A Mode Analysis of Logic Programs by Abstract Interpretation

Lunjin Lu

Department of Educational Information Technology
East China Normal University
Shanghai 200062, China

Abstract. This paper presents a mode analysis of logic programs by means of abstract interpretation. Mode information is inferred together with sharing and aliasing information. Aliasing information is used to improve accuracy of analysis whilst sharing information is used to ensure the safety of analysis.
Keywords: Mode, Sharing, Aliasing, Abstract Interpretation, Abstract Unification

1 Introduction

In logic programming, information about the instantiation modes of variables helps the compiler to generate efficient codes by using different specialised unification algorithms for terms in different instantiation modes [19]. This has led to much intensive research into automatic inference of mode information from program texts. Especially, there have been many efforts in inferring mode information by abstract interpretation [1, 2, 7, 8, 15, 16, 17, 18].

This paper presents a mode analysis of logic programs by means of an abstract domain for descriptions of sets of substitutions together with its associated abstract unification operator and least upper bound operator as required by the framework in [14]. The novelty of our work is to make uses of aliasing information to improve the accuracy of analysis. By aliasing, we refer to the situation where two variables are instantiated to syntactically identical terms. The abstract domain is the direct product of three sub-domains for modes, pair sharing and aliasing respectively. Variables of interest are described by mode descriptions which will be described later. The pair sharing among these variables is used to ensure the correctness of analysis since mode descriptions are not closed under instantiation. Aliasing information is used to improve the accuracy of analysis. While an improvement on the accuracy of analysis is expected due to the introduction of new component information [11, 16], we exploit aliasing information to improve the accuracy of analysis in a number of ways.

The remainder of this paper is organised as follows. Section 2 introduces some logic programming concepts and notations used in this paper, and briefly recalls the abstract intepretation framework within which we present the mode analysis. The abstract unification operator and the abstract least upper bound operator are described. Section 3 describes the abstract domain for mode analysis and section 4 defines some ancillary operators used by the abstract unification operator. Section 5 gives the abstract unification operator and states its correctness. Section 6 concludes the paper with a brief discussion. Proofs for lemmas and theorems are omitted due to space limit.

2 Preliminaries

We assume that the reader is familiar with terminology of logic programming [13]. Let P be the program to be analysed, Σ a set of *function symbols*, Π a set of *predicate symbols*, Vars a denumerable set of variables. Term denotes the set of *terms* that can be constructed from Σ and Vars. Atom denotes the set of *atoms* constructible from Π, Σ and Vars.

$dom(\theta)$ denotes the domain of a *substitution* θ. $\mathbf{V}_E$ denotes the set of variables in an expression E. $A \equiv B$ denotes that A is syntactically identical to B. Let f be a partial function from and D be a set. Then $f \upharpoonright D$ denotes the *restriction* of f to D. The function composition operator $\circ$ binds stronger than $\upharpoonright$.

An *equation* is a formula of the form $l = r$ where either $l, r \in \mathsf{Term}$ or $l, r \in \mathsf{Atom}$. The set of equations is denoted as Eq. Let $E \in \wp(\mathsf{Eq})$. E is in *solved form* if, for each $(l = r) \in E$, l is a variable that does not occur on the rightside of any equation in E. For a set of equations $E \in \wp(\mathsf{Eq})$, $mgu \in \wp(\mathsf{Eq}) \mapsto Sub \cup \{fail\}$ returns either a most general unifier for E if E is unifiable or $fail$ otherwise. $mgu(\{l = r\})$ is sometimes written as $mgu(l, r)$. $eq(\theta)$ denotes the set of equations in solved form corresponding to a substitution θ. $eq(fail) \stackrel{def}{=} fail$. For any $\mathsf{V} \subseteq \mathsf{Vars}$, $\mathsf{Term}(\mathsf{V}), \mathsf{Atom}(\mathsf{V})$ and $\mathsf{Eq}(\mathsf{V})$ denote respectively the subsets of Term, Atom, and Eq involving only variables in V.

The abstract interpretation framework in [14] abstracts a collecting semantics for normal logic programs which is constructed from two operators on $< \wp(Sub), \subseteq >$. One is the set union operator $\cup : \wp(Sub) \times \wp(Sub) \mapsto \wp(Sub)$. The other is $UNIFY : \mathsf{Atom}(\mathbf{V}_\mathsf{P}) \times \wp(Sub) \times \mathsf{Atom}(\mathbf{V}_\mathsf{P}) \times \wp(Sub) \mapsto \wp(Sub)$ defined as follows.

$$UNIFY(A, \Theta, B, \Omega) \stackrel{def}{=} \{unify(A, \theta, B, \omega) \neq fail \mid \theta \in \Theta \wedge \omega \in \Omega\}$$

where

$$unify(A, \theta, B, \omega) \stackrel{def}{=} \begin{cases} \text{let } \rho \textit{ be a renaming such that } \mathbf{V}_{\rho\circ\theta(A)} \cap \mathbf{V}_{\omega(B)} = \emptyset \\ \text{in } mgu(\rho \circ \theta(A), \omega(B)) \circ \omega \end{cases}$$

with $fail \circ \theta \stackrel{def}{=} \theta \circ fail \stackrel{def}{=} fail$. Let $< \mathsf{ASub}, \sqsubseteq >$ denote the domain for abstract substitutions describing values of variables in $\mathbf{V}_\mathsf{P}$. The idea of using an abstract substitution in ASub to describe a set of substitutions is embodied by a monotonic function $\gamma : \mathsf{ASub} \mapsto \wp(Sub)$, called a *concretisation* function. The framework requires that, $< \mathsf{ASub}, \sqsubseteq >$ is a complete lattice. For a particular analysis, $\cup$ is approximated by the least upper bound $\sqcup : \mathsf{ASub} \times \mathsf{ASub} \mapsto \mathsf{ASub}$ on ASub and $UNIFY$ by $AUNIFY : \mathsf{Atom}(\mathbf{V}_\mathsf{P}) \times \mathsf{ASub} \times \mathsf{Atom}(\mathbf{V}_\mathsf{P}) \times \mathsf{ASub} \mapsto \mathsf{ASub}$. The framework requires that, for any $A, B \in \mathsf{Atom}(\mathbf{V}_\mathsf{P})$, any $\theta^\flat, \sigma^\flat \in \mathsf{ASub}$, $UNIFY(A, \gamma(\theta^\flat), B, \gamma(\sigma^\flat)) \subseteq \gamma \circ AUNIFY(A, \theta^\flat, B, \sigma^\flat)$.

3 Abstract Domain

An abstract substitution $\theta^\flat$ is a triple $< \mathcal{I}, \mathcal{S}, \mathcal{A} >$ where $\mathcal{I}$ maps each variable in $\mathbf{V_P}$ to a mode description as described below. $\mathcal{S}$, called a pair sharing relation [4], describes possible sharing [10] among variables in $\mathbf{V_P}$ and possible non-linearity of variables in $\mathbf{V_P}$. $\mathcal{A}$ describes definite aliasing among variables in $\mathbf{V_P}$.

