The Correctness of Set-Sharing

Patricia M. Hill[1], Roberto Bagnara[2*], and Enea Zaffanella[3]

[1] School of Computer Studies,
University of Leeds,
Leeds, LS2 9JT, United Kingdom
hill@scs.leeds.ac.uk

[2] Dipartimento di Matematica,
Università degli Studi di Parma, Italy.
bagnara@prmat.math.unipr.it

[3] Servizio IX Automazione,
Università degli Studi di Modena, Italy.
zaffanella@elektra.casa.unimo.it

Abstract. It is important that practical data flow analysers are backed by reliably proven theoretical results. Abstract interpretation provides a sound mathematical framework and necessary generic properties for an abstract domain to be well-defined and sound with respect to the concrete semantics. In logic programming, the abstract domain Sharing is a standard choice for sharing analysis for both practical work and further theoretical study. In spite of this, we found that there were no satisfactory proofs for the key properties of commutativity and idempotence that are essential for Sharing to be well-defined and that published statements of the safeness property assumed the occur-check. This paper provides a generalisation of the abstraction function for Sharing that can be applied to any language, with or without the occur-check. The results for safeness, idempotence and commutativity for abstract unification using this abstraction function are given.

Keywords: abstract interpretation, logic programming, occur-check, rational trees, set-sharing.

1 Introduction

Today, talking about sharing analysis for logic programs is almost the same as talking about the *set-sharing* domain Sharing of Jacobs and Langen [8, 9]. Researchers are primarily concerned with extending the domain with linearity, freeness, depth-k abstract substitutions and so on [2, 4, 12, 13, 16]. Key properties such as commutativity and soundness of this domain and its associated abstract operations are normally assumed to hold. The main reason for this is that [9] not only includes a proof of the soundness but also refers the reader to the thesis of Langen [14] for proofs of commutativity and idempotence.

In abstract interpretation, the concrete semantics of a program is approximated by an abstract semantics. In particular, the concrete domain is replaced

* Much of this work was supported by EPSRC grant GR/L19515.

by an abstract domain and each elementary operation on the concrete domain is replaced by a corresponding abstract operation on the abstract domain. Thus, assuming the global abstract procedure mimics the concrete execution procedure, each operation on elements in the abstract domain must produce an approximation of the corresponding operation on corresponding elements in the concrete domain. The key operation in a logic programming derivation is unification (*unify*) and the corresponding operation for an abstract domain is *aunify*.

An important step in standard unification algorithms is the *occur-check* that avoids the generation of infinite data structures. However, in computational terms, it is expensive and it is well known that Prolog implementations by default omit this check. Although standard unification algorithms that include the occur-check produce a substitution that is idempotent, the resulting substitution when the occur-check is omitted, may not be idempotent. In spite of this, most theoretical work on data-flow analysis of logic programming assume the result of *unify* is always idempotent. In particular both [9] and [14] assume in their proofs of soundness that the concrete substitutions are idempotent. Thus their results do not apply to the analysis of all Prolog programs.

If two terms in the concrete domain are unifiable, then *unify* computes the most general unifier (*mgu*). Up to renaming of variables, an mgu is unique. Moreover a substitution is defined as a *set* of bindings or equations between variables and other terms. Thus, for the concrete domain, the order and multiplicity of elements are irrelevant in both the computation and semantics of *unify*. It is therefore useful that the abstraction of the unification procedure should be unaffected by the order and multiplicity in which it abstracts the bindings that are present in the substitution. Furthermore, from a practical perspective, it is useful if the global abstract procedure can proceed in a different order to the concrete one without affecting the accuracy of the analysis results. Hence, it is extremely desirable that aunify is also commutative and idempotent. However, as discussed later in this paper, only a weak form of idempotence has ever been proved while the only previous proof of commutativity [14] is seriously flawed.

As sharing is normally combined with linearity and freeness domains that are not idempotent or commutative, [2, 12] it may be asked why these properties are important for sharing. In answer to this, we observe that the order and multiplicity in which the bindings in a substitution are analysed affects the accuracy of the linearity and freeness domains. It is therefore a real advantage to be able to ignore these aspects as far as the sharing domain is concerned.

This paper provides a generalisation of the abstraction function for $\mathsf{Sharing}$ that can be applied to any language, with or without the occur-check. The results for safeness, idempotence and commutativity for abstract unification using this abstraction function are given. Detailed proofs of the results stated in this paper are available in [7].

In the next section, the notation and definitions needed for equality and substitutions in the concrete domain are given. In Section 3, we introduce a new concept called *variable-idempotence* that generalises idempotence to allow for rational trees. In Section 4, we recall the definition of $\mathsf{Sharing}$ and define its abstraction function, generalised to allow for non-idempotent substitutions. We conclude in Section 5.

2 Equations and Substitutions

2.1 Notation

For a set S, $\# S$ is the cardinality of S, $\wp(S)$ is the powerset of S, whereas $\wp_{\mathrm{f}}(S)$ is the set of all the *finite* subsets of S. The symbol Vars denotes a denumerable set of variables, whereas $\mathcal{T}_{\mathit{Vars}}$ denotes the set of first-order terms over Vars for some given set of function symbols. The set of variables occurring in a syntactic object o is denoted by $\mathit{vars}(o)$.

2.2 Substitutions

If $x \in \mathit{Vars}$ and $s \in \mathcal{T}_{\mathit{Vars}}$, then $x \mapsto s$ is called a *binding*. A substitution is a total function $\sigma \colon \mathit{Vars} \to \mathcal{T}_{\mathit{Vars}}$ that is the identity almost everywhere; in other words, the *domain* of σ,

$$\mathrm{dom}(\sigma) \stackrel{\mathrm{def}}{=} \{\, x \in \mathit{Vars} \mid \sigma(x) \neq x \,\}$$

is finite. If $t \in \mathcal{T}_{\mathit{Vars}}$, we write $t\sigma$ to denote $\sigma(t)$.

Substitutions are denoted by the set of their *bindings*, thus σ is identified with the set $\{\, x \mapsto \sigma(x) \mid x \in \mathrm{dom}(\sigma) \,\}$. The composition of substitutions is defined in the usual way. Thus $\tau \circ \sigma$ is the substitution such that, for all terms t, $(\tau \circ \sigma)(t) = \tau(\sigma(t))$. A substitution is said *circular* if it has the form $\{x_1 \mapsto x_2, \ldots, x_{n-1} \mapsto x_n, x_n \mapsto x_1\}$. A substitution is in *rational solved form* if it has no circular subset. The set of all substitutions in rational solved form is denoted by Subst.

