A})$, and hence $free(\hat{A}\mu)$ because μ is Δ-preserving. We must prove that $free(\hat{A}\mu\theta)$.

In case (a), it is clear from the algorithm that $free_{\mathsf{AEqs}'}(\hat{A})$ only if $free_{\mathsf{AEqs}}(A)$, hence $free(A\mu)$ and by Lemma 4.9 also $free(\hat{A}\mu\theta)$.

In case (b), by inspection of Algorithm 1, one of the following cases must hold:

(1) $\neg share_{\mathsf{AEqs}}(\tau[A_1, \ldots, A_n], \hat{A})$ & $\neg share_{\mathsf{AEqs}}(A, \hat{A})$. Since μ is Δ-preserving, we know that $(\mathsf{Var}(\tau[A_1, \ldots, A_n]\mu) \cup \mathsf{Var}(A\mu)) \cap \mathsf{Var}(\hat{A}\mu) = \emptyset$, so by Lemma 4.11 case 1., we have $\hat{A}\mu\theta = \hat{A}\mu$ and thus $free(\hat{A}\mu) \rightarrow free(\hat{A}\mu\theta)$.

(2) $free_{\mathsf{AEqs}}(\tau[A_1, \ldots, A_n])$ & $free_{\mathsf{AEqs}}(A)$ & $free_{\mathsf{AEqs}}(\hat{A})$. Since we know that $free(\tau[A_1, \ldots, A_n]\mu)$ & $free(A\mu)$, it follows from Lemma 4.11 case 2. that $free(\hat{A}\mu) \rightarrow free(\hat{A}\mu\theta)$.

(3) $free_{\mathsf{AEqs}}(A)$& $\neg share_{\mathsf{AEqs}}(A, \hat{A})$. Since we know that $free(A\mu)$& $\mathsf{Var}(A\mu) \cap \mathsf{Var}(\hat{A}\mu) = \emptyset$, it follows from Lemma 4.11 case 3. that $free(\hat{A}\mu) \rightarrow free(\hat{A}\mu\theta)$.

(4) $free_{\mathsf{AEqs}}(\tau[A_1, \ldots, A_n])$ & $\neg share_{\mathsf{AEqs}}(\tau[A_1, \ldots, A_n], \hat{A})$. Symmetric to case (3).

In all cases we derive that $free(\hat{A}\mu) \rightarrow free(\hat{A}\mu\theta)$. Moreover, $\hat{A}\mu$ is free, thus so is $\hat{A}\mu\theta$.

The validity of 5. for $\hat{A} \in \mathsf{AVar}(\mathcal{E}) \setminus \mathsf{AVar}(\Delta')$, follows from the correctness of 2. To prove 5. for $\hat{A} \in \mathsf{AVar}(\Delta')$, first note that we only have to consider the case where $\neg free_{\mathsf{AEqs}'}(\hat{A})$ because if $free_{\mathsf{AEqs}'}(\hat{A})$ then $linear(\hat{A}\mu')$ follows from $free(\hat{A}\mu')$ proved above. As for 4., we distinguish for $\hat{A} \in \mathsf{AVar}(\Delta')$ two cases in correspondence with the algorithm: (a) $\hat{A} \doteq A_i$ for some $i \in \{1, \ldots, n\}$; (b) $\hat{A} \notin \{A, A_1, \ldots, A_n\}$. In both cases it is clear from the algorithm that $linear_{\mathsf{AEqs}'}(\hat{A})$ only if $linear_{\mathsf{AEqs}}(\hat{A})$, and hence $linear(\hat{A}\mu)$ because μ is Δ-preserving. We must prove that $linear(\hat{A}\mu\theta)$.

In case (a), it is clear from the algorithm (assuming $\neg free_{\mathsf{AEqs}'}(\hat{A})$) that $linear_{\mathsf{AEqs}}(A)$& $(\neg share_{\mathsf{AEqs}}(\hat{A}, A) \vee free_{\mathsf{AEqs}}(\tau[A_1, \ldots, A_n]))$, hence $linear(A\mu)$ and $A\mu$ and $\hat{A}\mu$ either do not share or they share but $\tau[A_1, \ldots, A_n]\mu$ is a free variable. In the first case, it follows from Lemma 4.11 case 4. and $linear(\hat{A}\mu)$ that $linear(\hat{A}\mu\theta)$. In the latter case, it follows that also $A\mu$ is a free variable since otherwise the unification fails due to the occur-check. From Lemma 4.11 case 2., it then follows that $linear(\hat{A}\mu\theta)$.

In case (b), by inspection of Algorithm 1 (again assuming $\neg free_{\mathsf{AEqs}'}(\hat{A})$), one of the following cases must hold:

(1) $\neg share_{\mathsf{AEqs}}(\tau[A_1, \ldots, A_n], \hat{A})$ & $\neg share_{\mathsf{AEqs}}(A, \hat{A})$. Since μ is Δ-preserving, we know that $\mathsf{Var}(\hat{A}\mu) \cap (\mathsf{Var}(\tau[A_1, \ldots, A_n]\mu) \cup \mathsf{Var}(A\mu)) = \emptyset$, so by Lemma 4.11 case 1., we have $\hat{A}\mu\theta = \hat{A}\mu$ and thus $linear(\hat{A}\mu) \rightarrow linear(\hat{A}\mu\theta)$.

(3) $free_{\mathsf{AEqs}}(A) \& \neg share_{\mathsf{AEqs}}(A, \hat{A})$. Since we know that $free(A\mu) \& \mathsf{Var}(A\mu) \cap \mathsf{Var}(\hat{A}\mu) = \emptyset$, it follows from Lemma 4.11 case 3. that $linear(\hat{A}\mu) \rightarrow linear(\hat{A}\mu\theta)$.

(4) $free_{\mathsf{AEqs}}(\tau[A_1, \ldots, A_n]) \ \& \ \neg share_{\mathsf{AEqs}}(\tau[A_1, \ldots, A_n], \hat{A})$. Similar to case (3).

(5) $free_{\mathsf{AEqs}}(A) \ \& \ linear_{\mathsf{AEqs}}(\tau[A_1, \ldots, A_n]) \ \& \ free_{\mathsf{AEqs}}(\hat{A})$. Since we know that $free(A\mu) \ \& \ linear(\tau[A_1, \ldots, A_n]\mu) \ \& \ free(\hat{A}\mu)$, it follows from Lemma 4.11 case 5. that $linear(\hat{A}\mu) \rightarrow linear(\hat{A}\mu\theta)$.

(6) $free_{\mathsf{AEqs}}(\tau[A_1, \ldots, A_n]) \ \& \ linear_{\mathsf{AEqs}}(A) \ \& \ free_{\mathsf{AEqs}}(\hat{A})$. Similar to case (5).

(7) $linear_{\mathsf{AEqs}}(A) \ \& \ \neg share_{\mathsf{AEqs}}(A, \hat{A})$. Since we know that $linear(A\mu) \ \& \ \mathsf{Var}(A\mu) \cap \mathsf{Var}(\hat{A}\mu) = \emptyset$, it follows from Lemma 4.11 case 4. that $linear(\hat{A}\mu) \rightarrow linear(\hat{A}\mu\theta)$.

(8) $linear_{\mathsf{AEqs}}(\tau[A_1, \ldots, A_n]) \ \& \ \neg share_{\mathsf{AEqs}}(\tau[A_1, \ldots, A_n], \hat{A})$. Similar to case (7).

In all cases we derive that $linear(\hat{A}\mu) \rightarrow linear(\hat{A}\mu\theta)$. Moreover, $\hat{A}\mu$ is linear, thus so is $\hat{A}\mu\theta$. Note that there is no case corresponding to (2) in the algorithm since we assume $\neg free_{\mathsf{AEqs}'}(\hat{A})$ and hence, if $free_{\mathsf{AEqs}}(\tau[A_1, \ldots, A_n]) \& free_{\mathsf{AEqs}}(A)$, it must be that $\neg free_{\mathsf{AEqs}}(\hat{A})$, since otherwise the algorithm would yield $free_{\mathsf{AEqs}'}(\hat{A})$, a contradiction.

To prove 6., we distinguish two cases (similar as in [21]): (a) $X \in \mathsf{Range}^{10}(\mu') \setminus \mathsf{Var}(A\mu' = \tau[A_1, \ldots, A_n]\mu')$; (b) $X \in \mathsf{Var}(A\mu' = \tau[A_1, \ldots, A_n]\mu')$. Recall that $SG^{\mu}_{\mathsf{AEqs}}(X) = \{A \in \mathsf{AVar}(\mathsf{AEqs}_V) \mid X \in \mathsf{Var}(A\mu)\}$, and $SG^{\mu'}_{\mathsf{AEqs}'}(X) = \{A \in \mathsf{AVar}(\mathsf{AEqs}'_V) \mid X \in \mathsf{Var}(A\mu')\}$.

In case (a), it follows from Lemma 4.1 and the construction of μ' that $X \in \mathsf{Range}(\mu) \setminus \mathsf{Var}(A\mu = \tau[A_1, \ldots, A_n]\mu)$ and hence (by Corollary 4.3) that $SG^{\mu'}_{\mathsf{AEqs}'}(X) = SG^{\mu}_{\mathsf{AEqs}}(X) \in \bar{\Delta}(\{A, A_1, \ldots, A_n\})$. By inspection of Algorithm 1 we have in any case $\bar{\Delta}(\{A, A_1, \ldots, A_n\}) \subseteq \Delta'$.

In case (b), let $t_1 = A\mu$, $t_2 = \tau[A_1, \ldots, A_n]\mu$, and $X \in \mathsf{Var}(t_1\theta)$ $(= \mathsf{Var}(t_2\theta))$.

- Suppose that t_1 is free and t_2 compound (or symmetrically, t_2 is free but t_1 is compound). First suppose that $X \doteq t_1$. Then it follows from Corollary 4.10 that X disappears, i.e., $X \notin \mathsf{Var}(t_1\theta)$, which is contrary to the assumption. So, only $X \in \mathsf{Var}(t_2)$ is possible. Let $Y \doteq t_1$, then $Y \notin \mathsf{Var}(t_2)$ (since otherwise the unification fails due to the occur-check) and in particular $X \neq Y$. It follows from Corollary 4.10 that $SG^{\mu'}_{\mathsf{AEqs}'}(X) = (SG^{\mu}_{\mathsf{AEqs}}(X) \cup SG^{\mu}_{\mathsf{AEqs}}(Y)) \setminus (\{A, A_1, \ldots, A_n\} \setminus \mathsf{AVar}(\mathcal{E}))$.