A term is in mode ⓕ if it is a variable. It is in mode ⓖ if it is a ground term. It is in mode ⓞ if it is neither a variable nor a ground term. Let $\triangle = \{ⓕ, ⓖ, ⓞ\}$. The mode of a term t is denoted by $m(t)$. When a set of substitutions is approximated, a variable X is given a subset of $\triangle$ as its mode description. For instance, if X is given a mode description $\{ⓕ, ⓖ\}$ then X is instantiated to either a ground term or a variable. $< \wp(\triangle), \subseteq >$ is a complete lattice.

An instantiation state $\mathcal{I}$ on $\mathbf{V_P}$ is a function from $\mathbf{V_P}$ into $\wp(\triangle)$. A substitution θ satisfies $\mathcal{I}$ if $\forall X \in \mathbf{V_P}.(m \circ \theta(X) \in \mathcal{I}(X))$. Let $\mathbf{IS}$ be the set of instantiation states on $\mathbf{V_P}$. $\mathbf{IS}$, ordered by the point-wise extension of $\subseteq$, is a complete lattice.

Two distinct variables X, Y share under a substitution θ if $\mathbf{V}_{\theta(X)} \cap \mathbf{V}_{\theta(Y)} \neq \emptyset$. A variable X is non-linear under a substitution θ if there is at least one variable which occurs more than once in $\theta(X)$. Let $X \in \mathsf{Vars}$ and $t \in \mathsf{Term}$. $repeat(X, t) = true$ iff X occurs more than once in t. The multiplicity function $\chi \in \mathsf{Term} \mapsto \{0, 1, 2\}$ is defined as follows [2].

$$\chi(t) \stackrel{def}{=} \text{if } \mathbf{V}_t = \emptyset \text{ then } 0 \text{ elseif } \exists X.repeat(X, t) \text{ then } 2 \text{ else } 1$$

The possible sharing among variables in $\mathbf{V_P}$ and possible non-linearity of variables in $\mathbf{V_P}$ under a substitution θ are represented as a symmetric relation $\mathcal{S}$ on $\mathbf{V_P}$. A substitution θ satisfies $\mathcal{S}$ if, for any $X, Y \in \mathbf{V_P}$,

$$(X \not\equiv Y \wedge (X, Y) \notin \mathcal{S} \rightarrow \mathbf{V}_{\theta(X)} \cap \mathbf{V}_{\theta(Y)} = \emptyset) \wedge ((X, X) \notin \mathcal{S} \rightarrow \chi \circ \theta(X) \neq 2)$$

Let $\mathbf{SR}$ be the set of symmetric relations on $\mathbf{V_P}$. $< \mathbf{SR}, \subseteq >$ is a complete lattice.

Two variables X and Y are aliased under a substitution θ if $\theta(X) \equiv \theta(Y)$. The aliasing among variables in $\mathbf{V_P}$ is represented by an equivalence relation $\mathcal{A}$ on $\mathbf{V_P}$. A substitution θ is said to satisfy $\mathcal{A}$ if $\forall X, Y \in \mathbf{V_P}.((X, Y) \in \mathcal{A} \rightarrow \theta(X) \equiv \theta(Y))$. Let $\mathbf{ER}$ be the set of equivalence relations on $\mathbf{V_P}$. $< \mathbf{ER}, \supseteq >$ is a complete lattice with the infimum $\mathbf{V_P}^2$, the supremum $\{< X, X > \mid X \in \mathbf{V_P}\}$, the least upper bound $\cap$ and the greatest lower bound $\lambda \mathcal{A}_1, \mathcal{A}_2 \in \mathbf{ER}.(\mathcal{A}_1 \cup \mathcal{A}_2)^*$ where $(\cdot)^*$ is the transitive closure operator.

$\mathsf{ASub} \stackrel{def}{=} \mathbf{IS} \times \mathbf{SR} \times \mathbf{ER}$ and $\sqsubseteq$ is defined as the component-wise order on ASub. $< \mathsf{ASub}, \sqsubseteq >$ is a complete lattice with $\bot = < \lambda X \in \mathbf{V_P}.\emptyset, \emptyset, \mathbf{V_P}^2 >$ as its infimum. Let $\theta^\flat = < \mathcal{I}, \mathcal{S}, \mathcal{A} > \in \mathsf{ASub}$. A substitution θ is said to satisfy $\theta^\flat$ if it satisfies $\mathcal{I}$, $\mathcal{S}$ and $\mathcal{A}$ simultaneously. Define $\mathbf{M}_{\theta^\flat} \stackrel{def}{=} \mathcal{I}$, $\mathbf{S}_{\theta^\flat} \stackrel{def}{=} \mathcal{S}$ and $\mathbf{A}_{\theta^\flat} \stackrel{def}{=} \mathcal{A}$. $\gamma : \mathsf{ASub} \mapsto \wp(Sub)$ defined below is monotonic.

$$\gamma(\theta^\flat) \stackrel{def}{=} \{\theta \in Sub \mid \theta \ satisfies \ \theta^\flat\}$$

The abstract domain has been constructed by the direct product approach [4]. An abstract substitution in the abstract domain may contain redundance that is dealt with in section 5. Another approach to combine abstract domains is the reduced product approach [5] whereby an abstract substitution does not contain redundance. However, the abstract operators on component abstract domains have to be redefined in the reduced product approach while they can be directly used in the direct product approach.

4 Ancillary operators

This section defines some ancillary operators used in the abstract unification operator.

Abstract substitutions in ASub are predicates over $\mathsf{V_P}$. Let Ψ be a fixed renaming substitution such that $\Psi(\mathsf{V_P}) \cap \mathsf{V_P} = \emptyset$ and $\widetilde{\mathsf{V_P}} \stackrel{def}{=} \mathsf{V_P} \cup \Psi(\mathsf{V_P})$. Since many operators used to define $AUNIFY$ operate on predicates over $\widetilde{\mathsf{V_P}}$, we first construct an abstract domain $< \widetilde{\mathsf{ASub}}, \widetilde{\sqsubseteq} >$ to describe the values of variables in $\widetilde{\mathsf{V_P}}$. From the definitions of $\mathbf{IS}, \mathbf{SR}$ and $\mathbf{ER}$, $\widetilde{\mathbf{IS}}, \widetilde{\mathbf{SR}}$ and $\widetilde{\mathbf{ER}}$ are defined by substituting $\widetilde{\mathsf{V_P}}$ for $\mathsf{V_P}$, and $\widetilde{\mathbf{IS}}, \widetilde{\mathbf{SR}}, \widetilde{\mathbf{ER}}$ for $\mathbf{IS}, \mathbf{SR}, \mathbf{ER}$ respectively. $< \widetilde{\mathsf{ASub}}, \widetilde{\sqsubseteq} >$ is defined from $\widetilde{\mathbf{IS}}, \widetilde{\mathbf{SR}}$ and $\widetilde{\mathbf{ER}}$ just as $< \mathsf{ASub}, \sqsubseteq >$ from $\mathbf{IS}, \mathbf{SR}$ and $\mathbf{ER}$. $\widetilde{\bot}$ is defined from $\widetilde{\mathsf{V_P}}$ in the same way as $\bot$ from $\mathsf{V_P}$. $\widetilde{\gamma} : \widetilde{\mathsf{ASub}} \mapsto \wp(Sub)$ defined in the following is monotonic.

