Abstract Fixpoint Semantics and Abstract Procedural Semantics of Definite Logic Programs

Lunjin Lu* Peter Greenfield

School of Computer Science, The University of Birmingham
Edgbaston, Birmingham B15 2TT, UK

Abstract

An abstraction function α is called stable if $\alpha(t_1) = \alpha(t_2)$ implies $\alpha(t_1\sigma) = \alpha(t_2\sigma)$ for any terms t_1, t_2 and any substitution σ. For each such abstraction function α, an abstract fixpoint semantics of a definite logic program P is given as the least fixpoint of a function $\Phi_{P,\alpha}$. The safeness of using $\Phi_{P,\alpha}$ as an approximation of van Emden and Kowalski's T_P is given. A partial SLD resolution over the concrete domain is then defined to give a procedural characterisation of the abstract fixpoint semantics. Then, the partial SLD resolution is generalised resulting in an abstract SLD resolution working on the abstract domain. The soundness and the completeness of the abstract SLD resolution state that the abstract procedural semantics given by the abstract SLD resolution is equal to the abstract fixpoint semantics. The usefulness of the results is illustrated by depth abstractions. The generalisation of the results to normal logic programs and other possible extensions are also discussed.

1 Introduction

Abstract interpretations are useful in deriving properties of programs for various program analyses. Program analyses are viewed as program execution over non-standard data domains. Cousot and Cousot's first laid solid semantic foundations for abstract interpretations [3, 4]. Their idea is to define a collecting semantics for a program which associates each program point with the set of all storage states that are possibly obtained when the execution reaches the point. In practice, the collecting semantics is rarely calculated. Instead, an approximation of the collecting semantics is calculated by simulating over a non-standard data domain the computation of the collecting semantics over the standard data domain. The standard data domain and the non-standard domain are called the concrete domain and the abstract domain respectively. Throughout this paper, bold face lowercase letters and calligraphic uppercase letters denote abstract objects.

*Supported by the Sino-British Friendship Scholarship Scheme.

The abstract interpretation method has been investigated by the logic programming community. Mellish [9], Jones *et al.* [5], Bruynooghe [2] and Marriott *et al.* [7] have offered abstract interpretation frameworks for logic programs. A number of applications of abstract interpretation of logic programs have been proposed such as occur check [12], mode inference [8, 9, 1] and type inference [2, 1].

In this paper we discuss a class of abstract interpretations of definite logic programs that are characterised by stable abstraction functions, and give a procedural characterisation of the abstract fixpoint semantics.

In section 2 we give the notion of partial unification and define abstract fixpoint semantics. Section 3 defines a partial SLD resolution procedure over the concrete domain and relates it to the abstract fixpoint semantics by proving its soundness and completeness. Section 4 generalises the partial SLD resolution procedure resulting in an abstract SLD resolution procedure over the abstract domain. Section 5 illustrates the computation of the abstract fixpoint semantics for depth abstractions [11, 7] and the application of the abstract SLD resolution procedure. In section 6 we discuss related work and possible extensions to the results in this paper.

Before proceeding to the next section, we give some preliminary definitions. Let us assume that we deal with a definite logic program P, and that its Herbrand universe, Herbrand base and the least fixpoint semantics are U_P, B_P and M_P respectively. The concrete fixpoint semantics of definite logic programs that we consider is given by van Emden and Kowalski [13].

Definition 1.1 (van Emden and Kowalski) *The fixpoint semantics of a definite logic program P is the least fixpoint of function $T_P : \wp(B_P) \mapsto \wp(B_P)$ defined as follows.*

$$T_P(I) = \left\{ A_0\sigma \middle| \begin{array}{l} \exists\sigma.\exists A_0 \leftarrow A_1, \ldots, A_m \in P. \\ A_1\sigma, \ldots, A_m\sigma \in I, A_0\sigma \in B_P \end{array} \right\} \quad (1)$$

Definition 1.2 *If A is an atom, we define $[A]$ as the set of all its ground instances, i.e.,*

$$[A] = \{A\sigma | \exists\sigma.A\sigma \in B_P\}$$

Let $\mathcal{T}erms$ be the set of terms, α be a function from $\mathcal{T}erms$ to $\mathcal{T}erms$, called the set of abstract terms. The concrete domain we consider is a sub-lattice of $(\wp(\mathcal{T}erms), \subseteq)$ and the abstract domain a sub-lattice of $(\wp(\mathcal{T}erms), \subseteq)$.

The abstraction function α induces an equivalence relation $\equiv_\alpha$ on $\mathcal{T}erms$, as $t_1 \equiv_\alpha t_2$ iff $\alpha(t_1) = \alpha(t_2)$. When $t_1 \equiv_\alpha t_2$, t_1 (or t_2) is called α-equivalent to t_2 (or t_1). It is possible that two or more instances of a term t belong to the same equivalent class, i.e., there exist some θ_1 and θ_2 such that $t\theta_1 \equiv_\alpha t\theta_2$.