  Because μ is a Δ-preserving abstract term replacement, $SG^{\mu}_{\mathsf{AEqs}}(Y) \in \Delta(\{A\})$, but $SG^{\mu}_{\mathsf{AEqs}}(Y) \notin \Delta(\{A_1, \ldots, A_n\})$ since $Y \notin \mathsf{Var}(\tau[A_1, \ldots, A_n]\mu)$. On the other hand, $SG^{\mu}_{\mathsf{AEqs}}(X) \in \Delta(\{A_1, \ldots, A_n\})$, but $SG^{\mu}_{\mathsf{AEqs}}(X) \notin \Delta(\{A\})$ since $A\mu \neq X$. This is why it is safe, in the case that $free_{\mathsf{AEqs}}(A) \ \& \ compound_{\mathsf{AEqs}}(\tau[A_1, \ldots, A_n])$ or $compound_{\mathsf{AEqs}}(A) \ \& \ free_{\mathsf{AEqs}}(\tau[A_1, \ldots, A_n])$, to first reduce the sharing component Δ with the set $\Delta(\{A\}) \cap \Delta(\{A_1, \ldots, A_n\})$. By inspection of Algorithm 1, it follows that in any case $SG^{\mu'}_{\mathsf{AEqs}'}(X) \in \Delta'$, but only in the cases where the abstract domain does derive *definite compoundness* information for abstract variables, the algorithm can avoid to include elements $S_\tau \cup S_A$ in Δ' where S_τ or S_A are in $\Delta(\{A_1, \ldots, A_n\}) \cap \Delta(\{A\})$.

[10] Recall, $\mathsf{Range}(\mu') = \cup_{A \in \mathsf{Dom}(\mu')} \mathsf{Var}(A\mu')$.

- Suppose that t_1 and t_2 are both free. Suppose that $X \doteq t_1$. Since $X \in \mathsf{Var}(t_1\theta)$, it follows from Corollary 4.10 that $\exists Y : Y \doteq t_2 \ \& \ Y\theta = X$ and $SG^{\mu'}_{\mathsf{AEqs}'}(X) = (SG^{\mu}_{\mathsf{AEqs}}(X) \cup SG^{\mu}_{\mathsf{AEqs}}(Y)) \setminus (\{A, A_1, \ldots, A_n\} \setminus \mathsf{AVar}(\mathcal{E}))$. On the other hand, suppose $X \doteq t_2$. Then it follows again from Corollary 4.10 that $SG^{\mu'}_{\mathsf{AEqs}'}(X) = (SG^{\mu}_{\mathsf{AEqs}}(X) \cup SG^{\mu}_{\mathsf{AEqs}}(Y)) \setminus (\{A, A_1, \ldots, A_n\} \setminus \mathsf{AVar}(\mathcal{E}))$, where $Y \doteq t_1$.

  Because in the first case $SG^{\mu}_{\mathsf{AEqs}}(X) \in \Delta(\{A\})$ and $SG^{\mu}_{\mathsf{AEqs}}(Y) \in \Delta(\{A_1, \ldots, A_n\})$, and vice versa in the second case $SG^{\mu}_{\mathsf{AEqs}}(X) \in \Delta(\{A_1, \ldots, A_n\})$ and $SG^{\mu}_{\mathsf{AEqs}}(Y) \in \Delta(\{A\})$, it follows by inspection of Algorithm 1, that $SG^{\mu'}_{\mathsf{AEqs}'}(X) \in \Delta'$.

- Suppose that both t_1, t_2 are linear (but not free) and do not share, and suppose (without loss of generality) that $X \in \mathsf{Var}(t_1)$, then it follows from Corollary 4.5 that $\exists! Y \in \mathsf{Var}(t_2) : SG^{\mu'}_{\mathsf{AEqs}'}(X) = (SG^{\mu}_{\mathsf{AEqs}}(X) \cup SG^{\mu}_{\mathsf{AEqs}}(Y)) \setminus (\{A, A_1, \ldots, A_n\} \setminus \mathsf{AVar}(\mathcal{E}))$. As in the previous case, $SG^{\mu'}_{\mathsf{AEqs}'}(X) \in \Delta'$ holds.

- Suppose that t_1, t_2 do not share, t_1 is linear (and non-free) but t_2 is not. Because μ is a Δ-preserving abstract term replacement and $\neg linear(\tau[A_1, \ldots, A_n]\mu)$, we know that $\neg linear_{\mathsf{AEqs}}(\tau[A_1, \ldots, A_n])$.

  First suppose $X \in \mathsf{Var}(t_1)$. It follows from Corollary 4.7 that $\exists! Y \in \mathsf{Var}(t_2) \ \& \ \exists \{X_{i_1}, \ldots, X_{i_\ell}\} \subseteq \mathsf{Var}(t_1) : SG^{\mu'}_{\mathsf{AEqs}'}(X) = (SG^{\mu}_{\mathsf{AEqs}}(X) \cup SG^{\mu}_{\mathsf{AEqs}}(Y) \cup_{i=i_1}^{i_\ell} SG^{\mu}_{\mathsf{AEqs}}(X_i)) \setminus (\{A, A_1, \ldots, A_n\} \setminus \mathsf{AVar}(\mathcal{E}))$, where $\ell \geq 0$. Because $\{SG^{\mu}_{\mathsf{AEqs}}(X), SG^{\mu}_{\mathsf{AEqs}}(X_i) \mid i = i_1, \ldots, i_\ell\} \subseteq \Delta(\{A\})$ and $SG^{\mu}_{\mathsf{AEqs}}(Y) \in \Delta(\{A_1, \ldots, A_n\})$, it follows from $\neg linear_{\mathsf{AEqs}}(\tau[A_1, \ldots, A_n])$ and an inspection of Algorithm 1 that $SG^{\mu'}_{\mathsf{AEqs}'}(X) \in \Delta'$.

  On the other hand suppose $X \in \mathsf{Var}(t_2)$. It follows from Corollary 4.7 that $\exists \{X_{i_1}, \ldots, X_{i_\ell}\} \subseteq \mathsf{Var}(t_1) : SG^{\mu'}_{\mathsf{AEqs}'}(X) = (SG^{\mu}_{\mathsf{AEqs}}(X) \cup_{i=i_1}^{i_\ell} SG^{\mu}_{\mathsf{AEqs}}(X_i)) \setminus (\{A, A_1, \ldots, A_n\} \setminus \mathsf{AVar}(\mathcal{E}))$, where $\ell \geq 1$. Because $\{SG^{\mu}_{\mathsf{AEqs}}(X_i) \mid i = i_1, \ldots, i_\ell\} \subseteq \Delta(\{A\})$ and $SG^{\mu}_{\mathsf{AEqs}}(X) \in \Delta(\{A_1, \ldots, A_n\})$, it follows again from $\neg linear_{\mathsf{AEqs}}(\tau[A_1, \ldots, A_n])$ and an inspection of Algorithm 1 that $SG^{\mu'}_{\mathsf{AEqs}'}(X) \in \Delta'$.

- The symmetric case, namely t_1, t_2 not sharing, t_2 linear but t_1 not, is proved analogously.

- Now consider the most general case: t_1, t_2 both non-linear and/or sharing. Because μ is a Δ-preserving abstract term replacement and $\neg linear(\tau[A_1, \ldots, A_n]\mu) \ \& \ \neg linear(A\mu)$ or $\tau[A_1, \ldots, A_n]\mu$ and $A\mu$ share, we know $(\neg linear_{\mathsf{AEqs}}(\tau[A_1, \ldots, A_n]) \ \& \ \neg linear_{\mathsf{AEqs}}(A)) \vee share_{\mathsf{AEqs}}(\tau[A_1, \ldots, A_n], A)$.

  Because $X \in \mathsf{Var}(t_1\theta)$ $(= \mathsf{Var}(t_2\theta))$, it follows from Corollary 4.2 that $S_1 = \{Z \in \mathsf{Var}(t_1) \mid X \in \mathsf{Var}(Z\theta)\} \neq \emptyset$, and $S_2 = \{Z \in \mathsf{Var}(t_2) \mid X \in \mathsf{Var}(Z\theta)\} \neq \emptyset$ and

$$SG^{\mu'}_{\mathsf{AEqs}'}(X) = \bigcup_{Z \in S_1 \cup S_2} SG^{\mu}_{\mathsf{AEqs}}(Z).$$

  Clearly, for all $Z \in S_1$, $SG^{\mu}_{\mathsf{AEqs}}(Z) \in \Delta(\{A\})$, for all $Z \in S_2$, $SG^{\mu}_{\mathsf{AEqs}}(Z) \in \Delta(\{A_1, \ldots, A_n\})$, so by inspection of Algorithm 1, $SG^{\mu'}_{\mathsf{AEqs}'}(X) \in \Delta'$.

□

5 Discussion and Related Work

Finding a highly accurate abstract unification algorithm in general is a non-trivial problem and presupposes a thorough study of the freeness and linearity properties of the composition of concrete substitutions. In Section 4, an attempt is made to formulate certain of these properties. Similar but less general lemma's concerning linearity properties of most general unifiers have been given in [2, 7, 23], but as far as we know, no full study has appeared in the literature.

The present algorithm incorporates the optimisation suggested in [30], namely in the definition of the sharing component Δ' in rule 12, if one of A or $\tau[A_1, \ldots, A_n]$ is definitely free then no *closure under union* operation is needed. The algorithm further performs a few optimisations that take into account that concrete unifications may fail due to the occur-check.