2.3 Equations

An *equation* is of the form $s = t$ where $s, t \in \mathcal{T}_{\mathit{Vars}}$. Eqs denotes the set of all equations.

We are concerned in this paper to keep the results on sharing as general as possible. In particular, we do not want to restrict ourselves to a specific equality theory. Thus we allow for any equality theory T over $\mathcal{T}_{\mathit{Vars}}$ that includes the *basic axioms* denoted by the following schemata.

$$s = s, \tag{1}$$

$$s = t \iff t = s, \tag{2}$$

$$r = s \land s = t \implies r = t, \tag{3}$$

$$f(s_1, \ldots, s_n) = f(t_1, \ldots, t_n) \iff s_1 = t_1, \ldots, s_n = t_n. \tag{4}$$

Of course, T can include other axioms. For example, it is usual in logic programming and most implementations of Prolog to assume an equality theory based on syntactic identity and characterised by the axiom schemata given by Clark [3]. This consists of the basic axioms together with the following:

$$\neg f(s_1, \dots, s_n) = g(t_1, \dots, t_m) \tag{5}$$

$$\forall z \in \mathit{Vars}\ \forall t \in (\mathcal{T}_{\mathit{Vars}} \setminus \mathit{Vars}) : z \in \mathit{vars}(t) \implies \neg(z = t). \tag{6}$$

The *identity axioms* characterised by the schemata 5 ensure the equality theory is Herbrand and depends only on the syntax. Equality theory for a non-Herbrand domain replaces these axioms by ones that depend instead on the semantics of the domain. Axioms characterised by the schemata 6 are called the *occur-check axioms* and are an essential part of the standard unification procedure in SLD-resolution.

An alternative approach used in some implementations of Prolog, does not require the occur-check axioms. This approach is based on the theory of rational trees [5, 6]. It assumes the basic axioms and the identity axioms together with a set of *uniqueness axioms* [10, 11]. These state that each equation in rational solved form uniquely defines a set of trees. Thus, an equation $z = t$ where $z \in \mathit{vars}(t)$ and $t \in (\mathcal{T}_{\mathit{Vars}} \setminus \mathit{Vars})$ denotes the axiom (expressed in terms of the usual first-order quantifiers [15]):

$$\forall x \in \mathit{Vars}\ : \bigl(z = t \land (x = t\{z \mapsto x\} \implies z = x)\bigr).$$

The basic axioms defined by schemata 1, 2, 3, and 4, which are all that are required for the results in this paper, are included in both these theories.

A substitution σ may be regarded as a set of equations $\{\, x = t \mid x \mapsto t \in \sigma \,\}$. A set of equations $e \in \wp_{\mathrm{f}}(\mathrm{Eqs})$ is *unifiable* if there is $\sigma \in \mathit{Subst}$ such that $T \vdash (\sigma \implies e)$. σ is called a *unifier* for e. σ is said to be a *relevant* unifier of e if $\mathit{vars}(\sigma) \subseteq \mathit{vars}(e)$. That is, σ does not introduce any new variables. σ is a most general unifier for e if, for every unifier σ' of e, $T \vdash (\sigma' \implies \sigma)$. An mgu, if it exists, is unique up to the renaming of variables. In this paper, $\mathrm{mgu}(e)$ always denotes a relevant unifier of e.

3 Variable-Idempotence

It is usual in papers on sharing analysis to assume that all the substitutions are idempotent. Note that a substitution σ is *idempotent* if, for all $t \in \mathcal{T}_{\mathit{Vars}}$, $t\sigma\sigma = t\sigma$. However, the sharing domain is just concerned with the variables. So, to allow for substitutions representing rational trees, we generalise idempotence to *variable-idempotence.*

Definition 1. *A substitution σ is* variable-idempotent *if*

$$\forall t \in \mathcal{T}_{\mathit{Vars}} : \mathit{vars}(t\sigma\sigma) = \mathit{vars}(t\sigma).$$

The set of all variable-idempotent substitutions is denoted by VSubst.

It is convenient to use the following alternative characterisation of variable-idempotence: A substitution σ is variable-idempotent if and only if,

$$\forall (x \mapsto t) \in \sigma : \mathit{vars}(t\sigma) = \mathit{vars}(t).$$

Thus any substitution consisting of a single binding is variable-idempotent. Moreover, all idempotent substitutions are also variable-idempotent.

Example 1. The substitution $\{x \mapsto f(x)\}$ is not idempotent but is variable-idempotent. Also, $\{x \mapsto f(y), y \mapsto z\}$ is not idempotent or variable-idempotent but is equivalent (with respect to some equality theory T) to $\{x \mapsto f(z), y \mapsto z\}$, which is idempotent.

We define the transformation $\overset{\mathcal{S}}{\longmapsto} \subseteq \mathit{Subst} \times \mathit{Subst}$, called $\mathcal{S}$-transformation, as follows:

$$\frac{(x \mapsto t) \in \sigma \qquad (y \mapsto s) \in \sigma \qquad x \neq y}{\sigma \overset{\mathcal{S}}{\longmapsto} \bigl(\sigma \setminus \{y \mapsto s\}\bigr) \cup \{y \mapsto s[x/t]\}}$$

Any substitution σ can be transformed to a *variable-idempotent substitution* σ' *for* σ by a finite sequence of $\mathcal{S}$-transformations. Furthermore, if the substitutions σ and σ' are regarded as equations, then they are equivalent with respect to any equality theory that includes the basic equality axioms. These two statements are direct consequences of Lemmas 1 and 2, respectively.

Lemma 1. *Let T be an equality theory that satisfies the basic equality axioms. Suppose that σ and σ' are substitutions such that $\sigma \overset{\mathcal{S}}{\longmapsto} \sigma'$. Then, regarding σ and σ' as sets of equations, $T \vdash (\sigma \iff \sigma')$.*

Proof. Suppose that $(x \mapsto t), (y \mapsto s) \in \sigma$ where $x \neq y$ and suppose also $\sigma' = \bigl(\sigma \setminus \{y \mapsto s\}\bigr) \cup \{y \mapsto s[x/t]\}$. We first show by induction on the depth of the term s that

$$x = t \implies s = s[x/t].$$

Suppose s has depth 1. If s is x, then $s[x/t] = t$ and the result is trivial. If s is a variable distinct from x or a constant, then $s[x/t] = s$ and the result follows from equality Axiom 1. Suppose now that $s = f(s_1, \ldots, s_n)$ and the result holds for all terms of depth less than that of s. Then, by the inductive hypothesis, for each $i = 1, \ldots, n$,

$$x = t \implies s_i = s_i[x/t].$$

Hence, by Axiom 4,

$$x = t \implies f(s_1, \ldots, s_n) = f\bigl(s_1[x/t], \ldots, s_n[x/t]\bigr)$$

and hence

$$x = t \implies f(s_1, \ldots, s_n) = f(s_1, \ldots, s_n)[x/t].$$

Thus, combining this result with Axiom 3, we have

$$\begin{aligned}\{x = t, y = s\} &\implies \{x = t, y = s, s = s[x/t]\}\\ &\implies \{x = t, y = s[x/t]\}.\end{aligned}$$