Definition 1.3 *The fact that* $t\theta_1 \equiv_\alpha t\theta_2$ *is denoted as* $\theta_1 \simeq_{\alpha,t} \theta_2$.

We generalise α to be a function $\alpha : \wp(\mathcal{T}erms) \mapsto \wp(\mathcal{T}erms)$ as $\alpha(S) = \{\alpha(s) | s \in S\}$. We define a concretisation function $\gamma : \mathcal{T}erms \mapsto \wp(\mathcal{T}erms)$ as $\gamma(\mathbf{t}) = \{t | t \in \mathcal{T}erms, \mathbf{t} = \alpha(t)\}$ and generalise γ to be $\gamma : \wp(\mathcal{T}erms) \mapsto \wp(\mathcal{T}erms)$ as $\gamma(S) = \{\gamma(s) | s \in S\}$.

2 Partial unification and abstract fixpoint semantics

In this section, we define the notion of partial unification for each abstraction function α, denoted by α-unification. In standard unification, two terms t_1 and t_2 are considered unifiable if there exists a substitution σ such that $t_1\sigma = t_2\sigma$. In α-unification, the identity relation $=$ is replaced by the equivalence relation $\equiv_\alpha$. Also, we give an abstract fixpoint semantics of a definite logic program for each stable abstraction function and establish its correctness.

Definition 2.1 *Let t_1 and t_2 be terms, and σ be a substitution. σ is called an α-unifier of t_1 and t_2 if $t_1\sigma \equiv_\alpha t_2\sigma$, i.e., $\alpha(t_1\sigma) = \alpha(t_2\sigma)$. Two terms t_1 and t_2 are called α-unifiable if they have at least one α-unifier.*

A term t is said to subsume a term s if there is a substitution σ such that $t\sigma \equiv_\alpha s$.

Definition 2.2 *An α-unifier σ of term t_1 and term t_2 is said to be most general if for any other α-unifier σ' of term t_1 and t_2, $t'\sigma$ is not subsumed by $t'\sigma'$, where $t' = t_1$ or $t' = t_2$.*

Definition 2.3 *An abstraction function is called stable, if, for any substitution θ and any terms t_1 and t_2, $t_1\theta \equiv_\alpha t_2\theta$ when $t_1 \equiv_\alpha t_2$.*

Lemma 2.1 *If α is stable, then,*

a *for any term t_1, any ground term t_2, and any θ, if $t_1 \equiv_\alpha t_2$ then $t_1\theta \equiv_\alpha t_2$.*

b *If $t_1 \in \gamma(\mathbf{t}_1)$ and $t_2 \in \gamma(\mathbf{t}_2)$ are α-unifiable with an α-unifier σ, then, for any $t_1' \in \gamma(\mathbf{t}_1)$ and any $t_2' \in \gamma(\mathbf{t}_2)$, t_1' and t_2' are α-unifiable and σ is an α-unifier of t_1' and t_2'.*

c *If a term t is α unifiable with a term $t_1 \in \gamma(\mathbf{t}_1)$ with α-mgu σ, then t is α unifiable with any term $t_1' \in \gamma(\mathbf{t}_1)$ with α-mgu σ.*

d *If σ is an α-mgu of $t_1 \in \gamma(\mathbf{t}_1)$ and $t_2 \in \gamma(\mathbf{t}_2)$, then σ is also an α-mgu of $t_1' \in \gamma(\mathbf{t}_1)$ and $t_2' \in \gamma(\mathbf{t}_2)$.*

Proof: First consider (**a**). $t_1 \equiv_\alpha t_2$ implies $t_1\theta \equiv_\alpha t_2\theta$ for any θ since α is stable. Since t_2 is ground, we have $t_2\theta = t_2$ and hence that $t_2\theta \equiv_\alpha t_2$. So, $t_1\theta \equiv_\alpha t_2$.

Now consider (**b**). By definition 2.3, $t_1'\sigma \equiv_\alpha t_1\sigma$ and $t_2'\sigma \equiv_\alpha t_2\sigma$. So, $t_1'\sigma \equiv_\alpha t_2'\sigma$ since $t_1\sigma \equiv_\alpha t_2\sigma$.

(**c**) is a special case of (**b**).

(**d**) σ is an α-unifier of t_1' and t_2' by (b). It suffices to show that $t_1'\sigma$ is not subsumed by $t_1'\sigma'$ and $t_2'\sigma$ is not subsumed by $t_2'\sigma'$ for any other α-unifier σ' of t_1' and t_2'. $t_1'\sigma$ is not subsumed by $t_1'\sigma'$ otherwise $t_1\sigma$ is subsumed by $t_1\sigma'$ and σ is not a most general α-unifier of t_1 and t_2. $\Box$

From now on, all the abstract functions that we consider are stable unless otherwise stated.