$$\widetilde{\gamma}(\theta^\flat) \stackrel{def}{=} \{\theta \in \ Sub \mid \theta \ satisfies \ \theta^\flat\}$$

$\uplus : \mathsf{ASub} \times \mathsf{ASub} \mapsto \widetilde{\mathsf{ASub}}$ renames its first argument from ASub, and combines the result with its second argument also from ASub to form an abstract substitution in $\widetilde{\mathsf{ASub}}$. $rest : \widetilde{\mathsf{ASub}} \mapsto \mathsf{ASub}$ restricts a predicate over $\widetilde{\mathsf{V_P}}$ into a predicate over $\mathsf{V_P}$.

$$\theta^\flat \uplus \sigma^\flat \stackrel{def}{=} < \Psi(\mathbf{M}_{\theta^\flat}) \cup \mathbf{M}_{\sigma^\flat}, \Psi(\mathbf{S}_{\theta^\flat}) \cup \mathbf{S}_{\sigma^\flat}, \Psi(\mathbf{A}_{\theta^\flat}) \cup \mathbf{A}_{\sigma} > \tag{1}$$

$$rest(\sigma^\flat) \stackrel{def}{=} < \mathbf{M}_{\sigma^\flat} \restriction \mathsf{V_P}, \mathbf{S}_{\sigma^\flat} \cap \mathsf{V_P}^2, \mathbf{A}_{\sigma^\flat} \cap \mathsf{V_P}^2 > \tag{2}$$

Define $\overline{m} : \mathsf{Term}(\widetilde{\mathsf{V_P}}) \times \widetilde{\mathbf{IS}} \mapsto \wp(\triangle)$ as below. We have that $m \circ \sigma(t) \in \overline{m}(t, \mathcal{I})$ for any $t \in \mathsf{Term}(\widetilde{\mathsf{V_P}})$, any $\mathcal{I} \in \widetilde{\mathbf{IS}}$ and any σ satisfying $\mathcal{I}$.

$$\overline{m}(t, \mathcal{I}) \stackrel{def}{=} \begin{cases} \text{if } t \in \widetilde{\mathsf{V_P}} \text{ then } \mathcal{I}(t) \text{ elseif } \exists X \in \mathsf{V}_t.\mathcal{I}(X) = \emptyset \text{ then } \emptyset \\ \text{elseif } \forall X \in \mathsf{V}_t.\mathcal{I}(X) = \{ⓖ\} \text{ then } \{ⓖ\} \\ \text{elseif } \forall X \in \mathsf{V}_t.(ⓖ \in \mathcal{I}(X)) \text{ then } \{ⓖ, ⓞ\} \text{ else } \{ⓞ\} \end{cases}$$

Fig. 1 defines $\otimes \in \wp(\triangle) \times \wp(\triangle) \mapsto \wp(\triangle)$ which approximates the mode description of the most general common instance of two terms given their mode descriptions. The lemma 1 is a justification of this definition.

$\otimes$	$\emptyset$	$\{\textcircled{f}\}$	$\{\textcircled{g}\}$	$\{\textcircled{o}\}$	$\{\textcircled{f},\textcircled{g}\}$	$\{\textcircled{f},\textcircled{o}\}$	$\{\textcircled{g},\textcircled{o}\}$	$\triangle$
$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
$\{\textcircled{f}\}$	$\emptyset$	$\{\textcircled{f}\}$	$\{\textcircled{g}\}$	$\{\textcircled{o}\}$	$\{\textcircled{f},\textcircled{g}\}$	$\{\textcircled{f},\textcircled{o}\}$	$\{\textcircled{g},\textcircled{o}\}$	$\triangle$
$\{\textcircled{g}\}$	$\emptyset$	$\{\textcircled{g}\}$	$\{\textcircled{g}\}$	$\{\textcircled{g}\}$	$\{\textcircled{g}\}$	$\{\textcircled{g}\}$	$\{\textcircled{g}\}$	$\{\textcircled{g}\}$
$\{\textcircled{o}\}$	$\emptyset$	$\{\textcircled{o}\}$	$\{\textcircled{g}\}$	$\{\textcircled{g},\textcircled{o}\}$	$\{\textcircled{g},\textcircled{o}\}$	$\{\textcircled{g},\textcircled{o}\}$	$\{\textcircled{g},\textcircled{o}\}$	$\{\textcircled{g},\textcircled{o}\}$
$\{\textcircled{f},\textcircled{g}\}$	$\emptyset$	$\{\textcircled{f},\textcircled{g}\}$	$\{\textcircled{g}\}$	$\{\textcircled{g},\textcircled{o}\}$	$\{\textcircled{f},\textcircled{g}\}$	$\triangle$	$\{\textcircled{g},\textcircled{o}\}$	$\triangle$
$\{\textcircled{f},\textcircled{o}\}$	$\emptyset$	$\{\textcircled{f},\textcircled{o}\}$	$\{\textcircled{g}\}$	$\{\textcircled{g},\textcircled{o}\}$	$\triangle$	$\triangle$	$\{\textcircled{g},\textcircled{o}\}$	$\triangle$
$\{\textcircled{g},\textcircled{o}\}$	$\emptyset$	$\{\textcircled{g},\textcircled{o}\}$	$\{\textcircled{g}\}$	$\{\textcircled{g},\textcircled{o}\}$	$\{\textcircled{g},\textcircled{o}\}$	$\{\textcircled{g},\textcircled{o}\}$	$\{\textcircled{g},\textcircled{o}\}$	$\{\textcircled{g},\textcircled{o}\}$
$\triangle$	$\emptyset$	$\triangle$	$\{\textcircled{g}\}$	$\{\textcircled{g},\textcircled{o}\}$	$\triangle$	$\triangle$	$\{\textcircled{g},\textcircled{o}\}$	$\triangle$

Fig. 1. Operator $\otimes$.

Lemma 1. *Let* $t_1, t_2 \in \mathsf{Term}$. *If* $\eta = mgu(t_1, t_2) \neq fail$ *then*

$$m \circ \eta(t_1) = m \circ \eta(t_2) \in \{m(t_1)\} \otimes \{m(t_2)\}$$

∎

Let $X \in \mathsf{Vars}$,$t \in \mathsf{Term}$ and $\sigma \in Sub$. The unification of $\sigma(X)$ and $\sigma(t)$ may not only change the instantiation mode of a variable $Y \in \{X\} \cup \mathbf{V}_t$ but also change the instantiation mode of a variable $Z \notin \{X\} \cup \mathbf{V}_t$ if Z shares with any variable $W \in \{X\} \cup \mathbf{V}_t$ under σ. The following lemmas 2 and 3 identify those variables whose instantiation modes are not affected by the unification of $\sigma(X)$ and $\sigma(t)$.