Similarly, combining this result with Axioms 2 and 3,

$$\begin{aligned}\{x = t, y = s[x/t]\} &\implies \{x = t, y = s[x/t], s = s[x/t]\}\\ &\implies \{x = t, y = s\}.\end{aligned}$$

□

Note that the condition $x \neq y$ in Lemma1 is necessary. For example, suppose $\sigma = \{x \mapsto f(x)\}$ and $\sigma' = \{x \mapsto f(f(x))\}$. Then we do not have $\sigma' \implies \sigma$.

Lemma 2. *Suppose that, for each $j = 0, \dots, n$:*

$$\sigma_j = \{x_1 \mapsto t_{1,j}, \dots, x_n \mapsto t_{n,j}\},$$

where $t_{j,j} = t_{j,j-1}$ and if $j > 0$, for each $i = 1, \dots, n$, where $i \neq j$, $t_{i,j} = t_{i,j-1}[x_j/t_{j,j-1}]$. Then, for each $j = 0, \dots, n$,

$$\nu_j = \{x_1 \mapsto t_{1,j}, \dots, x_j \mapsto t_{j,j}\}$$

is variable-idempotent and, if $j > 0$, σ_j can be obtained from σ_{j-1} by a sequence of $\mathcal{S}$-transformations.

Proof. The proof is by induction on j. Since ν_0 is empty, the base case when $j = 0$ is trivial. Suppose, therefore that $1 \leq j \leq n$ and the hypothesis holds for ν_{j-1} and σ_{j-1}. By the definition of ν_j, we have $\nu_j = \{x_j \mapsto t_{j,j-1}\} \circ \nu_{j-1}$. Consider an arbitrary i, $1 \leq i \leq j$. We will show that $vars(t_{i,j}\nu_j) = vars(t_{i,j})$.

Suppose first that $i = j$. Then since $t_{j,j} = t_{j,j-1}$, $t_{j,j-1} = t_{j,0}\nu_{j-1}$ and, by the inductive hypothesis, $vars(t_{j,0}\nu_{j-1}\nu_{j-1}) = vars(t_{j,0}\nu_{j-1})$, we have

$$\begin{aligned}vars(t_{j,j}\nu_j) &= vars\big(t_{j,0}\nu_{j-1}\nu_{j-1}\{x_j \mapsto t_{j,j}\}\big)\\ &= vars\big(t_{j,0}\nu_{j-1}\{x_j \mapsto t_{j,j}\}\big)\\ &= vars\big(t_{j,j}\{x_j \mapsto t_{j,j}\}\big)\\ &= vars(t_{j,j}).\end{aligned}$$

Suppose now that $i \neq j$. Then,

$$vars(t_{i,j}) = vars\big(t_{i,j-1}\{x_j \mapsto t_{j,j-1}\}\big).$$

and, by the inductive hypothesis, $vars(t_{i,j-1}\nu_{j-1}) = vars(t_{i,j-1})$.

If $x_j \notin vars(t_{i,j-1})$, then

$$\begin{aligned}vars(t_{i,j}\nu_{j-1}) &= vars\big(t_{i,j-1}\{x_j \mapsto t_{j,j-1}\}\nu_{j-1}\big)\\ &= vars(t_{i,j-1}\nu_{j-1})\\ &= vars(t_{i,j}).\end{aligned}$$

On the other hand, if $x_j \in vars(t_{i,j-1})$, then

$$\begin{aligned}
vars(t_{i,j}\nu_{j-1}) &= vars\bigl(t_{i,j-1}\{x_j \mapsto t_{j,j-1}\}\nu_{j-1}\bigr) \\
&= vars(t_{i,j-1}\nu_{j-1}) \setminus \{x_j\} \cup vars(t_{j,j-1}\nu_{j-1}) \\
&= vars(t_{i,j-1}) \setminus \{x_j\} \cup vars(t_{j,j-1}) \\
&= vars\bigl(t_{i,j-1}\{x_j \mapsto t_{j,j-1}\}\bigr) \\
&= vars(t_{i,j}).
\end{aligned}$$

Thus, in both cases,

$$\begin{aligned}
vars(t_{i,j}\nu_j) &= vars\bigl(t_{i,j}\nu_{j-1}\{x_j \mapsto t_{j,j-1}\}\bigr) \\
&= vars\bigl(t_{i,j}\{x_j \mapsto t_{j,j-1}\}\bigr) \\
&= vars\bigl(t_{i,j-1}\{x_j \mapsto t_{j,j-1}\}\{x_j \mapsto t_{j,j-1}\}\bigr).
\end{aligned}$$

However, a substitution consisting of a single binding is variable-idempotent. Thus

$$\begin{aligned}
vars(t_{i,j}\nu_j) &= vars\bigl(t_{i,j-1}\{x_j \mapsto t_{j,j-1}\}\bigr) \\
&= vars(t_{i,j}).
\end{aligned}$$

Therefore, for each $i = 1, \dots, j$, $vars(t_{i,j}\nu_j) = vars(t_{i,j})$. It then follows (using the alternative characterisation of variable-idempotence) that ν_j is variable-idempotent. $\square$

Example 2. Let

$$\sigma_0 = \bigl\{x_1 \mapsto f(x_2), x_2 \mapsto g(x_3, x_4), x_3 \mapsto x_1\bigr\}.$$

Then

$$\begin{aligned}
\sigma_1 &= \bigl\{x_1 \mapsto f(x_2), x_2 \mapsto g(x_3, x_4), x_3 \mapsto f(x_2)\bigr\}, \\
\sigma_2 &= \bigl\{x_1 \mapsto f(g(x_3, x_4)), x_2 \mapsto g(x_3, x_4), x_3 \mapsto f(g(x_3, x_4))\bigr\}, \\
\sigma_3 &= \bigl\{x_1 \mapsto f(g(f(g(x_3, x_4)), x_4)), x_2 \mapsto g(f(g(x_3, x_4)), x_4), x_3 \mapsto f(g(x_3, x_4))\bigr\}.
\end{aligned}$$

Note that σ_3 is variable-idempotent and that $T \vdash \sigma_0 \iff \sigma_3$.