Example 5.1 *Consider the abstract equation* $A = f(A_1, A_2, A_3)$ *in the context of a sharing component* $\Delta = \{\{A_1, A^f\}, \{\hat{A}^f, A_2, A^f\}, \{\hat{A}^f, A_3, A^f\}, \{A^f\}\}$. *Since* $\Delta(\{\hat{A}\}) \subseteq \Delta(\{A\}) \cap \Delta(\{A_1, A_2, A_3\})$ *and* $free_{\mathsf{AEqs}}(\hat{A})$, *any* Δ*-preserving abstract term replacement assigns a free variable to* $\hat{A}$ *that necessarily also occurs in the terms assigned to* A *and* $f(A_1, A_2, A_3)$. *Because also* $free_{\mathsf{AEqs}}(A)$ *holds, any concrete unification described by the abstract equation fails due to occur-check. Thus, it is safe for the abstract algorithm to derive* $\bot$. *On the other hand, suppose* $\hat{A}$ *is not definitely free; because* $\Delta(\{A_1, \ldots, A_n\}) \subseteq \Delta(\{A\})$, *the only unifiable concrete equations described result in that case from* Δ*-preserving abstract term replacements that assign ground terms to* $\hat{A}, A_1, A_2,$ *and* A_3. *This is used to improve the sharing information derived.*

However, in most Prolog implementations, the occur-check is omitted from the unification algorithm for efficiency reasons. Syntactic conditions for occur-check free programs are given e.g. in [2], and it is argued that most well-known Prolog programs meet these conditions. For such programs the optimisations incorporated in Algorithm 1 may increase the accuracy, but an experimental evaluation is needed to really assess their practical value. On the other hand, for programs that construct infinite terms (rational trees [10]), the present work is not readily applicable since we prove correctness of the abstract unification algorithm with respect to a concrete unification algorithm that incorporates the occur-check. It is outside the scope of the present paper to address the impact of the possible creation of infinite terms on the abstract unification algorithm and soundness proof; nevertheless, the issue has practical importance for compiler optimisations and should be considered in future work.

Several other abstract domains for variable sharing and freeness information have been proposed [11, 23, 24, 30, 32] for use in abstract interpretation based analyses of logic programs and their properties, precision and efficiency, have been studied extensively. Muthukumar and Hermenegildo [30] presented a sharing and freeness analysis in which the set-sharing domain is combined with a freeness component associating program variables with the annotations F (free), G (ground), or NF (possibly non-free). The abstract domain, $\wp(\wp(\mathsf{PVar})) \times \wp(\mathsf{PVar} \rightarrow \{G, F, NF\})$, was designed to represent groundness, freeness, and potential variable sharing among the program variables PVar, but neither linearity nor structural information. As an illustration, consider the example of [30]:

goal	$\mathrm{pred}(f(X_1,X_2),f(X_3,X_4),X_3,X_5,X_6,X_6)$
sharing	$\{\{X_2\},\{X_3\},\{X_5\},\{X_6\},\{X_1,X_2\}\}$
freeness	$\{X_1/F, X_2/NF, X_3/F, X_4/G, X_5/F, X_6/F\}$

In the formalism of the present approach, we have:

$\mathcal{E};\Delta$	$\{X_1 = A_1, X_2 = A_2, X_4 = A_4\};\qquad \{\{A_2\},\{A_1,A_2\}\}$
ann_{AEqs}	$A_1 \mapsto \{f,\ell\}, A_2 \mapsto \emptyset, A_4 \mapsto \{c,\ell\}$

Note that because $V = \{X_1, X_2, X_3, X_4, X_5, X_6\}$ (i.e., the set of variables of interest), the definite freeness of X_3, X_5, and X_6 is in fact implicit. Also, the particular names of the abstract variables are irrelevant; any consistent renaming will do as well.

If this goal is unified with the clause head $\mathrm{pred}(Y_1, Y_1, a, Y_2, Y_2, f(Y_1, Y_3))$, the resulting abstract substitution has sharing component $\{\{Y_2, Y_3\}\}$ and freeness component $\{Y_1/G$, $Y_2/NF, Y_3/F\}$. According to the algorithm of the present paper, we obtain:

$\mathcal{E}';\Delta'$	$\{Y_1 = f(a,A_4), Y_2 = f(f(a,A_4),Y_3)\};\qquad \{\}$
$ann_{\mathsf{AEqs}'}$	$A_4 \mapsto \{c,\ell\}$

or if depth-bound 1 is imposed on mixed terms:

$\mathcal{E}'';\Delta''$	$\{Y_1 = f(a,A_4), Y_2 = f(A_5,Y_3)\};\qquad \{\}$
$ann_{\mathsf{AEqs}''}$	$A_4 \mapsto \{c,\ell\}, A_5 \mapsto \{c,\ell\}$

Hence, we also derive that Y_1 is ground, Y_3 is free, and Y_2 and Y_3 possibly share; in fact the latter sharing between Y_2 and Y_3 is derived to be definite sharing, and both Y_1 and Y_2 are known to have principal functor $f/2$. Note that no fixed depth bound need be imposed, since it can be adjusted dynamically through upperbound or widening operations, similar as in the generic pattern domain $\mathtt{Pat}(\mathcal{R})$ of [26, 27, 33].

Indeed, the composite domain has some more features in common with the $\mathtt{Pat}(\mathcal{R})$ domain. The abstract equations component is in fact an alternative representation of the pattern component of $\mathtt{Pat}(\mathcal{R})$ which provides both structural and same-value information but uses integer indices to identify subterms instead of abstract variables (e.g., indices $1,\ldots,6$ for the equations of $\mathcal{E}''$ where 6 subterms can be distinguished). The sharing component and annotation mapping can be seen as a particular instance of the generic part $\mathcal{R}$ for associating information with the so-called *leaves* (subterms for which the principal functor is not known), and could as well be replaced by some other abstract domain, for instance, Søndergaard's domain for pair sharing and groundness analysis, extended with a component to provide freeness information. The representation of the abstract equation system AEqs'' above in the $\mathtt{Pat}(\mathcal{R})$ notation could be something like:

same-value	$\{Y_1 \mapsto 1, Y_2 \mapsto 4, Y_3 \mapsto 6\};$
pattern	$\{\,(1, f(2,3)),\ (2, a),\ (4, f(5,6))\,\};$
$\mathcal{R}$	$\langle\ \{\{6\}\};\quad 3 \mapsto \{c,\ell\}, 5 \mapsto \{c,\ell\}, 6 \mapsto \{f,\ell\}\ \rangle$

From this comparison, it appears that the abstract equations provide a quite natural and compact representation for definite equality between subterms and non-disjunctive structural information. Moreover, algorithms for the unification and upperbound operations are easily derived as rather obvious extensions of the well-known algorithms for unification and anti-unification [25] respectively. On the other hand, the generic part $\mathcal{R}$ of the $\mathtt{Pat}(\mathcal{R})$ domain can be used to associate information with *non-leaves* too. A possible

application of this feature might be *structure sharing information*. In principle, this could be realized in the present composite domain, but not with the definition of solved form and the transition system as presented here.

King [23] presented an enhancement of the basic set-sharing domain that tracks linearity of terms besides variable sharing and groundness information. The embedding of linearity into sharing groups appears very attractive since it distinguishes more accurately which variables occur multiply and thereby yields a more accurate sharing analysis than the present proposal. Freeness and compoundness are not represented in the approach. In [24], King and Soper present a depth-k sharing and freeness analysis based on the set-sharing domain that encodes structural properties of substitutions in the sharing groups. Groundness, possible sharing and freeness properties are represented, but no linearity nor principal functor information. The structural properties of substitutions captured by this domain only concern potential depth-k path occurrences in the runtime values of program variables. The freeness information derived is basicly 'possible' freeness information; in order to derive 'definite' freeness information as in the abstract equation systems domain, an additional component has to be added to the depth-k sharing and freeness domain for tracing 'possible instantiation' information.

Finally, [6] introduced another domain of abstract equation systems for describing possible variable sharing, definite freeness or compoundness of program variables. The formalism uses meta-variables ($\mathcal{Z}, \mathcal{Z}_i, \ldots$, comparable to definitely free abstract variables) and meta-terms ($\bot[V], \bot^+[V], \bot^\bullet[V]$, comparable to abstract terms (arbitrary, definitely compound, definitely free, resp.) that possibly share with the variables in the set V). An abstract equation may describe different concrete equations to which different concrete rules apply. Consequently, the abstract unification algorithm is non-deterministic; different solved forms may be obtained, all of which must be considered to interpret the definite freeness, compoundness, or possible sharing information derived. Therefore, the approach appears less suited for an efficient implementation. However, due to the careful and exhaustive case analysis performed by the non-deterministic algorithm, the analysis can be highly accurate, if a good choice is made for the order in which the abstract equations are selected and for the representation of the occurrences of definitely free meta-terms (either $\bot^\bullet[\mathcal{Z}]$ or $\mathcal{Z}$). For example, the analysis is more precise than ours in the case of the abstract equation system $\{X = f(Y), \bot[X] = f(Z)\}$, which corresponds in our representation to $\langle\{X = f(Y), X = A_1^f, A_2 = f(Z)\}; \{\{A_1^f\}, \{A_1^f, A_2\}, \{A_2\}\}\rangle$, ($\bot[X]$ represents a term possibly sharing with X but not with Y or Z). The non-deterministic algorithm of [6] is able to derive that Y remains free while Algorithm 1 of the present paper only derives Y to remain definitely linear. On the other hand, the groundness and linearity information expressible in the formalism of [6] is limited and is not exploited to improve the precision of the unification.

Future work entails implementing the composite domain and operations and studying more carefully the precision and efficiency with respect to the analyses discussed above. To this end, some further operations need to be specified. For instance, for the top-down framework of Bruynooghe [4], a safe upperbound, projection, and extension operation must be developed. Appendix A contains preliminary specifications of the operations. In Appendix B, we show an analysis trace of the `append/3` predicate for an initial call specifying that all three arguments are free variables. This gives the reader a good intuitive idea of the kind of operations involved.

6 Conclusion

Abstract equation systems provide an expressive formalism for the combined derivation of groundness, freeness, linearity, and possible sharing information, as well as depth-bounded structural, compoundness, and subterm equality information. Moreover, they are suited to specify a high level unification algorithm as an equation solving transition system, the correctness of which has been justified with respect to a concrete equation solving algorithm. The soundness proof of the abstract linearity and freeness information derived by the algorithm is based on a study of the invariance of certain linearity and freeness properties under the application of concrete idempotent and relevant most general unifiers.

References

[1] K. R. Apt. Logic programming. In J. Van Leeuwen, editor, *Handbook of Theoretical Computer Science, Volume B: Formal Models and Semantics*, pages 493–574. Elsevier, 1990.

[2] K. R. Apt. Why the occur-check is not a problem. In M. Bruynooghe and M. Wirsing, editors, *Proceedings of the Fourth International Symposium on Programming Language Implementation and Logic Programming, PLILP'92*, pages 69–86, Leuven, Belgium, 1992.

[3] R. Barbuti, R. Giacobazzi, and G. Levi. A general framework for semantics-based bottom-up abstract interpretation of logic programs. *ACM Trans. Prog. Lang. Syst.*, 15(1):133–181, Jan. 1993.