We are interested in $\alpha(M_P)$ and we relate it to $\alpha(B_P)$ by means of a function defined over $\wp(\alpha(B_P))$.

Definition 2.4 *Let P be a definite logic program and $\alpha(B_P) \subseteq \mathcal{T}erms$ be the α abstraction of B_P. We define $\Phi_{P,\alpha} : \wp(\alpha(B_P)) \mapsto \wp(\alpha(B_P))$ as follows.*

$$\Phi_{P,\alpha}(\mathcal{I}) = \left\{ \alpha(A_0\sigma) \;\middle|\; \begin{array}{l} \exists\sigma.\exists A_0 \leftarrow A_1, \ldots, A_m \in P. \\ A_0\sigma \in B_P, \ldots, A_m\sigma \in B_P, \\ \alpha(A_1\sigma) \in \mathcal{I}, \ldots, \alpha(A_m\sigma) \in \mathcal{I} \end{array} \right\} \quad (2)$$

$\Phi_{P,\alpha}$ is a monotonic function and $(\wp(\alpha(B_P)), \subseteq)$ is a complete lattice with set union $\cup$ as least upper bound operator. So, $\Phi_{P,\alpha}$ has the least fixpoint $lfp(\Phi_{P,\alpha}) = \cup_{i=0}^{i=\infty} \Phi_{P,\alpha}^i(\emptyset)$.

In order to compute $lfp(\Phi_{P,\alpha})$, we reformulate $\Phi_{P,\alpha}$ as follows.

Definition 2.5

$$\Phi'_{P,\alpha}(\mathcal{I}) = \left\{ \mathcal{A} \;\middle|\; \begin{array}{l} \exists\sigma.\exists A_0 \leftarrow A_1, \ldots, A_m \in P. \\ \alpha(A_1\sigma), \ldots, \alpha(A_m\sigma) \in \mathcal{I}, \\ \mathcal{A} \in \alpha([A_0\sigma] \end{array} \right\} \quad (3)$$

In this definition, we have relaxed the condition that the substitution σ makes the clause ground. The condition $\mathcal{A} \in \alpha([A_0\sigma])$ guarantees $\Phi'_{P,\alpha}(\mathcal{I}) \subseteq \alpha(B_P)$. We now prove that $\Phi'_{P,\alpha} = \Phi_{P,\alpha}$.

Lemma 2.2 $\Phi_{P,\alpha} = \Phi'_{P,\alpha}$.

Proof: It suffices to show that $\Phi'_{P,\alpha}(\mathcal{I}) = \Phi_{P,\alpha}(\mathcal{I})$ for any $\mathcal{I}$. This is done by proving $\Phi_{P,\alpha}(\mathcal{I}) \subseteq \Phi'_{P,\alpha}(\mathcal{I})$ and $\Phi'_{P,\alpha}(\mathcal{I}) \subseteq \Phi_{P,\alpha}(\mathcal{I})$ separately.

First, let $\mathcal{A} \in \Phi_{P,\alpha}(\mathcal{I})$. By definition 2.4, there exist some σ and some $C = A_0 \leftarrow A_1, \ldots, A_m \in P$ such that $C\sigma$ is ground, $\alpha(A_j\sigma) \in \Phi_{P,\alpha}(\mathcal{I})$ for $j \in \{1..m\}$ and $\mathcal{A} = \alpha(A_0\sigma)$. Since $A_0\sigma$ is ground, we have $[A_0\sigma] = \{A_0\sigma\}$ and hence $\alpha([A_0\sigma]) = \{\alpha(A_0\sigma)\} = \{\mathcal{A}\}$. Thus, $\mathcal{A} \in \alpha([A_0\sigma])$, which, together with $\alpha(A_j\sigma) \in \Phi_{P,\alpha}(\mathcal{I})$ for $j \in \{1..m\}$, implies that $\mathcal{A} \in \Phi'_{P,\alpha}(\mathcal{I})$. So, $\Phi_{P,\alpha}(\mathcal{I}) \subseteq \Phi'_{P,\alpha}(\mathcal{I})$.