4 Set-Sharing

4.1 The Sharing Domain

The Sharing domain is due to Jacobs and Langen [8]. However, we use the definition as presented in [1].

Definition 2. (The *set-sharing* lattice.) *Let*

$$SG \stackrel{\text{def}}{=} \{\, S \in \wp_{\mathrm{f}}(Vars) \mid S \neq \varnothing \,\}$$

and let $SH \stackrel{\text{def}}{=} \wp(SG)$. *The* set-sharing lattice *is given by the set*

$$SS \stackrel{\text{def}}{=} \{\, (sh, U) \mid sh \in SH, U \in \wp_{\mathrm{f}}(Vars), \forall S \in sh : S \subseteq U \,\} \cup \{\bot, \top\}$$

ordered by $\preceq_{SS}$ *defined as follows, for each* $d, (sh_1, U_1), (sh_2, U_2) \in SS$*:*

$$\begin{aligned} \bot &\preceq_{SS} d, \\ d &\preceq_{SS} \top, \\ (sh_1, U_1) \preceq_{SS} (sh_2, U_2) \quad &\iff \quad (U_1 = U_2) \wedge (sh_1 \subseteq sh_2). \end{aligned}$$

It is straightforward to see that every subset of SS has a least upper bound with respect to $\preceq_{SS}$. Hence SS is a complete lattice.[1]

An element sh of SH abstracts the property of sharing in a substitution σ. That is, if σ is idempotent, two variables x, y must be in the same set in sh if some variable, say v occurs in both $x\sigma$ and $y\sigma$. In fact, this is also true for variable-idempotent substitutions although it is shown below that this needs to be generalised for substitutions that are not variable-idempotent. Thus, the definition of the abstraction function α for sharing, requires an ancillary definition for the notion of *occurrence.*

Definition 3. (Occurrence.)
For each $n \in \mathbb{N}$, $\mathrm{occ}_i\colon Subst \times Vars \to \wp_{\mathrm{f}}(Vars)$ *is defined for each* $\sigma \in Subst$ *and each* $v \in Vars$*:*

$$\begin{aligned} \mathrm{occ}_0(\sigma, v) &\stackrel{\text{def}}{=} \{v\}, && \text{if } v = v\sigma; \\ \mathrm{occ}_0(\sigma, v) &\stackrel{\text{def}}{=} \varnothing, && \text{if } v \neq v\sigma; \\ \mathrm{occ}_n(\sigma, v) &\stackrel{\text{def}}{=} \{\, y \in Vars \mid x \in vars(y\sigma) \cap \mathrm{occ}_{n-1}(\sigma, v) \,\}, && \text{if } n > 0. \end{aligned}$$

It follows that, for fixed values of σ *and* v, $\mathrm{occ}_n(\sigma, v)$ *is monotonic and extensive with respect to the index* n. *Hence, as the range of* $\mathrm{occ}_n(\sigma, v)$ *is restricted to the finite set of variables in* σ, *there is an* $\ell = \ell(\sigma, v) \in \mathbb{N}$ *such that* $\mathrm{occ}_\ell(\sigma, v) = \mathrm{occ}_n(\sigma, v))$ *for all* $n \geq \ell$. *Let*

$$\mathrm{occ!}(\sigma, v) \stackrel{\text{def}}{=} \mathrm{occ}_\ell(\sigma, v).$$

Note that if σ is variable-idempotent, then $\mathrm{occ!}(\sigma, v) = \mathrm{occ}_1(\sigma, v)$. Note also that if $v \neq v\sigma$, then $\mathrm{occ!}(\sigma, v) = \varnothing$. Previous definitions for an occurrence operator such as that for sg in [8] have all been for idempotent substitutions. However, when σ is an idempotent substitution, $\mathrm{occ!}(\sigma, v)$ and $sg(\sigma, v)$ are the same for all $v \in Vars$.

We base the definition of abstraction on the occurrence operator, occ!.

---

[1] Notice that the only reason we have $\top \in SS$ is in order to turn SS into a lattice rather than a CPO.

Definition 4. (Abstraction.) *The concrete domain Subst is related to SS by means of the* abstraction function $\alpha\colon \wp(\mathit{Subst}) \times \wp_f(\mathit{Vars}) \to \mathit{SS}$. *For each* $\Sigma \in \wp(\mathit{Subst})$ *and each* $U \in \wp_f(\mathit{Vars})$,

$$\alpha(\Sigma, U) \stackrel{\mathrm{def}}{=} \bigsqcup_{\sigma \in \Sigma} \alpha(\sigma, U),$$

where $\alpha\colon \mathit{Subst} \times \wp_f(\mathit{Vars}) \to \mathit{SS}$ *is defined, for each* $\sigma \in \mathit{Subst}$ *and each* $U \in \wp_f(\mathit{Vars})$, *by*

$$\alpha(\sigma, U) \stackrel{\mathrm{def}}{=} \Bigl(\bigl\{ \operatorname{occ!}(\sigma, v) \cap U \bigm| v \in \mathit{Vars} \bigr\} \setminus \{\varnothing\}, U \Bigr).$$

The following result states that the abstraction for a substitution σ is the same as the abstraction for a variable-idempotent substitution for σ.

Lemma 3. *Let* σ *be a substitution,* σ' *a substitution obtained from* σ *by a sequence of* $\mathcal{S}$*-transformations,* U *a set of variables and* $v \in \mathit{Vars}$. *Then*

$$v = v\sigma \iff v = v\sigma', \quad \operatorname{occ!}(\sigma, v) = \operatorname{occ!}(\sigma', v), \quad \textit{and } \alpha(\sigma, U) = \alpha(\sigma', U).$$

Proof. Suppose first that σ' is obtained from σ by a single $\mathcal{S}$-transformation. Thus we can assume that $x \mapsto t$ and $y \mapsto s$ are in σ where $x \in \mathit{vars}(s)$ and that

$$\sigma' = \bigl(\sigma \setminus \{y \mapsto s\}\bigr) \cup \bigl\{y \mapsto s[x/t]\bigr\}.$$

It follows that, since σ is in rational solved form, σ has no circular subset and hence $v = v\sigma \iff v = v\sigma'$. Thus, if $v \neq v\sigma$, then we have $v \neq v\sigma'$ and $\operatorname{occ!}(\sigma, v) = \operatorname{occ!}(\sigma', v) = \varnothing$.