Lemma 2. *Let* $X, Y \in \mathsf{Vars}$, $t \in \mathsf{Term}$ *and* $\sigma \in Sub$ *such that* Y *does not share with any variable in* $\{X\} \cup \mathbf{V}_t$ *under* σ. *If* $\eta = mgu(\sigma(X), \sigma(t)) \neq fail$ *then* $\eta \circ \sigma(Y) = \sigma(Y)$. ∎

Lemma 3. *Let* $X, Y \in \mathsf{Vars}$, $t \in \mathsf{Term}$, $\sigma \in Sub$, *and* $\eta = mgu(\sigma(X), \sigma(t)) \neq fail$. $m \circ \eta \circ \sigma(Y) = m \circ \sigma(Y)$ *if either (1)* $m \circ \sigma(X) = \textcircled{g} \wedge m \circ \sigma(t) = \textcircled{g}$, *or (2)* $m \circ \sigma(X) = \textcircled{f} \wedge m \circ \sigma(t) = \textcircled{f}$, *or (3)* $m \circ \sigma(X) = \textcircled{f}$, $Y \neq X$ *and* Y *does not share with* X *under* σ, *or (4)* $m \circ \sigma(t) = \textcircled{f}$ *and* Y *does not share with any* $U \in \mathbf{V}_t$. ∎

According to lemmas 2-3, the instantiation mode of a variable $Z \notin \mathbf{V}_e$ is affected by the unification of $\{\sigma(e)\}$ only if Z shares with some varaible in a subset of $\mathbf{V}_e$ under σ. *aff* : $\mathsf{Eq}(\widetilde{\mathbf{V}_{\mathsf{P}}}) \times \widetilde{\mathbf{IS}} \mapsto \wp(\widetilde{\mathbf{V}_{\mathsf{P}}})$ defined below computes that subset of $\mathbf{V}_e$.

$$aff(X = t, \mathcal{I}) \stackrel{def}{=}$$

$$\begin{cases} \text{if } \mathcal{I}(X) = \emptyset \vee \overline{m}(t, \mathcal{I}) = \emptyset \vee \mathcal{I}(X) = \overline{m}(t, \mathcal{I}) = \{\textcircled{g}\} \vee \mathcal{I}(X) = \overline{m}(t, \mathcal{I}) = \{\textcircled{f}\} \\ \text{then } \emptyset \text{ elseif } \mathcal{I}(X) = \{\textcircled{f}\} \text{ then } \{X\} \text{ elseif } \overline{m}(t, \mathcal{I}) = \{\textcircled{f}\} \text{ then } \mathbf{V}_t \text{ else } \{X\} \cup \mathbf{V}_t \end{cases}$$

Let $\mathcal{I} \in \widetilde{\mathbf{IS}}$, $e \equiv (X = t) \in \mathsf{Eq}(\widetilde{\mathbf{V}_{\mathsf{P}}})$, $\sigma \in Sub$ satisfying $\mathcal{I}$. If $\eta = mgu(\{\sigma(e)\}) \neq$ *fail* then $est : \mathsf{Eq}(\widetilde{\mathbf{V}_{\mathsf{P}}}) \times \widetilde{\mathbf{IS}} \mapsto \widetilde{\mathbf{IS}}$ defined below approximately evaluates the instantiation mode of $\eta \circ \sigma(Y)$ for each variable U in $\mathbf{V}_e$. Each variable outside $\mathbf{V}_e$ is given the mode description $\triangle$. The lemma 4 justifies this definition.

$$est(e,\mathcal{I}) \stackrel{def}{=} \lambda U. \begin{cases} \mathsf{if}\ U \notin \mathbf{V}_e\ \mathsf{then}\ \triangle\ \mathsf{elseif}\ t \equiv U \vee U \equiv X\ \mathsf{then}\ \mathcal{I}(X) \otimes \overline{m}(t,\mathcal{I}) \\ \mathsf{elseif}\ \mathcal{I}(X) = \emptyset \vee \overline{m}(t,\mathcal{I}) = \emptyset\ \mathsf{then}\ \emptyset \\ \mathsf{elseif}\ \mathcal{I}(X) = \{ⓖ\} \vee \overline{m}(t,\mathcal{I}) = \{ⓖ\}\ \mathsf{then}\ \{ⓖ\}\ \mathsf{else}\ \triangle \otimes \mathcal{I}(U) \end{cases} \tag{3}$$

Lemma 4. *Let $X, U \in \mathsf{Vars}$, $t \in \mathsf{Term}$ such that $U \in \mathbf{V}_t$, $\sigma \in Sub$ and $\eta = mgu(\sigma(X), \sigma(t)) \neq fail$. Then (1) If $t = U$ is a variable then $m \circ \eta \circ \sigma(U) \in \{m \circ \sigma(X)\} \otimes \{m \circ \sigma(t)\}$; and (2) If $t \neq U \wedge (m \circ \sigma(X) = ⓖ \vee m \circ \sigma(t) = ⓖ)$ then $m \circ \eta \circ \sigma(U) = ⓖ$; and (3) If $t \neq U \wedge m \circ \sigma(X) \neq ⓖ \wedge m \circ \sigma(t) \neq ⓖ$ then $m \circ \eta \circ \sigma(U) \in \triangle \otimes \{m \circ \sigma(U)\}$.* ∎

If two different variables X and Y are aliased together by a substitution σ and X and Y do not share under σ then both X and Y are ground under σ. Define $grd : \widetilde{\mathbf{V}_{\mathsf{P}}} \times \widetilde{\mathsf{ASub}} \mapsto \{true, false\}$ as follows.

$$grd(X, \sigma^\flat) \stackrel{def}{=} \mathbf{M}_{\sigma^\flat}(X) = \{ⓖ\} \vee \exists Y \in \widetilde{\mathbf{V}_{\mathsf{P}}}.(X \not\equiv Y \wedge (X,Y) \in \mathbf{A}_{\sigma^\flat} \wedge (X,Y) \notin \mathbf{S}_{\sigma^\flat})$$

For any $\sigma \in \widetilde{\gamma}(\sigma^\flat)$ and any $X \in \widetilde{\mathbf{V}_{\mathsf{P}}}$, if $grd(X, \sigma^\flat) = true$ then $\sigma(X)$ is ground.