Now suppose that $\mathcal{A} \in \Phi'_{P,\alpha}(\mathcal{I})$. By definition 2.5, there exist some σ and some $C = A_0 \leftarrow A_1, \ldots, A_m \in P$ such that $\alpha(A_j\sigma) \in \Phi_{P,\alpha}(\mathcal{I})$ for $j \in \{1..m\}$ and $\mathcal{A} \in \alpha([A_0\sigma])$. Let θ be the substitution such that $A_0\sigma\theta \in B_P$ and $\mathcal{A} = \alpha(A_0\sigma\theta)$. Then, for any $j \in \{1..m\}$, $\alpha(A_j\sigma\theta) \in \mathcal{I}$ since $A_j\sigma \in \mathcal{I}$ and $\mathcal{I} \subseteq \alpha(B_P)$. Let $\sigma' = \sigma\theta$. If $C\sigma'$ is ground, we have $\alpha(A_j\sigma') \in \mathcal{I}$ and $A_0\sigma' \in B_P$ and $\mathcal{A} = \alpha(A_0\sigma')$. If $C\sigma'$ is not ground, then there is some θ' such that $C\sigma'\theta'$ is ground. We have $\alpha(A_j\sigma'\theta') \in \mathcal{I}$ and $\mathcal{A} = \alpha(A_0\sigma'\theta') = \alpha(A_0\sigma')$. So, $\mathcal{A} \in \Phi_{P,\alpha}(\mathcal{I})$ and hence $\Phi'_{P,\alpha}(\mathcal{I}) \subseteq \Phi_{P,\alpha}(\mathcal{I})$. This completes the proof of the lemma. □

Lemma 2.1.c and lemma 2.2 allow the use of the α-unification algorithm in the computation of $lfp(\Phi_{P,\alpha})$. $\Phi_{P,\alpha}(\mathcal{I})$ can be computed with the help of α-unification. That $\exists\sigma.\forall j \in \{1..m\}\alpha(A_j\sigma) \in \mathcal{I}$ can be checked by verifying that A_1 is α-unifiable with some $A'_1 \in \gamma(\mathcal{A}'_1)$ for some $\mathcal{A}'_1 \in \mathcal{I}$ with α-mgu σ_1, then verifying that $A_2\sigma_1$ is α-unifiable with some $A'_2 \in \gamma(\mathcal{A}'_2)$ for some $\mathcal{A}'_2 \in \mathcal{I}$ with α-mgu σ_2, and so on.

We now present the correctness of the abstract interpretation framework as Lemmas 2.3, 2.4 and 2.5.

Lemma 2.3 *For any* $\mathcal{I} \in \wp(\alpha(B_P))$, $T_P(\gamma(\mathcal{I})) \subseteq \gamma(\Phi_{P,\alpha}(\mathcal{I}))$.

Proof: Let $A \in T_P(\gamma(\mathcal{I}))$. Then, by definition 1.1, there exist a clause $A_0 \leftarrow A_1, \ldots, A_m$ and some σ such that $A_0\sigma \in B_P$, $A = A_0\sigma$ and $A_1\sigma \in \gamma(\mathcal{I}), \ldots, A_m\sigma \in \gamma(\mathcal{I})$, i.e., $\alpha(A_1\sigma) \in \mathcal{I}, \ldots, \alpha(A_m\sigma) \in \mathcal{I}$. Thus, by definition 2.4, $\alpha(A_0\sigma) \in \Phi_{P,\alpha}(\mathcal{I})$ since $A_0\sigma \in B_P$. So, $\alpha(A) \in \Phi_{P,\alpha}(\mathcal{I})$. By the definition of γ, $A \in \gamma(\Phi_{P,\alpha}(\mathcal{I}))$. So, $T_P(\gamma(\mathcal{I})) \subseteq \gamma(\Phi_{P,\alpha}(\mathcal{I}))$. □

Lemma 2.3 states that $\Phi_{P,\alpha}$ safely approximates T_P. That is, atomic ground goals outside $\gamma(lfp(\Phi_{P,\alpha}))$ cannot succeed.

Lemma 2.4

$$\forall i \geq 0.\forall A.(A \in T^i_P(\emptyset) \Rightarrow \alpha(A) \in \Phi^i_{P,\alpha}(\emptyset)) \qquad (4)$$

Proof: The proof is by induction on i. First, the result holds for i=0, since $T^0_P(\emptyset) = \emptyset$. Next assume that the result holds for $i \leq n$. Let $A \in T_P(T^n_P(\emptyset))$. From definition 1.1, there exist $A_0 \leftarrow A_1, \ldots, A_m \in P$ and σ such that $A = A_0\sigma \in B$ and $A_j\sigma \in T^n_P(\emptyset)$ for $j \in 1..m$. By the induction assumption, $A_j\sigma \in T^n_P(\emptyset)$ implies that $\alpha(A_j\sigma) \in \Phi^n_{P,\alpha}(\emptyset)$. That, in addition to definition 2.4, implies that $\alpha(A) = \alpha(A_0\sigma) \in \Phi^{n+1}_{P,\alpha}(\emptyset)$. Thus, the result holds for $n+1$ and this completes the proof of the lemma. □

The reading of Lemma 2.4 is that every element in the concrete semantics has its abstraction image in the corresponding abstract fixpoint semantics.