We now assume that $v = v\sigma = v\sigma'$ and prove that

$$\operatorname{occ}_m(\sigma, v) \subseteq \operatorname{occ!}(\sigma', v).$$

The proof is by induction on m. By Definition 3, $\operatorname{occ}_0(\sigma, v) = \operatorname{occ}_0(\sigma', v) = \{v\}$, so that the result holds for $m = 0$. Suppose then that $m > 0$ and that $v_m \in \operatorname{occ}_m(\sigma, v)$. By Definition 3, there exists $v_{m-1} \in \mathit{vars}(v_m\sigma)$ where $v_{m-1} \in \operatorname{occ}_{m-1}(\sigma, v)$. Hence, by the inductive hypothesis, $v_{m-1} \in \operatorname{occ!}(\sigma', v)$. If $v_{m-1} \in \mathit{vars}(v_m\sigma')$, then, by Definition 3, $v_m \in \operatorname{occ!}(\sigma', v)\bigr)$. On the other hand, if $v_{m-1} \notin \mathit{vars}(v_m\sigma')$, then $v_m = y$, $v_{m-1} = x$, and $x \in \mathit{vars}(s)$ (so that $\mathit{vars}(t) \subseteq \mathit{vars}(s[x/t])$). However, by hypothesis, $v = v\sigma$, so that $x \neq v$ and $m > 1$. Thus, by Definition 3, there exists $v_{m-2} \in \mathit{vars}(t)$ such that $v_{m-2} \in \operatorname{occ}_{m-2}(\sigma, v)$. By the inductive hypothesis, $v_{m-2} \in \operatorname{occ!}(\sigma', v)$. Since $y \mapsto s[x/t] \in \sigma'$, and $v_{m-2} \in \mathit{vars}(s[x/t])$, $v_{m-2} \in \mathit{vars}(y\sigma')$. Thus, by Definition 3, $y \in \operatorname{occ!}(\sigma', v)$.

Conversely, we now prove that, for all m,

$$\operatorname{occ}_m(\sigma', v) \subseteq \operatorname{occ!}(\sigma, v).$$

The proof is again by induction on m. As in the previous case, $\operatorname{occ}_0(\sigma', v) = \operatorname{occ}_0(\sigma, v) = \{v\}$, so that the result holds for $m = 0$. Suppose then that $m > 0$ and that $v_m \in \mathrm{occ}_m(\sigma', v)$. By Definition 3, there exists $v_{m-1} \in \mathit{vars}(v_m\sigma')$ where $v_{m-1} \in \mathrm{occ}_{m-1}(\sigma', v)$. Hence, by the inductive hypothesis, $v_{m-1} \in \mathrm{occ!}(\sigma, v)$. If $v_m \in \mathrm{occ}(\sigma, v_{m-1})$, then, by Definition 3, $v_m \in \mathrm{occ!}(\sigma, v)$. On the other hand, if $v_{m-1} \notin \mathit{vars}(v_m\sigma)$, then $v_m = y$, $v_{m-1} \in \mathit{vars}(t)$ and $x \in \mathit{vars}(s)$. Thus, as $y \mapsto s \in \sigma$, $y \in \mathit{vars}(x\sigma)$. However, since $x \mapsto t \in \sigma$, $v_{m-1} \in \mathit{vars}(x\sigma)$ so that, by Definition 3, $x \in \mathrm{occ!}(\sigma, v)$. Thus, again by Definition 3, $y \in \mathrm{occ!}(\sigma, v)$.

Thus, if σ' is obtained from σ by a single $\mathcal{S}$-transformation, we have the required results: $v = v\sigma \iff v = v\sigma'$, $\mathrm{occ!}(\sigma, v) = \mathrm{occ!}(\sigma', v)$, and $\alpha(\sigma, U) = \alpha(\sigma', U)$.

Suppose now that there is a sequence $\sigma = \sigma_1, \ldots, \sigma_n = \sigma'$ such that, for $i = 2, \ldots, n$, σ_i is obtained from σ_{i-1} by a single $\mathcal{S}$-step. If $n = 1$, then $\sigma = \sigma'$. If $n > 1$, we have by the first part of the proof that, for each $i = 2, \ldots, n$, $v = v\sigma_{i-1} \iff v = v\sigma_i$, $\mathrm{occ!}(\sigma_{i-1}, v) = \mathrm{occ!}(\sigma_i, v)$, and $\alpha(\sigma_{i-1}, U) = \alpha(\sigma_i, U)$, and hence the required results. $\square$

Example 3. Consider again Example 2. Then

$$\begin{aligned}
\mathrm{occ}_1(\sigma_0, x_4) &= \{x_2, x_4\},\\
\mathrm{occ}_2(\sigma_0, x_4) &= \{x_1, x_2, x_4\},\\
\mathrm{occ}_3(\sigma_0, x_4) &= \{x_1, x_2, x_3, x_4\} = \mathrm{occ!}(\sigma_0, x_4),
\end{aligned}$$

and

$$\mathrm{occ}_1(\sigma_3, x_4) = \{x_1, x_2, x_3, x_4\} = \mathrm{occ!}(\sigma_3, x_4).$$

Thus, if $V = \{x_1, x_2, x_3, x_4\}$,

$$\alpha(\sigma_0, V) = \alpha(\sigma_3, V) = \bigl\{\{x_1, x_2, x_3, x_4\}\bigr\}.$$

4.2 Abstract Operations for Sharing Sets

We are concerned in this paper in establishing results for the abstract operation aunify which is defined for arbitrary sets of equations. However, by building the definition of aunify in three steps via the definitions of amgu (for sharing sets) and Amgu (for sharing domains) and stating corresponding results for each of them, we provide an outline for the overall method of proof for the aunify results. Details of all proofs are available in [7].

In order to define the abstract operation amgu we need some ancillary definitions.