$red : \widetilde{\mathsf{ASub}} \mapsto \widetilde{\mathsf{ASub}}$ defined below reduces redundance in abstract substitutions. It makes use of the following properties of a substitution σ: (1) if $\sigma(X)$ is ground then X does not share with any variable under σ and (2) $\sigma(X)$ is non-linear only if $\sigma(X)$ is in mode ⓞ. Here, aliasing information helps to reduce redundance in sharing information through grd.

$$red(\sigma^\flat) \stackrel{def}{=} < \mathbf{M}_{\sigma^\flat}, \left\{ (X,Y) \in \mathbf{S}_{\sigma^\flat} \middle| \begin{array}{c} \neg grd(X,\sigma^\flat) \wedge \neg grd(Y,\sigma^\flat) \\ \wedge\ (X \equiv Y \to ⓞ \in \mathbf{M}_{\sigma^\flat}(X)) \end{array} \right\}, \mathbf{A}_{\sigma^\flat} > \tag{4}$$

$\overline{\chi} : \mathsf{Term}(\widetilde{\mathbf{V}_{\mathsf{P}}}) \times \widetilde{\mathsf{ASub}} \mapsto \{0,1,2\}$ defined below approximates $\chi \circ \sigma(t)$ for any σ satisfying $\sigma^\flat$, given $t \in \mathsf{Term}$ and $\sigma^\flat \in \widetilde{\mathsf{ASub}}$. Through grd, aliasing information helps to define $\overline{\chi}$ more accurately that is later used to estimate pair sharing caused by the unification of an equation. Let $t' \in \mathsf{Term}$ be a non-variable term.

$$\overline{\chi}(X,\sigma^\flat) \stackrel{def}{=} \mathsf{if}\ grd(X,\sigma^\flat)\ \mathsf{then}\ 0\ \mathsf{elseif}\ (X,X) \in \mathbf{S}_{\sigma^\flat}\ \mathsf{then}\ 2\ \mathsf{else}\ 1$$

$$\overline{\chi}(t',\sigma^\flat) \stackrel{def}{=} \begin{cases} 0\ \mathsf{if}\ \forall X \in \mathbf{V}_{t'}.\overline{\chi}(X,\sigma^\flat) = 0 \\ 1\ \mathsf{elseif}\ \forall X \in \mathbf{V}_{t'}.(repeat(X,t') \to grd(X,\sigma^\flat)) \\ \qquad\qquad \wedge \\ \qquad\qquad \forall Y, Z \in \mathbf{V}_{t'}.(\overline{\chi}(Y,\sigma^\flat) = 1 \wedge \overline{\chi}(Z,\sigma^\flat) = 1 \to (Y,Z) \notin \mathbf{S}_{\sigma^\flat}) \\ 2\ \mathsf{otherwise} \end{cases}$$

Aliasing information is also used to detect failures in the normal unification. One rationale is that a term cannot be a proper sub-term of itself. Let $\mathcal{A} \in \widetilde{\mathbf{ER}}$ and $E \in \wp(\mathsf{Eq})$ be a set of equations of the form $X = t$ where X is a variable and t is a non-variable term. Define $pst : \widetilde{\mathbf{ER}} \times \wp(\mathsf{Eq}(\widetilde{\mathbf{V}_{\mathsf{P}}})) \mapsto \wp(\wp(\widetilde{\mathbf{V}_{\mathsf{P}}}) \times \wp(\widetilde{\mathbf{V}_{\mathsf{P}}}))$ as follows.

$$pst(\mathcal{A}, E) \stackrel{def}{=} \{< [Y]_{\mathcal{A}}, [X]_{\mathcal{A}} > \mid (X = t) \in E \wedge Y \in \mathbf{V}_t\}$$

If $< [Y]_{\mathcal{A}}, [X]_{\mathcal{A}} > \in pst(\mathcal{A}, E)$ then $\sigma(Y)$ is a proper sub-term of $\sigma(X)$ for any substitution σ satisfying $\mathcal{A}$. The proper sub-term relationship is transitive. Therefore, if there is a variable X such that $< [X]_{\mathcal{A}}, [X]_{\mathcal{A}} > \in (pst(\mathcal{A}, E))^*$ then $\sigma(E)$ cannot be unified for any σ satisfying $\mathcal{A}$. Another rationale for detecting failures in the normal unification is that if $\sigma(t_1) \equiv \sigma(t_2)$ then t_1 and t_2 must unify. Define $failure : \widetilde{\mathbf{ER}} \times \wp(\mathsf{Eq}(\widetilde{\mathbf{V}_{\mathsf{P}}})) \mapsto \{true, false\}$ as follows.

$$failure(\mathcal{A}, E) \stackrel{def}{=} \begin{cases} \exists X. < [X]_{\mathcal{A}}, [X]_{\mathcal{A}} > \in (pst(\mathcal{A}, E))^* \\ \vee \\ \exists (X_1 = t_1), (X_2 = t_2) \in E. \\ (X_1 \not\equiv X_2 \wedge (X_1, X_2) \in \mathcal{A} \wedge mgu(t_1, t_2) = fail) \end{cases}$$

Example 1. Let $\widetilde{\mathbf{V}_{\mathsf{P}}} = \{U, V, W, X, Y, Z\}$, $E = \{W = f(V), U = g(Y), Z = h(X)\}$ and $\mathcal{A}$ be the equivalence relation on $\widetilde{\mathbf{V}_{\mathsf{P}}}$ which partitions $\widetilde{\mathbf{V}_{\mathsf{P}}}$ into the following equivalence classes: $\{U, V\}$, $\{W, X\}$ and $\{Y, Z\}$. Then $failure(\mathcal{A}, E)$ is true since $< \{U, V\}, \{U, V\} > \in (pst(\mathcal{A}, E))^*$.

Let $E' = \{W = f(Y, U), X = g(Z)\}$. Then $failure(\mathcal{A}, E')$ is true since $f(Y, U)$ and $g(Z)$ do not unify and $< W, X > \in \mathcal{A}$. ∎

5 Abstract Unification

Let $\theta^\flat, \sigma^\flat \in \mathsf{ASub}$, $A, B \in \mathsf{Atom}(\mathbf{V}_{\mathsf{P}})$. $AUNIFY$ computes $AUNIFY(A, \theta^\flat, B, \sigma^\flat)$ as follows. It first computes $E_0 = eq \circ mgu(\Psi(A), B)$. If $E_0 = fail$ then it returns $\perp$. Otherwise, it computes an abstract substitution $\eta^\flat \in \widetilde{\mathsf{ASub}}$ such that, $mgu(\zeta(E_0)) \circ \zeta \in \widetilde{\gamma}(\eta^\flat)$ for any $\zeta \in \widetilde{\gamma}(\theta^\flat \uplus \sigma^\flat)$. This is accomplished by $solve : \wp(\mathsf{Eq}(\widetilde{\mathbf{V}_{\mathsf{P}}})) \times \widetilde{\mathsf{ASub}} \mapsto \widetilde{\mathsf{ASub}}$ which we will explain later. It finally restricts $\eta^\flat$ to $\mathbf{V}_{\mathsf{P}}$ via $rest$.

$$AUNIFY(A, \theta^\flat, B, \sigma^\flat) \stackrel{def}{=} \begin{cases} \text{let } E_0 = eq \circ mgu(\Psi(A), B) \text{ in} \\ \text{if } E_0 \neq fail \text{ then } rest \circ solve(E_0, \theta^\flat \uplus \sigma^\flat) \text{ else} \perp \end{cases} \tag{5}$$