Lemma 2.5

$$\alpha(M_P) \subseteq lfp(\Phi_{P,\alpha}) \qquad (5)$$

Proof: $\alpha(M_P) = \alpha(lfp(T_P)) = \alpha(\cup_{i=0}^{i=\infty} T^i_P(\emptyset)) = \cup_{i=0}^{i=\infty} \alpha(T^i_P(\emptyset))$. This and lemma 2.4 imply that $\alpha(M_P) \subseteq \cup_{i=0}^{i=\infty} \Phi^i_{P,\alpha}(\emptyset) = lfp(\Phi_{P,\alpha})$. □

Lemma 2.5 states that $lfp(\Phi_{P,\alpha})$ is an abstract interpretation of M_P.

Lemma 2.6 $lfp(\Phi_{P,\alpha}) = \alpha(M_P)$ *does not hold in general.*

Proof: The proof will be presented in example 5.1. □

3 Partial SLD resolution

In this section, we first define a partial SLD resolution procedure, called α-SLD and then give an abstract procedural semantics of definite logic programs. The partial SLD differs from the standard SLD in that it uses the partial unification defined in section 2 instead of the standard unification. The theoretical development in this section parallels that in [6]. The main result is that the abstraction of the set of abstract goals which α-SLD can derive is the same as $lfp(\Phi_{P,\alpha})$.

Definition 3.1 *Let G be $\leftarrow A_1, \ldots, A_j, \ldots, A_p$ and $C = B \leftarrow B_1, \ldots, B_q$, called the input clause, be a variant of a clause of P. Then G' is α-derived from G and C using α-mgu σ if the following conditions hold:*

a *A_j is an atom, called the selected atom, in G;*

b *σ is an α-mgu of A_j and B;*

c $G' = \leftarrow (A_1, \ldots, A_{j-1}, B_1, \ldots, B_q, A_{j+1}, \ldots, A_p)\sigma$

Definition 3.2 *An α-SLD derivation of $P \cup \{G\}$ consists of a sequence $G_0 = G, G_1, \ldots$ of goals, a sequence $C_1, C_2, \ldots$ of variants of program clauses and a sequence $\sigma_1, \sigma_2, \ldots$ of α-mgu's such that each G_{i+1} is α-derived from G_i and C_{i+1} using σ_{i+1}.*

Definition 3.3 *An α-SLD refutation of $P \cup \{G\}$ is a finite α-SLD derivation of $P \cup \{G\}$ with the empty goal □ as the last goal in the derivation. If $G_n = □$, then the refutation has length n.*

Definition 3.4 *An unrestricted α-SLD refutation is an α-SLD refutation except that we drop the requirement that the substitutions σ_i be α-mgu's. They only need to be α-unifiers.*

Lemma 3.1 *Let P be a definite logic program and G be a definite goal. If $P \cup \{G\}$ has an unrestricted α-SLD refutation, then $P \cup G$ has an α-SLD refutation such that, if $\theta_1, \ldots, \theta_n$ are the α-unifiers for the unrestricted α-SLD refutation and $\theta'_1, \ldots, \theta'_n$ are α-mgu's for the α-SLD refutation, then there exists some λ such that $\theta_1 \ldots \theta_n \simeq_{\alpha,G} \theta'_1 \ldots \theta'_n\lambda$.*

Proof: The proof is by induction on the length of the unrestricted α-SLD refutation.

Suppose first that $n = 1$. Thus $P \cup \{G\}$ has an unrestricted α-SLD refutation $G_0 = G, G_1 = □$ with input clause C_1 and α-unifier θ_1. Let θ' be the α-mgu of the atom in G and the head of the unit clause C_1 such that $G\theta'$ subsume $G\theta$. Then $\theta_1 \simeq_{\alpha,G} \theta'_1\lambda$ for some λ. Furthermore, $P \cup \{G\}$ has an α-SLD refutation $G_0 = G, G_1 = \Box$ with input clause C_1 and α-mgu θ'_1.

Next suppose that the result holds for n-1. Let $P \cup \{G\}$ have an unrestricted α-SLD refutation $G_0 = G, G_1, \ldots, G_n = \Box$ of length n with input clauses $C_1, \ldots, C_n$ and α-unifiers $\theta_1, \ldots, \theta_n$. There exists an α-mgu θ'_1 for the selected atom in G and the head of C_1 such that $\theta'_1 \simeq_{\alpha,G} \theta_1\eta$ for some η. Thus $P \cup \{G\}$ has an unrestricted α-SLD refutation $G_0 = G, G'_1, G_2, \ldots, G_n = \Box$ with input clauses $C_1, \ldots, C_n$ and α-unifiers $\theta'_1, \eta\theta_2, \ldots, \theta_n$ and $G_1 \equiv_\alpha G'_1\eta$. By the induction hypothesis, $P \cup \{G'_1\}$ has an α-SLD refutation $G'_1, G'_2, \ldots, G'_n = \Box$ with α-mgu's $\theta'_2, \ldots, \theta'_n$ such that $\eta\theta_2 \ldots \theta_n \simeq_{\alpha,G'_1} \theta'_2 \ldots \theta'_n\lambda$ for some λ. Thus, $P \cup \{G\}$ has an α-SLD refutation $G_0 = G, G'_1, \ldots, G'_n$ with α-mgu's $\theta'_1, \ldots, \theta'_n$ such that $\theta_1 \ldots \theta_n \simeq_{\alpha,G} \theta'_1\eta\theta_2 \ldots \theta_n \simeq_{\alpha,G} \theta'_1 \ldots \theta'_n\lambda$. $\Box$