[4] M. Bruynooghe. A practical framework for the abstract interpretation of logic programs. *The Journal of Logic Programming*, 10(2):91–124, 1991.

[5] M. Bruynooghe and M. Codish. Freeness, sharing, linearity and correctness – all at once. In P. Cousot, M. Falaschi, G. Filè, and A. Rauzy, editors, *Proceedings of the Third International Workshop on Static Analysis*, LNCS 724, pages 153–164, Padova, Italy, Sept. 1993. Springer-Verlag.

[6] M. Codish, D. Dams, G. Filé, and M. Bruynooghe. Freeness analysis for logic programs – and correctness? In D. S. Warren, editor, *Proceedings of the Tenth International Conference on Logic Programming*, pages 116–131, Budapest, Hungary, 1993. The MIT Press. An extended version is available as Technical Report, Dept. of Comp. Science, K.U.Leuven.

[7] M. Codish, D. Dams, and E. Yardeni. Derivation and safety of an abstract unification algorithm for groundness and aliasing analysis. In K. Furukawa, editor, *Proceedings of the Eighth International Conference on Logic Programming*, pages 79–93, Paris, France, 1991. The MIT Press, Cambridge, Mass.

[8] M. Codish, D. Dams, and E. Yardeni. Bottom-up abstract interpretation of logic programs. *Theoretical Computer Science*, 124(1):93–126, Feb. 1994.

[9] M. Codish, A. Mulkers, M. Bruynooghe, M. García de la Banda, and M. Hermenegildo. Improving abstract interpretations by combining domains. In *Proceedings of the ACM Symposium on Partial Evaluation and Semantics-based Program Transformation*, pages 194–205. ACM Press, 1993.

[10] A. Colmerauer. Prolog and infinite trees. In K. L. Clark and S.-A. Tärnlund, editors, *Logic Programming*, pages 231–251. Academic Press, 1982.

[11] 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. D. Jones, editors, *Proceedings of PEPM'91, the ACM SIGPLAN Symposium on Partial Evaluation and Semantics-Based Program Manipulation*, pages 52–61, 1991.

[12] A. Cortesi, B. Le Charlier, and P. Van Hentenryck. Combinations of abstract domains for logic programming. In *Conference Record of POPL'94: 21st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 227–239, Portland, Oregon, 1994. ACM Press.

[13] P. Cousot and R. Cousot. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Proceedings of the Fourth ACM Symposium on Principles of Programming Languages*, pages 238–252, Los Angeles, 1977.

[14] P. Cousot and R. Cousot. Systematic design of program analysis frameworks. In *Conference Record of the Sixth Annual ACM Symposium on Principles of Programming Languages*, pages 269–282, San Antonio, Texas, Jan. 1979.

[15] P. Cousot and R. Cousot. Abstract interpretation and application to logic programs. *Journal of Logic Programming*, 13(2&3):103–179, July 1992.

[16] N. Dershowitz and J.-P. Jouannaud. Rewrite systems. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B: Formal Models and Semantics, pages 243–320. Elsevier Science Publishers, 1990.

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

[18] G. Huet. *Resolution d'Equations Dans Des Langages D'Ordre* $1, 2, \ldots, \omega$. Université de Paris VII (thèse d'état), 1976.

[19] G. Huet. Confluent reductions: Abstract properties and applications to term rewriting systems. *Journal of the Association for Computing Machinery*, 27(4):797–821, 1980.

[20] D. Jacobs and A. Langen. Accurate and Efficient Approximation of Variable Aliasing in Logic Programs. In E. L. Lusk and R. A. Overbeek, editors, *Proceedings of the North American Conference on Logic Programming*, pages 154–165, Cleveland, Ohio, USA, 1989.

[21] D. Jacobs and A. Langen. Static analysis of logic programs for independent AND-parallelism. *The Journal of Logic Programming*, 13(2 and 3):291–314, 1992.

[22] N. D. Jones and H. Søndergaard. A semantic-based framework for the abstract interpretation of Prolog. In S. Abramsky and C. Hankin, editors, *Abstract Interpretation of Declarative Languages*, Ellis Horwood Series in Computers and their Applications, pages 123–142. Ellis Horwood, Chichester, 1987.

[23] A. King. A synergistic analysis for sharing and groundness which traces linearity. In D. Sannella, editor, *Proceedings of Fifth European Symposium on Programming Languages and Systems – ESOP'94*, pages 363–378, Edinburgh, 1994. Springer-Verlag.

[24] A. King and P. Soper. Depth-k sharing and freeness. In P. Van Hentenryck, editor, *Proceedings of the 11th International Conference on Logic Programming*, pages 553–568, Santa Margherita Ligure, Italy, 1994.

[25] J.-L. Lassez, M. J. Maher, and K. Marriott. Unification revisited. In J. Minker, editor, *Foundations of Deductive Databases and Logic Programming*, pages 587–625. Morgan Kaufmann Publishers Inc., Los Altos, 1988.

[26] B. Le Charlier and P. Van Hentenryck. Reexecution in abstract interpretation of Prolog. In K. Apt, editor, *Proceedings of the Joint International Conference and Symposium on Logic Programming*, pages 750–764, Washington D.C., 1992. MIT Press, Cambridge.

[27] B. Le Charlier and P. Van Hentenryck. Experimental evaluation of a generic abstract interpretation algorithm for Prolog. *ACM Trans. Prog. Lang. Syst.*, 16(1):35–101, Jan. 1994.

[28] K. Marriott. Frameworks for abstract interpretation. *Acta Informatica*, 30(2):103–129, 1993.

[29] A. Martelli and U. Montanari. An efficient unification algorithm. *ACM Transactions on Programming Languages and Systems*, 4(2):258–282, Apr. 1982.

[30] K. Muthukumar and M. Hermenegildo. Combined determination of sharing and freeness of program variables through abstract interpretation. In K. Furukawa, editor, *Proceedings of the Eighth International Conference on Logic Programming*, pages 49–63, Paris, France, 1991. The MIT Press.

[31] H. Søndergaard. An application of abstract interpretation of logic programs: Occur check reduction. In B. Robinet and R. Wilhelm, editors, *ESOP'86 Proceedings European Symposium on Programming*, Lecture Notes in Computer Science 213, pages 327–338. Springer-Verlag, N.Y., 1986.

[32] R. Sundararajan and J. S. Conery. An abstract interpretation scheme for groundness, freeness, and sharing analysis of logic programs. In *Twelfth FST & TCS Conference*, New Delhi, India, Dec. 1992.

[33] P. Van Hentenryck, A. Cortesi, and B. Le Charlier. Type analysis of Prolog using type graphs. Technical Report CS-93-52, Brown University, Department of Computer Science, Nov. 1993.

Appendix

A Domain Dependent Operations: Pragmatic Definitions

The domain dependent operations required by the framework of abstract interpretation [4] are projection, extension, upperbound, and preorder relation. For the domain at hand, the operations have to take into account that the variable sharing information is spread across both the set of equations and the explicit sharing component; moreover, the upperbound should retain as much as possible the structural and subterm equality information.

Consider the following two abstract equation systems in solved form (with set of variables $V = \{X, Y, Z, U, W\}$).

AEqs^1_V	$\{Y = f(a, X, X, Z), U = f(W)\}; \{\}$
AEqs^2_V	$\{Y = f(h(X, A^f_2), g(X), X, A_3), W = f(U)\}; \{\{A^f_2, A_3\}\}$

By analogy with concrete equation systems in solved form, the concrete variables occurring in an abstract equation system in solved form can be classified into *parameters* and *eliminable* variables: the first are the variables occurring in the terms of the rhs of the equations, the latter are the variables of the lhs of the equations [25]. For convenience of the exposition given below, we also define the variables of V not occurring in the equations to be parameters. For AEqs^1_V, the eliminable variables are Y, U, the parameters X, Z, W; for AEqs^2_V, the eliminable variables are Y, W, the parameters X, U, Z.

In order to facilitate pragmatic[11] definitions for the preorder relation and upperbound, we introduce the notion of *solution solved form*: An abstract equation system in solved form, AEqs_V, is said to be in *solution solved form* if it contains for each variable $X \in V$ exactly one equation of the form $X = \tau[A_1, \ldots, A_n]$ and no other equations, i.e., it has an empty set of parameters and $V = \mathsf{Var}(\mathsf{AEqs})$. The solution solved form can be derived from the solved form by applying an equivalence preserving transformation rule that is in some sense the inverse of rule 13.:

14. $\langle \mathcal{E}; \Delta \rangle \overset{\text{fresh \& subst}}{\rightarrow} \langle \{X_1 = A'_1, \ldots, X_n = A'_n\} \cup \mathcal{E}[\{X_1 \leftarrow A'_1, \ldots, X_n \leftarrow A'_n\}]; \Delta \cup \{\{A'_1\}, \ldots, \{A'_n\}\}\rangle$
with $ann_{\mathsf{AEqs}'}(\hat{A}) = \begin{cases} \{f, \ell\}, & \text{if } \hat{A} \doteq A'_i, \text{ for } 1 \leq i \leq n, \\ ann_{\mathsf{AEqs}}(\hat{A}), & \text{otherwise,} \end{cases}$

where $\{X_1, \ldots, X_n\} \neq \emptyset$ is the set of parameters of $\mathcal{E}$ and $A'_1, \ldots, A'_n$ are new abstract variables not occurring in the system.

The (solution) solved form of $\bot$ is defined to be just $\bot$ again. For AEqs^1_V and AEqs^2_V, we obtain the solution solved forms

$\mathsf{AEqs}^{1'}_V$	$\{X = A^f_1, Y = f(a, A^f_1, A^f_1, A^f_2), Z = A^f_2, U = f(A^f_3), W = A^f_3\}$; $\{\{A^f_1\}, \{A^f_2\}, \{A^f_3\}\}$
$\mathsf{AEqs}^{2'}_V$	$\{X = A^f_1, Y = f(h(A^f_1, A^f_2), g(A^f_1), A^f_1, A_3), Z = A^f_5, U = A^f_4, W = f(A^f_4)\}$; $\{\{A^f_1\}, \{A^f_2, A_3\}, \{A^f_4\}, \{A^f_5\}\}$

Let AEqs be an abstract equation system and $\mathsf{ATerm}(\mathsf{AEqs})$ the set of abstract terms $\tau[A_1, \ldots, A_n]$ such that $\{A_1, \ldots, A_n\} \subseteq \mathsf{AVar}(\mathsf{AEqs})$. For the elements of $\mathsf{ATerm}(\mathsf{AEqs})$ the predicates defined in Section 2 can be given an alternative more pragmatic definition.