$solve$ can be thought of as an approximate constraint solver [2, 3]. A constraint is a pair $(E, \theta^\flat) \in \wp(\mathsf{Eq}(\widetilde{\mathbf{V}_{\mathsf{P}}})) \times \widetilde{\mathsf{ASub}}$. A substitution is said to satisfy $(E, \theta^\flat)$ iff it is equivalent (modulo renaming) to $mgu(\theta(E)) \circ \theta$ for some $\theta \in \widetilde{\gamma}(\theta^\flat)$. What $solve$ does is to transform $(E, \theta^\flat)$ into $(\emptyset, \eta^\flat)$ such that any substitution satisfying $(E, \theta^\flat)$ also satisfies $(\emptyset, \eta^\flat)$. Note that the set of substitutions satisfying $(\emptyset, \eta^\flat)$ is $\widetilde{\gamma}(\eta^\flat)$. $solve$ does so by first calling $sol : \wp(\mathsf{Eq}(\widetilde{\mathbf{V}_{\mathsf{P}}})) \times \widetilde{\mathsf{ASub}} \mapsto \widetilde{\mathsf{ASub}}$ to transform $(E, \theta^\flat)$ to $(E_2, \zeta^\flat)$ such that any substitution satisfying $(E, \theta^\flat)$ satisfies $(E_2, \zeta^\flat)$ where E_2 is the set of the equations in E whose right sides are not variables, then calling *failure* to detect failures in the normal unification according to information in E_2 and $\mathbf{A}_{\zeta^\flat}$, and then calling *sol* to transform $(E_2, \zeta^\flat)$ to $(\emptyset, \eta^\flat)$ such that any substitution satisfying $(E_2, \zeta^\flat)$ satisfies $(\emptyset, \eta^\flat)$. *sol* repeatedly calls $soln : \mathsf{Eq}(\widetilde{\mathbf{V}_{\mathsf{P}}}) \times \widetilde{\mathsf{ASub}} \mapsto \widetilde{\mathsf{ASub}}$ to transform $(E \cup \{e\}, \theta^\flat)$ into $(E, \sigma^\flat)$ such that any substitution satisfying $(E \cup \{e\}, \theta^\flat)$ also satisfies $(E, \sigma^\flat)$.

$$solve(E, \theta^\flat) \stackrel{def}{=} \begin{cases} \mathsf{let}\ E_2 = E \setminus \{(X = Y) \in E \mid Y \in \mathsf{Vars}\}, \\ \qquad \zeta^\flat = sol(\{(X = Y) \in E \mid Y \in \mathsf{Vars}\}, \theta^\flat), \\ \mathsf{in\ if}\ failure(\mathbf{A}_{\zeta^\flat}, E_2)\ \mathsf{then}\ \widetilde{\perp}\ \mathsf{else}\ sol(E_2, \zeta^\flat) \end{cases} \tag{6}$$

$$sol(E, \theta^\flat) \stackrel{def}{=} \mathsf{if}\ E = \emptyset\ \mathsf{then}\ \theta^\flat\ \mathsf{else}\ sol(E', soln(e, \theta^\flat))\ \mathsf{where}\ E = \{e\} \cup E' \tag{7}$$

Let $\sigma^\flat \in \widetilde{\mathsf{ASub}}$ and $e \in \mathsf{Eq}(\widetilde{\mathbf{V}_{\mathsf{P}}})$. $soln(e, \sigma^\flat)$ ensures that, for any $\sigma \in \widetilde{\gamma}(\sigma^\flat)$, if $\eta = mgu(\{\sigma(e)\}) \neq fail$ then $\eta \circ \sigma \in \widetilde{\gamma} \circ soln(e, \sigma^\flat)$. The unification of $\{\sigma(e)\}$ will possibly (i) make variables in $\mathbf{V}_e$ more instantiated, (ii) cause more pair sharing among variables in $\mathbf{V}_e$, and (iii) cause more aliasing among variables in $\mathbf{V}_e$. The composition of $\eta = mgu(\{\sigma(e)\})$ to σ will possibly (iv) make variables in $\widetilde{\mathbf{V}_{\mathsf{P}}} \setminus \mathbf{V}_e$ more instantiated because of sharing, (v) cause more pair sharing among variables in $\widetilde{\mathbf{V}_{\mathsf{P}}} \setminus \mathbf{V}_e$ and/or more pair sharing between variables in $\mathbf{V}_e$ and variables in $\widetilde{\mathbf{V}_{\mathsf{P}}} \setminus \mathbf{V}_e$, and (vi) cause more aliasing among variables in $\widetilde{\mathbf{V}_{\mathsf{P}}} \setminus \mathbf{V}_e$ and/or more aliasing between variables in $\mathbf{V}_e$ and variables in $\widetilde{\mathbf{V}_{\mathsf{P}}} \setminus \mathbf{V}_e$. The effects of (i), (ii) and (iii) are estimated by $ueq : \mathsf{Eq}(\widetilde{\mathbf{V}_{\mathsf{P}}}) \times \widetilde{\mathsf{ASub}} \mapsto \widetilde{\mathsf{ASub}}$. The effect of (iv), (v) and (vi) is estimated by $comp : \widetilde{\mathsf{ASub}} \times \widetilde{\mathsf{ASub}} \times \wp(\widetilde{\mathbf{V}_{\mathsf{P}}}) \times \wp(\widetilde{\mathbf{V}_{\mathsf{P}}}) \mapsto \widetilde{\mathsf{ASub}}$. *comp* takes $aff(e, \mathbf{M}_{\sigma^\flat})$ as an input so as to avoid unnecessary and possibly inaccurate updates to the instantiation modes of the varaibles outside $\mathbf{V}_e$. *comp* also takes $\mathbf{V}_e$ as an input so as to update the instantiation modes of those variables that are aliased with variables in $\mathbf{V}_e$. Here, aliasing information is used to make more accurate updates to mode information. *soln* finally calls *red* to reduce redundance.

$$soln(e, \sigma^\flat) \stackrel{def}{=} red \circ comp(\sigma^\flat, ueq(e, \sigma^\flat), aff(e, \mathbf{M}_{\sigma^\flat}), \mathbf{V}_e) \tag{8}$$

$$ueq(X = t, \sigma^\flat) \stackrel{def}{=} \eta^\flat \tag{9}$$

where

$$\mathbf{M}_{\eta^\flat} = est(X = t, \mathbf{M}_{\sigma^\flat}) \tag{10}$$

$$\mathbf{S}_{\eta^\flat} = \begin{cases} \mathsf{let}\ \kappa^1 = \overline{\chi}(X, \sigma^\flat),\ \kappa^2 = \overline{\chi}(t, \sigma^\flat)\ \mathsf{in} \\ \begin{cases} \emptyset & \mathsf{if}\ \kappa^1 = 0 \vee \kappa^2 = 0 \\ \{(X,Y), (Y,X) \mid Y \in \mathbf{V}_t\} & \mathsf{if}\ \kappa^1 = 1 \wedge \kappa^2 = 1 \\ \{(X,Y), (Y,X) \mid Y \in \mathbf{V}_t\} \cup \{(X,X)\} & \mathsf{if}\ \kappa^1 = 1 \wedge \kappa^2 = 2 \\ \{(X,Y), (Y,X) \mid Y \in \mathbf{V}_t\} \cup \mathbf{V}_t^2 & \mathsf{if}\ \kappa^1 = 2 \wedge \kappa^2 = 1 \\ \{(X,Y), (Y,X) \mid Y \in \mathbf{V}_t\} \cup \{(X,X)\} \cup \mathbf{V}_t^2 & \mathsf{otherwise} \end{cases} \end{cases} \tag{11}$$