Definition 5. (Auxiliary functions.) *The* closure under union *function (also called* star-union*),* $(\cdot)^\star \colon \mathit{SH} \to \mathit{SH}$*, is, for each* $\mathit{sh} \in \mathit{SH}$*,*

$$\mathit{sh}^\star \stackrel{\mathrm{def}}{=} \bigl\{\, S \in \mathit{SG} \bigm| \exists n \geq 1 \,.\, \exists T_1, \ldots, T_n \in \mathit{sh} \,.\, S = T_1 \cup \cdots \cup T_n \,\bigr\}.$$

For each $\mathit{sh} \in \mathit{SH}$ *and each* $T \in \wp_{\mathrm{f}}(\mathit{Vars})$*, the extraction of the* relevant component of sh with respect to T *is encoded by the function* $\mathrm{rel} \colon \wp_{\mathrm{f}}(\mathit{Vars}) \times \mathit{SH} \to \mathit{SH}$ *defined as*

$$\mathrm{rel}(T, \mathit{sh}) \stackrel{\mathrm{def}}{=} \{\, S \in \mathit{sh} \mid S \cap T \neq \varnothing \,\}.$$

For each $sh_1, sh_2 \in SH$*, the* binary union *function* $\mathrm{bin} \colon SH \times SH \to SH$ *is given by*

$$\mathrm{bin}(sh_1, sh_2) \stackrel{\mathrm{def}}{=} \{\, S_1 \cup S_2 \mid S_1 \in sh_1, S_2 \in sh_2 \,\}.$$

The function $\mathrm{proj} \colon SH \times \wp_{\mathrm{f}}(\mathit{Vars}) \to SH$ projects *an element of* SH *onto a set of variables of interest: if* $sh \in SH$ *and* $V \in \wp_{\mathrm{f}}(\mathit{Vars})$*, then*

$$\mathrm{proj}(sh, V) \stackrel{\mathrm{def}}{=} \{\, S \cap V \mid S \in sh, S \cap V \neq \varnothing \,\}.$$

Definition 6. (amgu.) *The function* amgu *captures the effects of a binding* $x \mapsto t$ *on an* SH *element. Let* x *be a variable and* t *a term. Let also* $sh \in SH$ *and*

$$A \stackrel{\mathrm{def}}{=} \mathrm{rel}\bigl(\{x\}, sh\bigr),$$
$$B \stackrel{\mathrm{def}}{=} \mathrm{rel}\bigl(\mathit{vars}(t), sh\bigr).$$

Then

$$\mathrm{amgu}(sh, x \mapsto t) \stackrel{\mathrm{def}}{=} \bigl(sh \setminus (A \cup B)\bigr) \cup \mathrm{bin}(A^\star, B^\star).$$

Then we have the following soundness result for amgu.

Lemma 4. *Let* $(sh, U) \in SS$ *and* $\{x \mapsto t\}, \sigma, \nu \in \mathit{Subst}$ *such that* ν *is a relevant unifier of* $\{x\sigma = t\sigma\}$ *and* $\mathit{vars}(x), \mathit{vars}(t), \mathit{vars}(\sigma) \subseteq U$*. Then*

$$\alpha(\sigma, U) \preceq_{SS} (sh, U) \implies \alpha(\nu \circ \sigma, U) \preceq_{SS} \bigl(\mathrm{amgu}(sh, x \mapsto t), U\bigr).$$

To prove this, observe that, by Lemma 2, if σ is not variable-idempotent, it can be transformed to a variable-idempotent substitution σ'. Hence, by Lemma 3, $\alpha(\sigma, U) = \alpha(\sigma', U)$. Therefore, the proof, which is given in [7], deals primarily with the case when σ is variable-idempotent.

Since a relevant unifier of e is a relevant unifier of any other set e' equivalent to e wrt to the equality theory T, this lemma shows that it is safe for the analyser to perform part or all of the concrete unification algorithm before computing amgu.

The following lemmas, proved in [7], show that amgu is commutative and idempotent.

Lemma 5. *Let* $sh \in SH$ *and* $\{x \mapsto r\} \in \mathit{Subst}$*. Then*

$$\mathrm{amgu}(sh, x \mapsto r) = \mathrm{amgu}\bigl(\mathrm{amgu}(sh, x \mapsto r), x \mapsto r\bigr).$$

Lemma 6. *Let* $sh \in SH$ *and* $\{x \mapsto r\}, \{y \mapsto t\} \in \mathit{Subst}$*. Then*

$$\mathrm{amgu}\bigl(\mathrm{amgu}(sh, x \mapsto r), y \mapsto t\bigr) = \mathrm{amgu}\bigl(\mathrm{amgu}(sh, y \mapsto t), x \mapsto r\bigr).$$

4.3 Abstract Operations for Sharing Domains

The definitions and results of Subsection 4.2 can be lifted to apply to sharing domains.

Definition 7. (Amgu.) *The operation* Amgu: $SS \times Subst \to SS$ *extends the* SS *description it takes as an argument, to the set of variables occurring in the binding it is given as the second argument. Then it applies* amgu*:*

$$\begin{aligned}
&\mathrm{Amgu}\bigl((sh, U), x \mapsto t\bigr) \\
&\quad \stackrel{\mathrm{def}}{=} \biggl(\mathrm{amgu}\Bigl(sh \cup \bigl\{\, \{u\} \bigm| u \in vars(x \mapsto t) \setminus U \,\bigr\}, x \mapsto t\Bigr), U \cup vars(x \mapsto t)\biggr).
\end{aligned}$$

The results for amgu can easily be extended to apply to Amgu.

Definition 8. (aunify.) *The function* aunify: $SS \times \mathrm{Eqs} \to SS$ *generalises* Amgu *to a set of equations* e*: If* $(sh, U) \in SS$*,* x *is a variable,* r *is a term,* $s = f(s_1, \ldots, s_n)$ *and* $t = f(t_1, \ldots, t_n)$ *are non-variable terms, and* $\overline{s} = \overline{t}$ *denote the set of equations* $\{s_1 = t_1, \ldots, s_n = t_n\}$*, then*

$$\mathrm{aunify}((sh, U), \varnothing) \stackrel{\mathrm{def}}{=} (sh, U),$$

if $e \in \wp_{\mathrm{f}}(\mathrm{Eqs})$ *is unifiable,*

$$\begin{aligned}
\mathrm{aunify}\bigl((sh, U), e \cup \{x = r\}\bigr) &\stackrel{\mathrm{def}}{=} \mathrm{aunify}\bigl(\mathrm{Amgu}(sh, U), x \mapsto r), e \setminus \{x = r\}\bigr), \\
\mathrm{aunify}\bigl((sh, U), e \cup \{s = x\}\bigr) &\stackrel{\mathrm{def}}{=} \mathrm{aunify}\bigl((sh, U), (e \setminus \{s = x\}) \cup \{x = s\}\bigr), \\
\mathrm{aunify}\bigl((sh, U), e \cup \{s = t\}\bigr) &\stackrel{\mathrm{def}}{=} \mathrm{aunify}\bigl((sh, U), (e \setminus \{s = t\}) \cup \overline{s} = \overline{t}\bigr),
\end{aligned}$$

and, if e *is not unifiable,*

$$\mathrm{aunify}((sh, U), e) \stackrel{\mathrm{def}}{=} \bot.$$

For the distinguished elements $\bot$ *and* $\top$ *of* SS

$$\mathrm{aunify}\bigl(\bot, e\bigr) \stackrel{\mathrm{def}}{=} \bot, \qquad \mathrm{aunify}\bigl(\top, e\bigr) \stackrel{\mathrm{def}}{=} \top.$$

As a consequence of this and the generalisation of Lemmas 4, 5 and 6 to Amgu, we have the following soundness, commutativity and idempotence results required for aunify to be sound and well-defined. As before, the proofs of these results are in [7].