$ground_{\mathsf{AEqs}}(\tau[A_1, \ldots, A_n]) \Leftrightarrow \forall i \in \{1, \ldots, n\} : ground_{\mathsf{AEqs}}(A_i)$;
$compound_{\mathsf{AEqs}}(\tau[A_1, \ldots, A_n]) \Leftrightarrow compound(\tau[A_1, \ldots, A_n]) \vee (\tau[A_1, \ldots, A_n]) \doteq A_1 \ \&\ compound_{\mathsf{AEqs}}(A_1)$;
$linear_{\mathsf{AEqs}}(\tau[A_1, \ldots, A_n]) \Leftrightarrow \forall i \in \{1, \ldots, n\} : linear_{\mathsf{AEqs}}(A_i) \ \&\ \neg(\exists i \in \{1, \ldots, n\} : (A_i \text{ occurs more than once in } \tau[A_1, \ldots, A_n] \ \&\ \neg ground_{\mathsf{AEqs}}(A_i))) \ \&$

[11] By this we mean operations that are expressed in terms of the properties of the abstract variables instead of in terms of Δ-preserving abstract term replacements.

$$\neg(\exists i, j \in \{1,\ldots,n\} : (i \neq j \ \&\ share_{\mathsf{AEqs}}(A_i, A_j)));$$
$free_{\mathsf{AEqs}}(\tau[A_1,\ldots,A_n]) \Leftrightarrow \tau[A_1,\ldots,A_n] \doteq A_1 \ \&\ free_{\mathsf{AEqs}}(A_1);$
$share_{\mathsf{AEqs}}(\tau[A_1,\ldots,A_n], A) \Leftrightarrow \exists i \in \{1,\ldots,n\} : share_{\mathsf{AEqs}}(A_i, A).$

We can also extend the mapping ann_{AEqs} to such abstract terms $\tau[A_1,\ldots,A_n]$.

$$ann_{\mathsf{AEqs}}(\tau[A_1,\ldots,A_n]) = \left\{ \begin{array}{lll} \text{if} & compound_{\mathsf{AEqs}}(\tau[A_1,\ldots,A_n]) & \text{then } ann_{aux} \cup \{c\} \\ & & \text{else } ann_{aux} \\ \text{where} & ann_{aux} = & \\ & \text{if } free_{\mathsf{AEqs}}(\tau[A_1,\ldots,A_n]) & \text{then } \{f,\ell\} \\ & \text{else if } linear_{\mathsf{AEqs}}(\tau[A_1,\ldots,A_n]) & \text{then } \{\ell\} \\ & \text{else} & \emptyset \end{array} \right.$$

Let $\mathsf{AEqs}^1_V = \langle \mathcal{E}_1; \Delta_1 \rangle$ and $\mathsf{AEqs}^2_V = \langle \mathcal{E}_2; \Delta_2 \rangle$ be two abstract equation systems in solution solved form. In order to define the preorder relation, we first define an ordering on the equation components as follows:

$$\mathcal{E}_1 \leq_{eq} \mathcal{E}_2 \Leftrightarrow \exists \sigma : \mathsf{AVar}(\mathsf{AEqs}^2_V) \to \mathsf{ATerm}(\mathsf{AEqs}^1_V).\ \ \mathcal{E}_2\sigma = \mathcal{E}_1.$$

If $\mathcal{E}_1 \leq_{eq} \mathcal{E}_2$, then $\mathcal{E}_1$ is an instance of $\mathcal{E}_2$; thus the structural and term equality information of $\mathcal{E}_2$ is more general then that of $\mathcal{E}_1$ (i.e., $\mathcal{E}_2$ contains less structural or less term equality constraints) and for any abstract term replacement μ for AEqs^1_V, $\sigma\mu$ is an abstract term replacement for AEqs^2_V. Next, we want to express that the information specified by Δ_2 and $ann_{\mathsf{AEqs}^2_V}$ is more general than the information specified by Δ_1 and $ann_{\mathsf{AEqs}^1_V}$. In order to do so, the latter first needs to be translated in terms of the abstract variables of AEqs^2_V. We use σ to translate the possible sharing expressed by AEqs^1_V (either implicitly by the equations or explicitly by the sharing component) and define, for each $A \in \mathsf{AVar}(\mathsf{AEqs}^1_V)$, the *abstract sharing group* w.r.t. the mapping $\sigma : \mathsf{AVar}(\mathsf{AEqs}^2_V) \to \mathsf{ATerm}(\mathsf{AEqs}^1_V)$ such that $\mathcal{E}_2\sigma = \mathcal{E}_1$, as follows:

$$ASG^{\sigma}_{\mathsf{AEqs}^2_V}(A) \stackrel{def}{=} \{A' \in \mathsf{AVar}(\mathsf{AEqs}^2_V) \mid A \in \mathsf{AVar}(A'\sigma)\}.$$

The definition is analogous to the definition of concrete sharing group. It is used in the next alternative definition for the preorder relation previously given in Section 2.

Definition A.1 (Preorder) *Let* $\mathsf{AEqs}^1_V = \langle \mathcal{E}_1; \Delta_1 \rangle$ *and* $\mathsf{AEqs}^2_V = \langle \mathcal{E}_2; \Delta_2 \rangle$ *be two abstract equation systems in solution solved form (different from* $\bot$*).*

$$\mathsf{AEqs}^1_V \leq_{ssf} \mathsf{AEqs}^2_V \Leftrightarrow \ \exists \sigma : \mathsf{AVar}(\mathsf{AEqs}^2_V) \to \mathsf{ATerm}(\mathsf{AEqs}^1_V) \text{ such that } \left\{ \begin{array}{l} \mathcal{E}_2\sigma = \mathcal{E}_1 \ \& \\ \forall A \in \mathsf{AVar}(\mathsf{AEqs}^2_V).\ ann_{\mathsf{AEqs}^2_V}(A) \subseteq ann_{\mathsf{AEqs}^1_V}(A\sigma) \ \& \\ \forall S \in \Delta_1.\ \bigcup_{A \in S} ASG^{\sigma}_{\mathsf{AEqs}^2_V}(A) \in \Delta_2 \end{array} \right.$$

For all abstract equation systems, AEqs_V*, in solution solved form, we define* $\bot \leq_{ssf} \mathsf{AEqs}_V$*.*

Example A.1 *As an illustration, consider again Example 2.2 of Section 2 where* $\mathsf{AEqs}^1_V = \langle \{X = f(Y)\}; \emptyset \rangle$*,* $\mathsf{AEqs}^2_V = \langle \{X = A_1, Y = A_2^{\ell}\}; \{\{A_1, A_2^{\ell}\}\} \rangle$*, and* $V = \{X, Y\}$*. To compare* AEqs^1_V *and* AEqs^2_V*, we first have to transform* AEqs^1_V *into solution solved form,*

$\mathsf{AEqs}_V^{1\,\prime} = \langle\{X = f(A_3^f), Y = A_3^f\}; \{\{A_3^f\}\}\rangle$. *Next, defining* $\sigma : A_1 \mapsto f(A_3^f), A_2^\ell \mapsto A_3^f$, *it is easily verified that the conditions of Definition A.1 are met:*

$$\begin{aligned} \mathcal{E}_2\sigma &= \mathcal{E}_1', \\ ann_{\mathsf{AEqs}_V^2}(A_1) &\subseteq ann_{\mathsf{AEqs}_V^{1\,\prime}}(f(A_3^f)), \\ ann_{\mathsf{AEqs}_V^2}(A_2^\ell) &\subseteq ann_{\mathsf{AEqs}_V^{1\,\prime}}(A_3^f), \end{aligned}$$

and since $\Delta_1' = \{\{A_3^f\}\}$,

$$\begin{aligned} \bigcup_{A \in \{A_3^f\}} ASG^\sigma_{\mathsf{AEqs}_V^2}(A) &= ASG^\sigma_{\mathsf{AEqs}_V^2}(A_3^f), \\ &= \{A' \in \mathsf{AVar}(\mathsf{AEqs}_V^2) \mid A_3^f \in \mathsf{AVar}(A'\sigma)\}, \\ &= \{A_1, A_2^\ell\} \in \Delta_2. \end{aligned}$$

We conclude that $\mathsf{AEqs}_V^{1\,\prime} \leq_{ssf} \mathsf{AEqs}_V^2$.

For $V = \{X, Y\}$, the maximal element in $\mathcal{AE}qs$ w.r.t. $\leq_{ssf}$ is $\mathsf{AEqs}_V^{max} = \langle\{X = A_1, Y = A_2\}; \{\{A_1\}, \{A_1, A_2\}, \{A_2\}\}\rangle$.

Lemma A.1 *Let* AEqs_V^1 *and* AEqs_V^2 *be two abstract equation systems in solution solved form (and different from* $\bot$*). We have*

$$\mathsf{AEqs}_V^1 \leq_{ssf} \mathsf{AEqs}_V^2 \quad \Rightarrow \quad \mathsf{AEqs}_V^1 \leq \mathsf{AEqs}_V^2$$

Proof of $\Rightarrow$. From the definition of the preorder relation $\leq_{ssf}$, we know

$$\mathsf{AEqs}_V^1 \leq_{ssf} \mathsf{AEqs}_V^2 \Leftrightarrow \exists\sigma : \mathsf{AVar}(\mathsf{AEqs}_V^2) \to \mathsf{ATerm}(\mathsf{AEqs}_V^1) \text{ such that } \begin{cases} \mathcal{E}_2\sigma = \mathcal{E}_1 \ \& \\ \forall A \in \mathsf{AVar}(\mathsf{AEqs}_V^2).\ ann_{\mathsf{AEqs}_V^2}(A) \subseteq ann_{\mathsf{AEqs}_V^1}(A\sigma) \ \& \\ \forall S \in \Delta_1.\ \bigcup_{A \in S} ASG^\sigma_{\mathsf{AEqs}_V^2}(A) \in \Delta_2 \end{cases}$$

To prove $\mathsf{AEqs}_V^1 \leq \mathsf{AEqs}_V^2$, it is sufficient to show that for any Δ_1-preserving abstract term replacement $\mu : \mathsf{AVar}(\mathsf{AEqs}_V^1) \to \mathsf{Term}$ for AEqs_V^1 such that $\mathcal{E}_1\mu$ is unifiable, $\sigma\mu : \mathsf{AVar}(\mathsf{AEqs}_V^2) \to \mathsf{Term}$ is a Δ_2-preserving abstract term replacement for AEqs_V^2 such that $\mathcal{E}_1\mu \approx_V \mathcal{E}_2\sigma\mu$. Since, $\mathcal{E}_2\sigma = \mathcal{E}_1$, the latter clearly holds. To prove that $\sigma\mu$ is Δ_2-preserving, we need to show that

1. $\forall \hat{A} \in \mathsf{AVar}(\mathsf{AEqs}_V^2).\ ground_{\mathsf{AEqs}_V^2}(\hat{A}) \Rightarrow ground(\hat{A}\sigma\mu)$;
2. $\forall \hat{A} \in \mathsf{AVar}(\mathsf{AEqs}_V^2).\ compound_{\mathsf{AEqs}_V^2}(\hat{A}) \Rightarrow compound(\hat{A}\sigma\mu)$;
3. $\forall \hat{A} \in \mathsf{AVar}(\mathsf{AEqs}_V^2).\ free_{\mathsf{AEqs}_V^2}(\hat{A}) \Rightarrow free(\hat{A}\sigma\mu)$;
4. $\forall \hat{A} \in \mathsf{AVar}(\mathsf{AEqs}_V^2).\ linear_{\mathsf{AEqs}_V^2}(\hat{A}) \Rightarrow linear(\hat{A}\sigma\mu)$;
5. $\forall X \in \mathsf{Range}(\sigma\mu).\ SG^{\sigma\mu}_{\mathsf{AEqs}_V^2}(X) \in \Delta_2$.