$$\mathbf{A}_{\eta^\flat} = \mathsf{if}\ t \in \widetilde{\mathbf{V}_{\mathsf{P}}}\ \mathsf{then}\ \{X, t\}^2\ \mathsf{else}\ \{(X,X)\} \cup \{(Y,Y) \mid Y \in \mathbf{V}_t\} \tag{12}$$

$$comp(\sigma^\flat, \eta^\flat, \mathsf{Z}, \mathsf{Z}_0) \stackrel{def}{=} \zeta^\flat \tag{13}$$

where

$$\mathbf{M}_{\zeta^\flat} = \lambda U. \begin{cases} \mathbf{M}_{\eta^\flat}(Y) & \text{if } \exists Y \in \mathsf{Z}_0.(U,Y) \in \mathbf{A}_{\sigma^\flat} \\ \triangle & \text{elseif } ⓕ \in \mathbf{M}_{\sigma^\flat}(U) \wedge \exists Z \in \mathsf{Z}.(U,Z) \in \mathbf{S}_{\sigma^\flat} \\ \{ⓖ, ⓞ\} & \text{elseif } ⓞ \in \mathbf{M}_{\sigma^\flat}(U) \wedge \exists Z \in \mathsf{Z}.(U,Z) \in \mathbf{S}_{\sigma^\flat} \\ \mathbf{M}_{\sigma^\flat}(U) & \text{otherwise} \end{cases} \tag{14}$$

$$\begin{aligned} \mathbf{S}_{\zeta^\flat} = {} & \mathbf{S}_{\sigma^\flat} \cup \mathbf{S}_{\eta^\flat} \cup \{(U,Z) | \exists V.((U,V) \in \mathbf{S}_{\eta^\flat} \wedge (V,Z) \in \mathbf{S}_{\sigma^\flat})\} \cup \\ & \{(W,V) | \exists U.((W,U) \in \mathbf{S}_{\sigma^\flat} \wedge (U,V) \in \mathbf{S}_{\eta^\flat})\} \cup \\ & \{(W,Z) | \exists U, V.((W,U) \in \mathbf{S}_{\sigma^\flat} \wedge (U,V) \in \mathbf{S}_{\eta^\flat} \wedge (V,Z) \in \mathbf{S}_{\sigma^\flat})\} \end{aligned} \tag{15}$$

$$\mathbf{A}_{\zeta^\flat} = (\mathbf{A}_{\sigma^\flat} \cup \mathbf{A}_{\eta^\flat})^* \tag{16}$$

Example 2. Consider the computation of $AUNIFY(A, \theta^\flat, B, \sigma^\flat)$ where

$$\begin{aligned} A &= t(X,Y,Z) \\ B &= t(K, f(K,L), L) \\ \theta^\flat &= <\{X \mapsto \{ⓖ\}, Y \mapsto \{ⓞ\}, Z \mapsto \{ⓕ\}\}, \{(Y,Y)\}, \{(X,X),(Y,Y),(Z,Z)\}> \\ \sigma^\flat &= <\{K \mapsto \{ⓕ\}, L \mapsto \{ⓕ\}\}, \emptyset, \{(K,K),(L,L)\}> \end{aligned}$$

Suppose $\Psi(X) = X_0$ for any $X \in \mathsf{V_P}$, we have,

$$E_0 = eq \circ mgu(\Psi(A), B) = \{X_0 = K, Y_0 = f(K,L), Z_0 = L\}$$

$$\begin{aligned} \zeta^\flat = \theta^\flat \uplus \sigma^\flat = {} & <\{X_0 \mapsto \{ⓖ\}, Y_0 \mapsto \{ⓞ\}, Z_0 \mapsto \{ⓕ\}, K \mapsto \{ⓕ\}, L \mapsto \{ⓕ\}\}, \\ & \{(Y_0,Y_0)\}, \\ & \{(X_0,X_0),(Y_0,Y_0),(Z_0,Z_0),(K,K),(L,L)\}> \end{aligned}$$

The computation of $\eta^\flat = solve(E_0, \zeta^\flat)$ is accomplished by calling *soln* to solve one equation in E_0 after another with $\zeta^\flat$ as the constraint for the first equation and the intermediate results as the constraints for other equations. The result is

$$\begin{aligned} \eta^\flat = {} & <\{X_0 \mapsto \{ⓖ\}, Y_0 \mapsto \{ⓖ, ⓞ\}, Z_0 \mapsto \triangle, K \mapsto \{ⓖ\}, L \mapsto \triangle\}, \\ & \{(Y_0,Y_0),(Z_0,Z_0),(L,L),(Y_0,L),(L,Y_0),(Z_0,L),(L,Z_0)\}, \\ & \left\{ \begin{array}{l} (X_0,K),(K,X_0),(Z_0,L),(L,Z_0), \\ (X_0,X_0),(Y_0,Y_0),(Z_0,Z_0),(K,K),(L,L) \end{array} \right\} > \end{aligned}$$

Finally, $rest(\eta^\flat) = <\{K \mapsto \{ⓖ\}, L \mapsto \triangle\}, \{(L,L)\}, \{(K,K),(L,L)\}>$. So,

$$AUNIFY(A, \theta^\flat, B, \sigma^\flat) = <\{K \mapsto \{ⓖ\}, L \mapsto \triangle\}, \{(L,L)\}, \{(K,K),(L,L)\}>$$

∎

Theorem 5. *$UNIFY(A, \gamma(\theta^\flat), B, \gamma(\sigma^\flat)) \subseteq \gamma \circ AUNIFY(A, \theta^\flat, B, \sigma^\flat)$ for any $A, B \in \mathsf{Atom}(\mathsf{V_P})$, $\theta^\flat, \sigma^\flat \in \mathsf{ASub}$.* ∎

Example 3. This example illustrates the improvement on mode analysis due to the incorporation of aliasing information. The program is the following and the goal is of the form :- a(X) with X being a variable.

```
a(Y) :- b(Y),c(Y,Z),d(Z).
b(W).    b(1).    c(U,U).     d(2).
```

The following are the results of mode analyses with and without aliasing information. Each textual point of the body of a clause or a goal is annotated with an abstract substitution which is of the form $(\mathcal{I}, \mathcal{S}, \mathcal{A})$ for the analysis with aliasing or of the form $(\mathcal{I}, \mathcal{S})$ for the analysis without aliasing.

```
% analysis with aliasing

:-         %([X/{f}],[],[(X,X)])
    a(X). %([X/{g}],[],[(X,X)])

a(Y) :-
           %(Y/{f},Z/{f}], [], [(Y,Y),(Z,Z)])
    b(Y),   %[Y/{f,g},Z/{f}], [], [(Y,Y),(Z,Z)])
    c(Y, Z),%([Y/{f,g},Z/{f,g}], [(Y,Z),(Z,Y)],
           %                  [(Y,Y),(Y,Z),(Z,Y),(Z,Z)]),
    d(Z).   %([Y/{g},Z/{g}], [], [(Y,Y),(Y,Z),(Z,Y),(Z,Z)]).

    b(W).   %([W/{f}], [], [(W,W)])
    b(1).   %([], [], [])
    c(U, U).%([U/{f,g}], [], [(U,U)])
    d(2).   %([], [], [])
```