Theorem 3.1 (Soundness of partial SLD) *Let P be a definite logic program and G be a definite goal of form $A_1, \ldots, A_m$. If $P \cup \{G\}$ has an α-SLD refutation of length n and $\sigma_1, \ldots, \sigma_n$ be α-mgu's, then $\forall j \in \{1..m\}.(\alpha([A_j\sigma_1 \ldots \sigma_n]) \subseteq \Phi^{(n)}_{P,\alpha}(\emptyset)$.*

Proof: The result is proved by induction on the length n of the α-SLD refutation.

Suppose first that $n = 1$. Then G must be the form of $\leftarrow A_1$ and the program P has a unit clause of form $A \leftarrow$ and $A_1\sigma_1 \equiv_\alpha A\sigma_1$. By the definition of α, $\alpha([A\sigma_1]) \subseteq \alpha([A]) \subseteq \Phi^1_{P,\alpha}(\emptyset)$. So, $\alpha([A_1\sigma_1]) \subseteq \Phi^{(n)}_{P,\alpha}(\emptyset)$ since $A_1\sigma \equiv_\alpha A\sigma$

Next suppose that the result holds for any α-SLD refutation of length n-1. Consider an α-SLD refutation of $P \cup \{G\}$ of length n. Let A_j be an atom of G. If A_j is not the selected atom, then $A_j\sigma_1$ is an atom of G_1, the second goal of the α-SLD refutation. $\alpha([A_j\sigma_1 \ldots \sigma_n]) \subseteq \Phi^{n-1}_{P,\alpha}(\emptyset)$ by the induction hypothesis. So, $\alpha([A_j\sigma_1 \ldots \sigma_n]) \subseteq \Phi^n_{P,\alpha}(\emptyset)$ by the monotonicity of $\Phi_{P,\alpha}$. If A_j be the selected atom, letting $B \leftarrow B_1, \ldots, B_q$, $(q \geq 0)$ be the first input clause, then $A_j\sigma_1 \equiv_\alpha B\sigma_1$. If $q = 0$, $\alpha([B]) \subseteq \Phi^1_{P,\alpha}(\emptyset)$ by the definition of $\Phi_{P,\alpha}$. Thus, $\alpha([A_j\sigma_1 \ldots \sigma_n]) \subseteq \alpha([A_j\sigma_1]) \subseteq \alpha([B]) \subseteq \Phi^1_{P,\alpha}(\emptyset) \subseteq \Phi^n_{P,\alpha}(\emptyset)$. If $q > 0$, by the induction hypothesis, $\alpha([B_i\sigma_1 \ldots \sigma_n]) \subseteq \Phi^{n-1}_{P,\alpha}(\emptyset)$ for $i = 1, 2, \ldots, q$. By the definition of $\Phi_{P,\alpha}$, we have that $\alpha([B\sigma_1 \ldots \sigma_n]) \subseteq \Phi^n_{P,\alpha}(\emptyset)$. Because α is stable, we have $\alpha([A_j\sigma_1 \ldots \sigma_n]) = \alpha([B\sigma_1 \ldots \sigma_n]) \subseteq \Phi^n_{P,\alpha}(\emptyset)$ since $A_j\sigma_1 \equiv_\alpha B\sigma_1$. $\Box$

Theorem 3.2 (Completeness of partial SLD) *If $\alpha(A) \in lfp(\Phi_{P,\alpha})$, then A has an α-SLD resolution.*

Proof: It suffices to show that for any atom A and any n, $\alpha(A) \in \Phi^n_{P,\alpha}(\emptyset)$ implies that A has an α-SLD refutation. The result is proved by induction on n.

Suppose that n=1. That $\alpha(A) \in \Phi_{P,\alpha}(\emptyset)$ implies that there is a unit clause of form $A_1 \leftarrow$ and a substitution θ such that $A \equiv_\alpha A_1\theta$ and $A_1\theta$ is ground. By lemma 2.1.a, $A\theta \equiv_\alpha A_1\theta$. So, A has an unrestricted α-SLD refutation. By lemma 3.1, A has an α-SLD refutation.