For 1. and 2., we give an indirect proof. Suppose that $ground_{\mathsf{AEqs}_V^2}(\hat{A})$ but not $ground(\hat{A}\sigma\mu)$. From the latter, it follows that $\exists A \in \mathsf{AVar}(\hat{A}\sigma).\ \neg ground(A\mu)$. By the definition of abstract sharing group $\hat{A} \in ASG^\sigma_{\mathsf{AEqs}_V^2}(A)$. Since μ is Δ_1-preserving, $\exists S \in \Delta_1.\ A \in S$, and by the definition of $\leq_{ssf}$, it follows that $\hat{A} \in \mathsf{AVar}(\Delta_2)$, a contradiction with $ground_{\mathsf{AEqs}_V^2}(\hat{A})$.

Suppose that $compound_{\mathsf{AEqs}^2_V}(\hat{A})$ but not $compound(\hat{A}\sigma\mu)$. From the latter, it follows that $free(\hat{A}\sigma\mu)$ and, since μ is Δ_1-preserving, that not $compound_{\mathsf{AEqs}^1_V}(\hat{A}\sigma)$ or $c \notin ann_{\mathsf{AEqs}^1_V}(\hat{A}\sigma)$. By the definition of $\leq_{ssf}$, we then know that $c \notin ann_{\mathsf{AEqs}^2_V}(\hat{A})$, a contradiction with $compound_{\mathsf{AEqs}^2_V}(\hat{A})$. To prove 3. (the prove of 4. is similar), note that because $free_{\mathsf{AEqs}^2_V}(\hat{A})$, we have $\{f,\ell\} = ann_{\mathsf{AEqs}^2_V}(\hat{A}) \subseteq ann_{\mathsf{AEqs}^1_V}(\hat{A}\sigma)$, by the definition of $\leq_{ssf}$. From the definition of $ann_{\mathsf{AEqs}^1_V}$ for abstract terms, it then follows that $free_{\mathsf{AEqs}^1_V}(\hat{A}\sigma)$ and since μ is Δ_1-preserving also $free(\hat{A}\sigma\mu)$ holds. To prove 5., we observe that

$$\begin{aligned} SG^{\sigma\mu}_{\mathsf{AEqs}^2_V}(X) &= \{A \in \mathsf{AVar}(\mathsf{AEqs}^2_V) \mid X \in \mathsf{Var}(A\sigma\mu)\}, \\ &\qquad \text{by the definition of concrete sharing group,} \\ &= \textstyle\bigcup_{A \in SG^{\mu}_{\mathsf{AEqs}^1_V}(X)} ASG^{\sigma}_{\mathsf{AEqs}^2_V}(A), \\ &\qquad \text{by the definition of abstract sharing group.} \end{aligned}$$

Since μ is Δ_1-preserving, we have $SG^{\mu}_{\mathsf{AEqs}^1_V}(X) \in \Delta_1$ and thus $SG^{\sigma\mu}_{\mathsf{AEqs}^2_V}(X) \in \Delta_2$ by the definition of $\leq_{ssf}$. □

The inverse of the above lemma does not hold, because an abstract equation system need not be in reduced form. Consider the systems $\mathsf{AEqs}^1_V = \langle\{X = A^f_1, Y = A^\ell_2, Z = A^f_3\}; \{\{A^f_1, A^\ell_2\}, \{A^\ell_2, A^f_3\}\}\rangle$ and $\mathsf{AEqs}^2_V = \langle\{X = A^f_1, Y = A^c_2, Z = A^f_3\}; \{\{A^f_1, A^c_2\}, \{A^c_2, A^f_3\}\}\rangle$. Then $\mathsf{AEqs}^1_V \leq \mathsf{AEqs}^2_V$ because every Δ_1-preserving abstract term replacement will map A^ℓ_2 to a *compound* linear term containing the two different variables that A^f_1 and A^f_3 are mapped to. However, $\mathsf{AEqs}^1_V \not\leq_{ssf} \mathsf{AEqs}^2_V$ since $ann_{\mathsf{AEqs}^2_V}(A_2) \not\subseteq ann_{\mathsf{AEqs}^1_V}(A_2)$ (σ is the identity mapping). On the other hand, consider a third system $\mathsf{AEqs}^3_V = \langle\{X = A^f_1, Y = A^c_2, Z = A^f_3\}; \{\{A^f_1, A^c_2\}, \{A^f_3\}\}\rangle$ and suppose that the set of functors that can occur in concrete terms is $\mathcal{F} = \{a/0, f/1\}$. Both AEqs^2_V and AEqs^3_V are in reduced form, nevertheless $\mathsf{AEqs}^2_V \leq \mathsf{AEqs}^3_V$, but $\mathsf{AEqs}^2_V \not\leq_{ssf} \mathsf{AEqs}^3_V$. The first holds because AEqs^2_V does not describe any (unifiable) concrete equation systems since there does not exist a concrete compound term with at least two distinct free variables in it. The latter holds because $\{A^c_2, A^f_3\} \notin \Delta_3$ (again σ is the identity mapping).

We only have the following conjectures that refer to the notion of underlying Herbrand universe (i.e., the subset of Term consisting of the *ground* terms only). In [25] the Herbrand universe is called *trivial* if it contains none or only one element, otherwise it is called *non-trivial* [25]. Here, we call the underlying Herbrand universe *trivial* if it contains none or only elements with principal functor of arity smaller or equal to 1. **Conjecture:** If the underlying Herbrand universe is non-trivial, then two abstract equation systems in *reduced* solution solved form are equivalent if and only if they are equal up to abstract variable renaming.

Conjecture 1 *Let* AEqs^1_V *and* AEqs^2_V *be two abstract equation systems in* reduced *solution solved form (and different from* $\bot$*) and let the underlying Herbrand universe be non-trivial. We have*

$$\mathsf{AEqs}^1_V \leq_{ssf} \mathsf{AEqs}^2_V \quad \Leftarrow \quad \mathsf{AEqs}^1_V \leq \mathsf{AEqs}^2_V$$

The (least) upperbound operation is based on the non-deterministic *Anti-unification Algorithm* of [25] (originally presented in [18]). The latter is enhanced with rules to extract the variable sharing, freeness, and linearity information from the least common

anti-instances of the abstract equations. Let ϕ be any injection from $\mathsf{MTerm} \times \mathsf{MTerm}$ to AVar, and define λ recursively on $\mathsf{MTerm} \times \mathsf{MTerm}$ as follows:

$$\begin{aligned} \lambda(f(s_1,\ldots,s_n), f(t_1,\ldots,t_n)) &= f(\lambda(s_1,t_1),\ldots,\lambda(s_n,t_n)), \\ &\quad \text{for every function or constant symbol f,} \\ \lambda(s,t) &= \phi(s,t), \\ &\quad \text{otherwise.} \end{aligned}$$

Different choices of ϕ may result in different terms; however, these will be equivalent up to abstract variable renaming. For the definition below, ϕ must only partially be defined for a subset of $\{(\tau_1,\tau_2) \mid \tau_1 \in \mathsf{ATerm}(\mathsf{AEqs}^1_V)\ \&\ \tau_2 \in \mathsf{ATerm}(\mathsf{AEqs}^2_V)\}$. In order to extract variable sharing information that is due to term equality constraints in the input terms τ_1 and τ_2 and translate it to their least common anti-instance $\lambda(\tau_1,\tau_2)$, where it is expressed as *possible* sharing, we define

$$\forall A \in \mathsf{AVar}(\tau_1). \quad \Pi^1_\phi(A) = \{\phi(s,t) \mid A \in \mathsf{AVar}(s)\ \&\ \phi(s,t) \text{ used to construct } \lambda(\tau_1,\tau_2)\},$$
$$\forall A \in \mathsf{AVar}(\tau_2). \quad \Pi^2_\phi(A) = \{\phi(s,t) \mid A \in \mathsf{AVar}(t)\ \&\ \phi(s,t) \text{ used to construct } \lambda(\tau_1,\tau_2)\}.$$

For example, consider $\tau_1 = g(b, A_1, A_1, h(A_1), A_2)$ and $\tau_2 = g(h(A_1,A_2), f(A_1), A_1, h(A_1), A_3)$. The partial definition of ϕ needed to construct $\lambda(\tau_1,\tau_2)$ is

$$\phi: \quad (b, h(A_1,A_2)) \mapsto A_1, \quad (A_1, f(A_1)) \mapsto A_2, \quad (A_1,A_1) \mapsto A_3, \quad (A_2,A_3) \mapsto A_4.$$

We have $\lambda(\tau_1,\tau_2) = g(A_1,A_2,A_3,h(A_3),A_4)$, $\Pi^1_\phi(A_1) = \{A_2,A_3\}$, $\Pi^1_\phi(A_2) = \{A_4\}$, $\Pi^2_\phi(A_1) = \{A_1,A_2,A_3\}$, $\Pi^2_\phi(A_2) = \{A_1\}$, and $\Pi^2_\phi(A_3) = \{A_4\}$. The sharing component $\Delta = \{\{A_2,A_3\},\{A_4\},\{A_1,A_2,A_3\},\{A_1\}\}$ is a safe upper approximation of the variable sharing due to subterm equality constraints present in τ_1 and τ_2.