Theorem 1. *Let* $(sh, U) \in SS$*,* $\sigma, \nu \in Subst$*, and* $e \in \wp_{\mathrm{f}}(\mathrm{Eqs})$ *be such that* $vars(\sigma) \subseteq U$ *and* ν *is a relevant unifier of* e*. Then*

$$\alpha(\sigma, U) \preceq_{SS} (sh, U) \implies \alpha(\nu \circ \sigma, U) \preceq_{SS} \mathrm{aunify}((sh, U), e).$$

Theorem 2. *Let* $(sh, U) \in SS$ *and* $e \in \wp_f(\text{Eqs})$. *Then*

$$\text{aunify}\big((sh, U), e\big) = \text{aunify}\Big(\text{aunify}\big((sh, U), e\big), e\Big).$$

Theorem 3. *Let* $(sh, U) \in SS$ *and* $e_1, e_2 \in \wp_f(\text{Eqs})$. *Then*

$$\text{aunify}\Big(\text{aunify}\big((sh, U), e_1\big), e_2\Big) = \text{aunify}\Big(\text{aunify}\big((sh, U), e_2\big), e_1\Big).$$

5 Discussion

The SS domain which was first defined by Langen [14] and published by Jacobs and Langen [8] is an important domain for sharing analysis. In this paper, we have provided a framework for analysing non-idempotent substitutions and presented results for soundness, idempotence and commutativity of aunify. In fact, most researchers concerned with analysing sharing and related properties using the SS domain, assume these properties hold. Why therefore are the results in this paper necessary? Let us consider each of the above properties one at a time.

5.1 Soundness

We have shown that, for any substitution σ over a set of variables U, the abstraction $\alpha(\sigma, U) = (sh, U)$ is unique (Lemma 3) and the aunify operation is sound (Theorem 1). Note that, in Theorem 1, there are no restrictions on σ; it can be non-idempotent, possibly including cyclic bindings (that is, bindings where the domain variable occurs in its co-domain). Thus this result is widely applicable.

Previous results on sharing have assumed that substitutions are idempotent. This is true if equality is syntactic identity and the implementation uses a unification algorithm based on that of Robinson [17] which includes the occur-check. With such algorithms, the resulting unifier is both unique and idempotent. Unfortunately, this is not what is implemented by most Prolog systems.

In particular, if the algorithm is as described in [11] and used in Prolog III [5], then the resulting unifier is in rational solved form. This algorithm does not generate idempotent or even variable-idempotent substitutions even when the occur-check would never have succeeded. However, it has been shown that the substitution obtained in this way uniquely defines a system of rational trees [5]. Thus our results show that its abstraction using α, as defined in this paper, is also unique and that *aunify* is sound.

Alternatively, if, as in most commercial Prolog systems, the unification algorithm is based on the Martelli-Montanari algorithm, but omits the occur check step, then the resulting substitution may not be idempotent. Consider the following example.

Suppose we are given as input the equation $p(z, f(x, y)) = p(f(z, y), z)$ with an initial substitution that is empty. We apply the steps in Martelli-Montanari procedure but without the occur-check:

	equations	substitution
1	$p(z, f(x,y)) = p(f(z,y), z)$	$\varnothing$
2	$z = f(z,y), f(x,y) = z$	$\varnothing$
3	$f(x,y) = f(z,y)$	$\{z \mapsto f(z,y)\}$
4	$x = z, y = y$	$\{z \mapsto f(z,y)\}$
5	$y = y$	$\{z \mapsto f(z,y), x \mapsto z\}$
6	$\varnothing$	$\{z \mapsto f(z,y), x \mapsto z\}$

Note that we have used three kinds of steps here. In lines 1 and 3, neither argument of the selected equation is a variable. In this case, the outer non-variable symbols (when, as in this example, they are the same) are removed and new equations are formed between the corresponding arguments. In lines 2 and 4, the selected equation has the form $v = t$, where v is a variable and t is not identical to v, then every occurrence of v is replaced by t in all the remaining equations and the range of the substitution. $v \mapsto t$ is then added to the substitution. In line 5, the identity is removed.

Let $\sigma = \{z \mapsto f(z,y), x \mapsto z\}$, be the computed substitution. Then, we have

$$\begin{aligned} vars(x\sigma) &= vars(z) = \{z\}, \\ vars(x\sigma^2) &= vars(f(z,y)) = \{y, z\}. \end{aligned}$$

Hence σ is not variable-idempotent.

We conjecture that the resulting substitution is still unique (up to variable renaming). In this case our results can be applied so that its abstraction using α, as defined in this paper, is also unique and *aunify* is sound.

5.2 Idempotence

Definition 8 defines aunify inductively over a set of equations, so that it is important for this definition that aunify is both idempotent and commutative.

The only previous result concerning the idempotence of aunify is given in thesis of Langen [14, Theorem 32]. However, the definition of aunify in [14] includes the renaming and projection operations and, in this case, only a weak form of idempotence holds. In fact, for the basic aunify operation as defined here and without projection and renaming, idempotence has never before been proven.

5.3 Commutativity

In the thesis of Langen the "proof" of commutativity of amguhas a number of omissions and errors [14, Lemma 30]. We highlight here, one error which we were unable to correct in the context of the given proof.

To make it easier to compare, we adapt our notation and, define amge only in the case that a is a variable:

$$\mathrm{amge}(a, b, sh) \stackrel{\mathrm{def}}{=} \mathrm{amgu}(sh, a \mapsto b).$$

To prove the lemma, it has to show that:

$$\text{amge}(a_2, b_2\, \text{amge}(a_1, b_1, sh)) = \text{amge}(a_1, b_1, \text{amge}(a_2, b_2, sh)).$$

holds when a_1 and a_2 are variables. This corresponds to "the second base case" of the proof. We use Langen's terminology:

- A set of variables X is at a term t iff $\text{var}(t) \cap X \neq \varnothing$.
- A set of variables X is at i iff X is at a_i or b_i.
- A union $X \cup_i Y$ is of Type i iff X is at a_i and Y is at b_i.