For mode analysis with aliasing information, the first goal b(Y) in the clause defining a/1 binds Y to either a ground term or a variable. The second goal c(Y,Z) in the same clause makes Y and Z both share and alias via the clause defining c/2. The last goal d(Z) makes Z ground via the clause defining d/1. It also makes Y ground because Y is aliased with Z.

```
% analysis without aliasing

:-          %([X/{f}],[])
    a(X).   %([X/{f,g,o}],[])

a(Y) :-     %([Y/{f},Z/{f}], [])
    b(Y),   %([Y/{f,g},Z/{f}], [])
    c(Y, Z),%([Y/{f,g},Z/{f,g,o}], [(Y,Z),(Z,Y)])
    d(Z).   %([Y/{f,g,o},Z/{g}], [])

b(W).   %([W/{f}], [])
b(1).   %([], [])
c(U, U).%([U/{f,g}], [])
d(2).   %([], [])
```

For mode analysis without aliasing information, the first goal b(Y) in the clause defining a/1 binds Y to either a ground term or a variable. The second goal c(Y,Z) in the same clause makes Y and Z share via the clause defining c/2. The last goal d(Z) makes Z ground via the clause defining d/1. The analysis cannot make Y ground since aliasing information is not available. It has to add mode $\textcircled{o}$ into the mode description of Y since Y and Z share and Z has been further instantiated.

The presence of aliasing information makes it possible to infer that X is ground (corresponding to mode description $\{\textcircled{g}\}$ whenever P succeeds on $a(X)$ while without aliasing no mode information about X (corresponding to mode description $\{\textcircled{f}, \textcircled{g}, \textcircled{o}\}$) can be inferred. ∎

6 Conclusion

We have presented a mode analysis for logic programs. Mode information is derived together with sharing and aliasing information. Sharing information is necessary to ensure the correctness of analysis while aliasing information is used to increase the accuracy of analysis. Aliasing information is used to reduce redundance in sharing information, to define the approximate multiplicity function more accurately, to detect failures in the normal unification and to make more accurate updates to mode information.

The algorithm for abstract unification resembles the algorithm for the normal unification as described in [13]. The idea of performing abstract unification by abstracting the normal unification process was inspired by [2]. However, there are significant differences between our work and [2]. Firstly, [2] uses one of $\{\mathbf{g}, \mathbf{a}\}$ as the mode description of a variable where **g** denotes the set of ground terms and **a** denotes Term. So, only groundness information is captured. In this case, sharing information is not necessary for the purpose of safeness but is used to increase the accuracy of mode inference. Since **a** is a safe mode description of any term, it is not necessary to update the mode description of any other variable when the mode description of one variable changes even if the two variables might share. Secondly, in [2], given a set of equations and an abstract substitution, an equation which is ground (more instantiated) in the abstract substitution is solved before an equation that is not ground (less instantiated) in the abstract substitution. We do not solve equations in a particular instantiation order because it is not possible to find an instantiation order on the mode descriptions that we have used. Indeed, $\otimes$ is not a least bound operator since $\{\textcircled{o}\}\otimes\{\textcircled{o}\} \neq \{\textcircled{o}\}$. To exploit an instantiation order, it is necessary to use a subset of $\wp(\triangle)$ as the set of mode descriptions such as that in [6].

References

1. M. Bruynooghe, G. Janssens, A. Callebaut, and B. Demoen. Abstract interpretation: towards the global optimisation of Prolog programs. In *Proceedings of the 1987 Symposium on Logic Programming*, pages 192–204. The IEEE Society Press, 1987.

2. M. Codish, D. Dams, and Yardeni E. Derivation and safety of an abstract unification algorithm for groundness and aliasing analysis. In Furukawa [9], pages 79–93.
3. 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 117–131. The MIT Press, 1993.
4. M. Codish, A. Mulkers, M. Bruynooghe, G. de la Banda, M. and M. Hermenegildo. Improving Abstract Interpretations by Combining Domains. In *Proceedings of the ACM SIGPLAN Symposium on Partial Evaluation and Semantics-Based Program Manipulation*, pages 194–205. ACM Press, 1993.
5. P. Cousot and R. Cousot. Abstract interpretation and application to logic programs. *The Journal of Logic Programming*, 13(1, 2, 3 and 4):103–179, 1992.
6. S. K. Debray. Functional computations in logic programs. *ACM Transactions on Programming Languages and Systems*, 11(3):451–481, 1989.
7. S. K. Debray. Static inference of modes and data dependencies in logic programs. *ACM Transactions on Programming Languages and Systems*, 11(3):418–450, 1989.
8. S. K. Debray and D. S. Warren. Automatic mode inference for logic programs. *The Journal of Logic Programming*, 5(3):207–230, September 1988.
9. K. Furukawa, editor. *Proceedings of the Eighth International Conference on Logic Programming*, Paris, France, 1991. The MIT Press.
10. N.D. Jones and H. Søndergaard. A semantics-based framework for abstract interpretation of Prolog. In S. Abramsky and C. Hankin, editors, *Abstract interpretation of declarative languages*, pages 123–142. Ellis Horwood Limited, 1987.
11. A. King. A synergistic analysis for sharing and groundness which traces linearity. *Draft*, 1994.
12. G. Levi and M. Martelli, editors. *Proceedings of the Sixth International Conference on Logic Programming*, Lisbon, 1989. The MIT Press.
13. J.W. Lloyd. *Foundations of Logic Programming*. Springer-Verlag, 1987.
14. L. Lu. Abstract interpretation, bug detection and bug diagnosis in normal logic programs. PhD thesis, University of Birmingham, 1994.
15. A. Mariën, G. Janssens, A. Mulkers, and M. Bruynooghe. The impact of abstract interpretation: An experiment in code generation. In Levi and Martelli [12], pages 33–47.
16. K. Muthukumar and M. Hermenegildo. Combined determination of sharing and freeness of program variables through abstract interpretation. In Furukawa [9], pages 49–63.
17. R. Sundararajan and J.S. Conery. An abstract interpretation scheme for groundness, freeness, and sharing analysis of logic programs. In R. Shyamasundar, editor, *Proceedings of 12th Conference on Foundations of Software Technology and Theoretical Computer Science*, pages 203–216. Springer-Verlag, 1992.
18. A. Taylor. Removal of dereferencing and trailing in Prolog compilation. In Levi and Martelli [12], pages 48–60.
19. D. H. D. Warren. Implementing Prolog - compiling predicate logic programs. Research Report 39 and 40, Department of artificial intelligence, The University of Edinburgh, 1977.

This article was processed using the LaTeX macro package with LLNCS style