Definition A.2 (Least Upperbound) *Let* $\mathsf{AEqs}^1_V = \langle \mathcal{E}_1; \Delta_1 \rangle$ *and* $\mathsf{AEqs}^2_V = \langle \mathcal{E}_2; \Delta_2 \rangle$ *be two abstract equation systems in solution solved form. The least upperbound of* AEqs^1_V *and* AEqs^2_V *is defined as follows:*

$$\mathsf{AEqs}^1_V \sqcup \mathsf{AEqs}^2_V = \langle \lambda(\mathcal{E}_1,\mathcal{E}_2); \Delta \rangle$$

where for the sake of applying λ*, the equations of* $\mathcal{E}_1, \mathcal{E}_2$ *are treated as terms, the variables of the lhs are treated as constants, and the equations are ordered according to the variables of the lhs.*

$$\Delta = \{ \bigcup_{A \in S} \Pi^1_\phi(A) \mid S \in \Delta_1 \} \cup \{ \bigcup_{A \in S} \Pi^2_\phi(A) \mid S \in \Delta_2 \}$$

and, for $A \in \mathsf{AVar}(\lambda(\mathcal{E}_1,\mathcal{E}_2))$ *and* $(\tau_1,\tau_2) = \phi^{-1}(A)$*, define*

$$ann_{\mathsf{AEqs}^1_V \sqcup \mathsf{AEqs}^2_V}(A) = ann_{\mathsf{AEqs}^1_V}(\tau_1) \cap ann_{\mathsf{AEqs}^2_V}(\tau_2).$$

Note that for $A \in \mathsf{AVar}(\mathcal{E}_1)$, $\Pi^1_\phi(A) \neq \emptyset$, and similarly for $A \in \mathsf{AVar}(\mathcal{E}_2)$, $\Pi^2_\phi(A) \neq \emptyset$. Hence, $\emptyset \notin \Delta$.

For example, consider again the abstract equation systems, $\mathsf{AEqs}^{1'}_V$ and $\mathsf{AEqs}^{2'}_V$, which are in solution solved form, and a (partial) definition for the injection ϕ,

$$\begin{array}{llll} (A^f_1, & A^f_1) & \mapsto A^f_1 & (a, \quad h(A^f_1, A^f_2)) \mapsto A^{c\ell}_2 \quad (A^f_1, \quad g(A^f_1)) \mapsto A^\ell_3 \\ (A^f_2, & A_3) & \mapsto A_4 & (A^f_2, \quad A^f_5) \mapsto A^f_5 \quad (f(A^f_3), \quad A^f_4) \mapsto A^\ell_6 \\ (A^f_3, & f(A^f_4)) & \mapsto A^\ell_7 & \end{array}$$

The least upperbound $\mathsf{AEqs}_V = \mathsf{AEqs}_V^{1'} \sqcup \mathsf{AEqs}_V^{2'}$ is given by

AEqs_V	$\{X = A_1^f, Y = f(A_2^{c\ell}, A_3^\ell, A_1^f, A_4), Z = A_5^f, U = A_6^\ell, W = A_7^\ell\}$; $\{\{A_1^f, A_3^\ell\}, \{A_4, A_5^f\}, \{A_6^\ell, A_7^\ell\}, \{A_1^f, A_2^{c\ell}, A_3^\ell\}, \{A_5^f\}, \{A_2^{c\ell}, A_4\}\}$

An application of rule 13. leads to no further reduction in this case, but it might be useful in general.

Lemma A.2 *Let* AEqs_V^1 *and* AEqs_V^2 *be two abstract equation systems in solution solved form. We have*

$$\mathsf{AEqs}_V^1 \leq_{ssf} \mathsf{AEqs}_V^1 \sqcup \mathsf{AEqs}_V^2 \tag{9}$$

$$\mathsf{AEqs}_V^2 \leq_{ssf} \mathsf{AEqs}_V^1 \sqcup \mathsf{AEqs}_V^2 \tag{10}$$

$$\mathsf{AEqs}_V^1 \leq_{ssf} \mathsf{AEqs}_V \;\&\; \mathsf{AEqs}_V^2 \leq_{ssf} \mathsf{AEqs}_V \;\Rightarrow\; \mathsf{AEqs}_V^1 \sqcup \mathsf{AEqs}_V^2 \leq_{ssf} \mathsf{AEqs}_V \tag{11}$$

Proof Let $\mathsf{AEqs}_V^1 \sqcup \mathsf{AEqs}_V^2 = \langle \lambda(\mathcal{E}_1, \mathcal{E}_2); \Delta \rangle$, where $\mathsf{AEqs}_V^1 = \langle \mathcal{E}_1; \Delta_1 \rangle$ and $\mathsf{AEqs}_V^2 = \langle \mathcal{E}_2; \Delta_2 \rangle$. To prove (9), we define σ_1 such that for $A \in \mathsf{AVar}(\mathsf{AEqs}_V^1 \sqcup \mathsf{AEqs}_V^2)$, $\sigma_1(A) = \phi^{-1}(A)\downarrow 1$.[12] The conditions of Definition A.1 are met: $\lambda(\mathcal{E}_1, \mathcal{E}_2)\sigma_1 = \mathcal{E}_1$ by the definitions of σ_1 and $\lambda(\mathcal{E}_1, \mathcal{E}_2)$; $\forall A \in \mathsf{AVar}(\mathsf{AEqs}_V^1 \sqcup \mathsf{AEqs}_V^2)$, let $(\tau_1, \tau_2) = \phi^{-1}(A)$, then

$$\begin{aligned} ann_{\mathsf{AEqs}_V^1 \sqcup \mathsf{AEqs}_V^2}(A) &= ann_{\mathsf{AEqs}_V^1}(\tau_1) \cap ann_{\mathsf{AEqs}_V^2}(\tau_2), \\ &\qquad \text{by the definition of the least upperbound,} \\ &\subseteq ann_{\mathsf{AEqs}_V^1}(A\sigma_1), \\ &\qquad \text{since } \tau_1 = A\sigma_1 \text{ by the definition of } \sigma_1; \end{aligned}$$

and finally $\forall S \in \Delta_1.\ \bigcup_{A \in S} ASG^{\sigma_1}_{\mathsf{AEqs}_V^1 \sqcup \mathsf{AEqs}_V^2}(A) \in \Delta$, by the definition of the least upperbound and because by the definition of σ_1 we have $ASG^{\sigma_1}_{\mathsf{AEqs}_V^1 \sqcup \mathsf{AEqs}_V^2}(A) = \Pi^1_\phi(A)$.

The proof of (10) is similar (define σ_2 such that for $A \in \mathsf{AVar}(\mathsf{AEqs}_V^1 \sqcup \mathsf{AEqs}_V^2)$, $\sigma_2(A) = \phi^{-1}(A)\downarrow 2$).

To prove (11), suppose that $\mathsf{AEqs}_V^1 \leq_{ssf} \mathsf{AEqs}_V$ & $\mathsf{AEqs}_V^2 \leq_{ssf} \mathsf{AEqs}_V$, where $\mathsf{AEqs}_V = \langle \mathcal{E}; \Delta' \rangle$. Since $\lambda(\mathcal{E}_1, \mathcal{E}_2)$ is the least common anti-instance of $\mathcal{E}_1$ and $\mathcal{E}_2$ (see [25]), and $\mathcal{E}$ is an anti-instance of both $\mathcal{E}_1$ and $\mathcal{E}_2$, there exists a mapping $\sigma : \mathsf{AVar}(\mathsf{AEqs}_V) \to \mathsf{ATerm}(\mathsf{AEqs}_V^1 \sqcup \mathsf{AEqs}_V^2)$ such that $\mathcal{E}\sigma = \lambda(\mathcal{E}_1, \mathcal{E}_2)$. Let $\sigma_1' : \mathsf{AVar}(\mathsf{AEqs}_V) \to \mathsf{ATerm}(\mathsf{AEqs}_V^1)$ and $\sigma_2' : \mathsf{AVar}(\mathsf{AEqs}_V) \to \mathsf{ATerm}(\mathsf{AEqs}_V^2)$ be the mappings satisfying the conditions of Definition A.1 for AEqs_V^1 and AEqs_V^2 respectively, and let $\sigma_1 = \phi^{-1}\downarrow 1$ and $\sigma_2 = \phi^{-1}\downarrow 2$ as above. Since $\mathcal{E}\sigma_1' = \mathcal{E}_1 = \lambda(\mathcal{E}_1, \mathcal{E}_2)\sigma_1 = \mathcal{E}\sigma\sigma_1$ and $\mathcal{E}\sigma_2' = \mathcal{E}_2 = \lambda(\mathcal{E}_1, \mathcal{E}_2)\sigma_2 = \mathcal{E}\sigma\sigma_2$, we have

$$\sigma_1' \restriction \mathsf{AVar}(\mathcal{E}) = \sigma\sigma_1 \restriction \mathsf{AVar}(\mathcal{E}) \quad \& \quad \sigma_2' \restriction \mathsf{AVar}(\mathcal{E}) = \sigma\sigma_2 \restriction \mathsf{AVar}(\mathcal{E}). \tag{12}$$

For all $A \in \mathsf{AVar}(\mathsf{AEqs}_V)$, by assumption $ann_{\mathsf{AEqs}_V}(A) \subseteq ann_{\mathsf{AEqs}_V^1}(A\sigma_1')$ and $ann_{\mathsf{AEqs}_V}(A) \subseteq ann_{\mathsf{AEqs}_V^2}(A\sigma_2')$. It follows from (12) that for all $A \in \mathsf{AVar}(\mathsf{AEqs}_V)$

$$ann_{\mathsf{AEqs}_V}(A) \subseteq ann_{\mathsf{AEqs}_V^1}(A\sigma\sigma_1) \quad \& \quad ann_{\mathsf{AEqs}_V}(A) \subseteq ann_{\mathsf{AEqs}_V^2}(A\sigma\sigma_2). \tag{13}$$

Since by the definition of least upperbound, for $\hat{A} \in \mathsf{AVar}(\lambda(\mathcal{E}_1, \mathcal{E}_2))$, $ann_{\mathsf{AEqs}_V^1 \sqcup \mathsf{AEqs}_V^2}(\hat{A}) = ann_{\mathsf{AEqs}_V^1}(\hat{A}\sigma_1) \cap ann_{\mathsf{AEqs}_V^2}(\hat{A}\sigma_2)$, and by the definition of σ, for $A \in \mathsf{AVar}(\mathsf{AEqs}_V)$, $A\sigma$ is

[12] We define $(a, b)\downarrow 1 = a$ and $(a, b)\downarrow 2 = b$.

an abstract term such that $\mathsf{AVar}(A\sigma) \subseteq \mathsf{AVar}(\lambda(\mathcal{E}_1, \mathcal{E}_2))$, it follows from the generalized definition of annotation for abstract terms that for all $A \in \mathsf{AVar}(\mathsf{AEqs}_V)$

$$ann_{\mathsf{AEqs}^1_V \sqcup \mathsf{AEqs}^2_V}(A\sigma) = ann_{\mathsf{AEqs}^1_V}(A\sigma\sigma_1) \cap ann_{\mathsf{AEqs}^2_V}(A\sigma\sigma_2). \tag{14}$$

Hence, from (13) and (14), $ann_{\mathsf{AEqs}_V}(A) \subseteq ann_{\mathsf{AEqs}^1_V \sqcup \mathsf{AEqs}^2_V}(A\sigma)$, as desired.