Next suppose that the result is true for $n - 1$. That $\alpha(A) \in \Phi^n_{P,\alpha}(\emptyset)$ implies that there exist some σ and some clause $C = A_0 \leftarrow A_1, \ldots, A_q$ such that C is ground, $A \equiv_\alpha A_0\sigma$ and $\alpha(A_1\sigma), \ldots, \alpha(A_q\sigma) \in \Phi^{n-1}_{P,\alpha}(\emptyset)$. The induction hypothesis implies that $A_j\sigma$ has an α-SLD refutation for $j = 1, \ldots, q$. So, $(A_1, \ldots, A_q)\sigma$ has an α-SLD refutation since $A_j\sigma$ is ground for $j = 1, \ldots, q$. That $A_0\sigma$ is ground and $A \equiv_\alpha A_0\sigma$ implies that $A\sigma \equiv_\alpha A_0\sigma$ Thus A has an unrestricted α-SLD refutation with $G_0 = A, G_1 = (A_1, \ldots, A_q)\sigma, \ldots$ and input clause $C_1 = C$ and $\sigma_1 = \sigma$. Hence, A has an α-SLD refutation by lemma 3.1. $\Box$

Lemma 3.2 *If $A \in \gamma(\mathcal{A})$ and $P \cup \{A\}$ has an α-SLD of length n $G_0 = A, G_1, \ldots, G_n$, with input clauses $C_1, C_2, \ldots, C_n$ of variants of program clauses and α-mgu's $\sigma_1, \sigma_2, \ldots, \sigma_n$, then for any $A' \in \gamma(\mathcal{A})$, $P \cup \{A'\}$ has an α-SLD of length n $G'_0 = A', G_1, G_2, \ldots, G_n$ with input clauses $C_1, C_2, \ldots, C_n$ of variants of program clauses and α-mgu's $\sigma_1, \sigma_2, \ldots, \sigma_n$.*

Proof: Since, σ_1 is the α-mgu of A and the head of C_1, σ_1 is the α-mgu of A' and the head of C_1 by lemma 2.1.d. Thus, we have $G'_1 \equiv_\alpha G_1$. $\Box$

4 Abstract SLD resolution

The partial SLD resolution works in the concrete domain. It can be used to resolve definite goals using partial unification. Theorems 3.1 and 3.2 summarise the relation between the the abstract fixpoint semantics and the set of atoms which can be resolved by the partial SLD resolution and give a procedural characterisation of the abstract fixpoint semantics in the concrete domain.

Lemma 3.2 states that atoms in an equivalent class are homogeneous with respect to α-SLD resolution. The property of the equivalent class allows us to use any atom of the class to represent the whole class with respect to α-resolution and generalise the notion of partial SLD resolution to that of abstract SLD resolution which works on the abstract domain and resolves abstract goals using partial unification. First, we generalise the notion of the partial unification from the concrete domain to the abstract domain.

Definition 4.1 *Two abstract terms $\mathbf{t}_1$ and $\mathbf{t}_2$ are $\underline{\alpha}$-unifiable with an $\underline{\alpha}$-unifier σ iff there exist some $t_1 \in \gamma(\mathbf{t}_1)$ and $t_2 \in \gamma(\mathbf{t}_2)$ such that σ is an α-unifier of t_1 and t_2. An $\underline{\alpha}$-unifier σ of $\mathbf{t}_1$ and $\mathbf{t}_2$ is an $\underline{\alpha}$-mgu of $\mathbf{t}_1$ and $\mathbf{t}_2$ iff for any $\underline{\alpha}$-unifier τ of $\mathbf{t}_1$ and $\mathbf{t}_2$, $\mathbf{t}_1\sigma$ is not subsumed by $\mathbf{t}_1\tau$ and $\mathbf{t}_2\sigma$ is not subsumed by $\mathbf{t}_2\tau$.*

Lemma 4.1 *Two abstract terms $\mathbf{t}_1$ and $\mathbf{t}_2$ are $\underline{\alpha}$-unifiable iff any element of $\gamma(\mathbf{t}_1)$ is α-unifiable with any element of $\gamma(\mathbf{t}_2)$.*

Proof: The proof results from definition 4.1 and lemma 2.1.b. □

Definition 4.2 $\mathbf{t}\sigma = \mathbf{t}'$ *iff* $\mathbf{t}' = \alpha(t\sigma)$ *for some* $t \in \gamma(\mathbf{t})$.