Let lhs $\stackrel{\text{def}}{=} \text{amge}(a_2, b_2, \text{amge}(a_1, b_1, S))$, and rhs $\stackrel{\text{def}}{=} \text{amge}(a_1, b_1, \text{amge}(a_2, b_2, S))$. Let also $Z \in \text{lhs}$ and $T \stackrel{\text{def}}{=} \text{aunify}(a_1, b_1, S)$. Consider the case when

$$\begin{aligned} Z &= X \cup_2 Y \text{ where } X \in \text{rel}(a_2, T), Y \in \text{rel}(b_2, T), \\ X &= U \cup_1 V \text{ where } U \in \text{rel}(a_1, sh), V \in \text{rel}(b_1, sh) \end{aligned}$$

and $U \cap (vars(a_2) \cup vars(b_2)) = \varnothing$ (that is, U is not at 2). Then the following quote [14, page 53, line 23] applies:

> In this case $(U \cup_1 V) \cup_2 Y = U \cup_1 (V \cup_2 Y)$. By the inductive assumption $V \cup_2 Y$ is in the rhs and therefore so is Z.

We give a counter-example to the statement "$V \cup_2 Y$ is in the rhs".

Suppose a_1, b_1, a_2, b_2 are variables. We let each of a_1, b_1, a_2, b_2 denote both the actual variable and the singleton set containing that variable. Suppose $sh = \{a_1, b_1 a_2, b_2\}$. Then, from the definition of amge,

$$\text{lhs} = \{a_1 b_1 a_2 b_2\}, \qquad \text{rhs} = \{a_1 b_1 a_2 b_2\}, \qquad T = \{a_1 b_1 a_2, b_2\}.$$

Let $Z = a_1 b_1 a_2 b_2$, $X = a_1 b_1 a_2$, $Y = b_2$, $U = a_1$, $V = b_1 a_2$. All the above conditions. However $V \cup_2 Y = b_1 a_2 b_2$ and this is not in $\{a_1 b_1 a_2 b_2\}$.

References

1. R. Bagnara, P. M. Hill, and E. Zaffanella. Set-sharing is redundant for pair-sharing. In P. Van Hentenryck, editor, *Static Analysis: Proceedings of the 4th International Symposium*, volume 1302 of *Lecture Notes in Computer Science*, pages 53–67, Paris, France, 1997. Springer-Verlag, Berlin.
2. M. Bruynooghe and M. Codish. Freeness, sharing, linearity and correctness — All at once. In P. Cousot, M. Falaschi, G. Filé, and A. Rauzy, editors, *Static Analysis, Proceedings of the Third International Workshop*, volume 724 of *Lecture Notes in Computer Science*, pages 153–164, Padova, Italy, 1993. Springer-Verlag, Berlin. An extended version is available as Technical Report CW 179, Department of Computer Science, K.U. Leuven, September 1993.
3. K. L. Clark. Negation as failure. In H. Gallaire and J. Minker, editors, *Logic and Databases*, pages 293–322, Toulouse, France, 1978. Plenum Press.

4. M. Codish, D. Dams, G. Filé, and M. Bruynooghe. Freeness analysis for logic programs-and correctness? In D. S. Warren, editor, *Logic Programming: Proceedings of the Tenth International Conference on Logic Programming*, MIT Press Series in Logic Programming, pages 116–131, Budapest, Hungary, 1993. The MIT Press. An extended version is available as Technical Report CW 161, Department of Computer Science, K.U. Leuven, December 1992.
5. A. Colmerauer. Prolog and Infinite Trees. In K. L. Clark and S. Å. Tärnlund, editors, *Logic Programming, APIC Studies in Data Processing*, volume 16, pages 231–251. Academic Press, New York, 1982.
6. A. Colmerauer. Equations and inequations on finite and infinite trees. In *Proceedings of the International Conference on Fifth Generation Computer Systems (FGCS'84)*, pages 85–99, Tokyo, Japan, 1984. ICOT.
7. P. M. Hill, R. Bagnara, and E. Zaffanella. The correctness of set-sharing. Technical Report 98.03, School of Computer Studies, University of Leeds, 1998.
8. D. Jacobs and A. Langen. Accurate and efficient approximation of variable aliasing in logic programs. In E. L. Lusk and R. A. Overbeek, editors, *Logic Programming: Proceedings of the North American Conference*, MIT Press Series in Logic Programming, pages 154–165, Cleveland, Ohio, USA, 1989. The MIT Press.
9. D. Jacobs and A. Langen. Static analysis of logic programs for independent AND parallelism. *Journal of Logic Programming*, 13(2&3):291–314, 1992.
10. J. Jaffar, J-L. Lassez, and M. J. Maher. Prolog-II as an instance of the logic programming scheme. In M. Wirsing, editor, *Formal Descriptions of Programming Concepts III*, pages 275–299. North Holland, 1987.
11. T. Keisu. *Tree Constraints.* PhD thesis, The Royal Institute of Technology, Stockholm, Sweden, May 1994. Also available in the SICS Dissertation Series: SICS/D–16–SE.
12. A. King. A synergistic analysis for sharing and groundness which traces linearity. In D. Sannella, editor, *Proceedings of the Fifth European Symposium on Programming*, volume 788 of *Lecture Notes in Computer Science*, pages 363–378, Edinburgh, UK, 1994. Springer-Verlag, Berlin.
13. A. King and P. Soper. Depth-k sharing and freeness. In P. Van Hentenryck, editor, *Logic Programming: Proceedings of the Eleventh International Conference on Logic Programming*, MIT Press Series in Logic Programming, pages 553–568, Santa Margherita Ligure, Italy, 1994. The MIT Press.
14. A. Langen. *Static Analysis for Independent And-Parallelism in Logic Programs.* PhD thesis, Computer Science Department, University of Southern California, 1990. Printed as Report TR 91-05.
15. M. J. Maher. Complete axiomatizations of the algebras of finite, rational and infinite trees. In *Proceedings, Third Annual Symposium on Logic in Computer Science*, pages 348–357, Edinburgh, Scotland, 1988. IEEE Computer Society.
16. K. Muthukumar and M. Hermenegildo. Compile-time derivation of variable dependency using abstract interpretation. *Journal of Logic Programming*, 13(2&3):315–347, 1992.
17. J. A. Robinson. A machine-oriented logic based on the resolution principle. *Journal of the ACM*, 12(1):23–41, 1965.