Finally, by the definition of least upperbound, σ_1, and σ_2,

$$\Delta = \{ \bigcup_{A \in S_1} ASG^{\sigma_1}_{\mathsf{AEqs}^1_V \sqcup \mathsf{AEqs}^2_V}(A) \mid S_1 \in \Delta_1 \} \cup \{ \bigcup_{A \in S_2} ASG^{\sigma_2}_{\mathsf{AEqs}^1_V \sqcup \mathsf{AEqs}^2_V}(A) \mid S_2 \in \Delta_2 \}.$$

We have to prove that $\forall S \in \Delta.\ \bigcup_{A \in S} ASG^{\sigma}_{\mathsf{AEqs}_V}(A) \in \Delta'$. Assume (without loss of generality) that $S = \bigcup_{A \in S_1} ASG^{\sigma_1}_{\mathsf{AEqs}^1_V \sqcup \mathsf{AEqs}^2_V}(A)$, where $S_1 \in \Delta_1$. By assumption,

$$\forall S_1 \in \Delta_1.\ \bigcup_{A \in S_1} ASG^{\sigma'_1}_{\mathsf{AEqs}_V}(A) \in \Delta' \quad \& \quad \forall S_2 \in \Delta_2.\ \bigcup_{A \in S_2} ASG^{\sigma'_2}_{\mathsf{AEqs}_V}(A) \in \Delta'. \tag{15}$$

So,

$$\begin{aligned} \bigcup_{A \in S} ASG^{\sigma}_{\mathsf{AEqs}_V}(A) &= \bigcup_{A \in S_1} ASG^{\sigma\sigma_1}_{\mathsf{AEqs}_V}(A) \\ &\qquad \text{by the definition of abstract sharing group,} \\ &= \bigcup_{A \in S_1} ASG^{\sigma'_1}_{\mathsf{AEqs}_V}(A) \\ &\qquad \text{since } \sigma'_1 \restriction \mathsf{AVar}(\mathcal{E}) = \sigma\sigma_1 \restriction \mathsf{AVar}(\mathcal{E}), \\ &\in \Delta', \qquad \text{by (15).} \end{aligned}$$

□

The projection operation serves to restrict the information expressed by an abstract equation system AEqs_{V_1} to a smaller set V_2 of concrete variables of interest. If AEqs_{V_1} is in solution solved form, the definition is straightforward.

Definition A.3 (Projection) *Let $\mathsf{AEqs}_{V_1} = \langle \mathcal{E}; \Delta \rangle$ be an abstract equation system in solution solved form and $V_2 \subseteq V_1$. The projection AEqs_{V_2} of AEqs_{V_1} on V_2, is defined by*

$$\begin{aligned} \mathsf{AEqs}_{V_2} &= \langle \mathcal{E}'; \Delta' \rangle, \\ \mathcal{E}' &= \{ (X = t) \in \mathcal{E} \mid X \in V_2 \}, \\ \Delta' &= \{ S \cap \mathsf{AVar}(\mathcal{E}') \mid S \in \Delta \} \setminus \{\emptyset\}, \\ ann_{\mathsf{AEqs}_{V_2}} &= ann_{\mathsf{AEqs}_{V_1}} \restriction \mathsf{AVar}(\mathcal{E}'). \end{aligned}$$

The extension operation involves a unification of two abstract equation systems.

Definition A.4 (Extension) *Let $\mathsf{AEqs}^1_V = \langle \mathcal{E}_1; \Delta_1 \rangle$ and $\mathsf{AEqs}^2_{V'} = \langle \mathcal{E}_2; \Delta_2 \rangle$ be two renamed apart*[13] *abstract equation systems in solution solved form, with $V' \subseteq V$. The extension of AEqs^1_V and $\mathsf{AEqs}^2_{V'}$ is defined as the solved form of $\mathsf{AEqs}_V = \langle \mathcal{E}; \Delta \rangle$ where $\mathcal{E} = \mathcal{E}_1 \cup \mathcal{E}_2$, $\Delta = \Delta_1 \cup \Delta_2$, and for $A \in \mathsf{AVar}(\mathsf{AEqs}^1_V)$, $ann_{\mathsf{AEqs}_V}(A) = ann_{\mathsf{AEqs}^1_V}(A)$, and for $A \in \mathsf{AVar}(\mathsf{AEqs}^2_V)$, $ann_{\mathsf{AEqs}_V}(A) = ann_{\mathsf{AEqs}^2_{V'}}(A)$.*

When the operations specified in this section (least upperbound, projection, and extension) introduce redundancies in the representation, the Reduce function can be applied to transform the result abstract equation system into reduced form.

[13] By this we mean that $\mathsf{AVar}(\mathsf{AEqs}^1_V) \cap \mathsf{AVar}(\mathsf{AEqs}^2_{V'}) = \emptyset$.

B Example Analysis Trace

Source code:
(Cl1) `append( nil, P, P ).` ①
(Cl2) `append( [Q|R], S, [Q|T] ) :-` ② `append( R, S, T ).` ③

Goal: `append(X,Y,Z)`

In the table below, we show an analysis trace for the `append/3` predicate starting from a call specification AEqs_{in} that states that the program variables of the goal, $V_{goal} = \{X, Y, Z\}$, are all non-sharing and free variables.

AEqs_{in}	$\{\}; \{\}$
AEqs_1	$\{X = nil, Y = P, Z = P\}; \{\}$
AEqs_{Cl1}	$\{X = nil, Y = A_1^f, Z = A_1^f\}; \{\{A_1^f\}\}$
AEqs_2	$\{X = [Q\|R], Y = S, Z = [Q\|T]\}; \{\}$
AEqs_3	$\{X = [Q\|nil], Y = A_1^f, Z = [Q\|A_1^f], R = nil, S = A_1^f, T = A_1^f\};$ $\{\{A_1^f\}\}$
AEqs_{Cl2}	$\{X = [A_2^f\|nil], Y = A_1^f, Z = [A_2^f\|A_1^f]\}; \{\{A_1^f\}, \{A_2^f\}\}$
AEqs_{out}	$\{X = A_3^{c\ell}, Y = A_4^f, Z = A_5^\ell\}; \{\{A_3^{c\ell}, A_5^\ell\}, \{A_4^f, A_5^\ell\}\}$
AEqs'_3	$\{X = [Q\|A_3^{c\ell}], Y = A_4^f, Z = [Q\|A_5^\ell], R = A_3^{c\ell}, S = A_4^f, T = A_5^\ell\};$ $\{\{A_3^{c\ell}, A_5^\ell\}, \{A_4^f, A_5^\ell\}\}$
AEqs'_{Cl2}	$\{X = [A_2^f\|A_3^{c\ell}], Y = A_4^f, Z = [A_2^f\|A_5^\ell]\};$ $\{\{A_2^f\}, \{A_3^{c\ell}, A_5^\ell\}, \{A_4^f, A_5^\ell\}\}$
AEqs'_{out}	$\{X = A_3^{c\ell}, Y = A_4^f, Z = A_5^\ell\}; \{\{A_3^{c\ell}, A_5^\ell\}, \{A_4^f, A_5^\ell\}\}$

On procedure-entry to clause Cl1, the abstract unification `append(X,Y,Z) = append(nil, P,P)` is performed for the set of program variables $V_{goal} \cup V_{Cl1} = \{X, Y, Z, P\}$, and yields an abstract solved form, AEqs_1, for program point ① which is the same as the concrete solved form that would result from the corresponding concrete unification.

Next, AEqs_1 must be restricted back to the set of variables V_{goal}. To this end, a new abstract variable is introduced and leads to AEqs_{Cl1}, i.e., the success abstract equation system for clause Cl1. The system AEqs_{Cl1} can also be considered as a first approximation of the output abstract equation system for the call `append(X,Y,Z)`.

On procedure-entry to clause Cl2, the unification `append(X,Y,Z) = append([Q|R],S, [Q|T])` is performed for the set of program variables $V_{goal} \cup V_{Cl2} = \{X, Y, Z, Q, R, S, T\}$ and yields an abstract solved form, AEqs_2, for program point ② which is again the same as would result from the corresponding concrete unification.

Note that the recursive call `append(R,S,T)` has the same pattern as the initial goal (it is obtained by restricting AEqs_2 to the set of variables $\{R, S, T\}$).

Procedure-exit yields the abstract equation system AEqs_3 for program point ③. It is obtained by taking the union of AEqs_2 and a renaming of the first approximation of the output, AEqs_{Cl1}, to the set of variables $\{R, S, T\}$, and reducing it to solved form.

In order to restrict AEqs_3 back to the set of variables V_{goal}, a new abstract variable is introduced. The result is AEqs_{Cl2}, the success abstract equation system for clause Cl2.

Now the upperbound of AEqs_{Cl1} and AEqs_{Cl2} has to be computed in order to obtain a second approximation for the output abstract equation system, AEqs_{out}. Note that no

structural information (except definite compoundness of the first argument) is retained and no depth-bound restriction is applied in this case.

Finally, using the second approximation, AEqs_{out}, in the procedure-exit operation at program point ③, and restarting the computation from there, we obtain new values AEqs'_3, AEqs'_{Cl2}, and AEqs'_{out}, which shows that a fixpoint has been reached.