Definition 4.3 *Let* $\mathcal{G}$ *be* $\leftarrow \mathcal{A}_1, \ldots, \mathcal{A}_j, \ldots, \mathcal{A}_p$ *and* $C = B \leftarrow B_1, \ldots, B_q$*, called input clause, be a variant of a clause of* P*. Then* $\mathcal{G}'$ *is* $\underline{\alpha}$*-derived from* $\mathcal{G}$ *and* C *using* $\underline{\alpha}$*-mgu* σ *if the following conditions hold:*

a $\mathcal{A}_j$ *is an atom, called selected atom, in* $\mathcal{G}$*;*

b σ *is a* $\underline{\alpha}$*-mgu of* $\mathcal{A}_j$ *and* $\alpha(B)$*;*

c $\mathcal{G}' =$

$$\leftarrow (\mathcal{A}_1, \ldots, \mathcal{A}_{j-1}, \alpha(B_1), \ldots, \alpha(B_q), \mathcal{A}_{j+1}, \ldots, \mathcal{A}_p)\sigma$$

Since the unification of two abstract terms is carried out by unifying two concrete terms, (**b**) is accomplished by unifying a concrete atom $A \in \gamma(\mathcal{A}_j)$ with B instead of applying α to B and then unifying $\mathcal{A}_j$ with $\alpha(B)$.

Definition 4.4

$$\alpha(\leftarrow A_1, \ldots, A_m) = \leftarrow \alpha(A_1), \ldots, \alpha(A_m)$$

Definition 4.5 *Let* $\mathcal{G}$ *be* $\leftarrow \mathcal{A}_1, \ldots, \mathcal{A}_m$*. We define*

$$\gamma(\mathcal{G}) = \{\leftarrow A_1, \ldots, A_m | A_1 \in \gamma(\mathcal{A}_1), \ldots, A_m \in \gamma(\mathcal{A}_m)\}$$

Lemma 4.2 *If* G' *is* α*-derived from* G *and* C *using* α*-mgu* σ*, then* $\alpha(G')$ *is* $\underline{\alpha}$*-derived from* $\alpha(G)$ *and* C *using* $\underline{\alpha}$*-mgu* σ*.*

Proof: The proof results from definition 3.1, definition 4.3 and lemma 3.2. □

Theorem 4.1 $\mathcal{G}'$ *is* $\underline{\alpha}$*-derived from* $\mathcal{G} = \mathcal{A}_1, \ldots, \mathcal{A}_p$ *using* $\underline{\alpha}$*-mgu* σ *and* $B \leftarrow B_1, \ldots, B_q$ *iff for any* $G' \in \gamma(\mathcal{G}')$*,* G' *is* α*-derived from any* $G \in \gamma(\mathcal{G})$ *using* α*-mgu* σ *and* $B \leftarrow B_1, \ldots, B_q$*.*

Proof: Sufficiency results from lemma 4.2. It remains to show the necessity.

Suppose that $\mathcal{G}'$ is derived from $\mathcal{G}$. Let $\mathcal{A}_j$ be the selected atom. Then $\mathcal{G}' =$

$$\leftarrow (\mathcal{A}_1, \ldots, \mathcal{A}_{j-1}, \alpha(B_1), \ldots, \alpha(B_q), \mathcal{A}_{j+1}, \ldots, \mathcal{A}_p)\sigma$$

Let $G' \in \gamma(\mathcal{G})$, then $G' = \leftarrow A'_1, \ldots, A'_{j-1}, B'_1, \ldots, B'_q, A'_{j+1}, \ldots, A'_p$ and $A'_i \in \gamma(\mathcal{A}_i\sigma)$ for $i \in \{1..(j-1), (j+1)..p\}$. By the transitivity of $\equiv_\alpha$, $A'_i \equiv_\alpha A_i\sigma$ for all $A_i \in \gamma(\mathcal{A}_i)$. $B'_1 \equiv_\alpha B_1\sigma$. So, G' is α-derived from any $G \in \gamma(\mathcal{G})$. This completes the proof. □

In parallel to definitions 3.2, 3.3 and 3.4, we can define the notions of $\underline{\alpha}$-derivation and $\underline{\alpha}$-refutation by simply replacing concrete goals with abstract goals.

Now we generalise the theorems 3.1 and 3.2 into the abstract domain.

Definition 4.6 *We define*

$$[\mathcal{A}] = \alpha([\gamma(\mathcal{A})])$$

Theorem 4.2 (Soundness of abstract SLD) *Let* P *be a definite logic program and* $\mathcal{G}$ *be a definite goal of form* $\leftarrow \mathcal{A}_1, \ldots, \mathcal{A}_m$*. If* $P \cup \{\mathcal{G}\}$ *has an* $\underline{\alpha}$*-SLD refutation of length* n *and* $\sigma_1, \ldots, \sigma_n$ *be* $\underline{\alpha}$*-mgu's, then* $\forall j \in \{1..m\}.([\mathcal{A}_j\sigma_1 \ldots \sigma_n]) \subseteq \Phi^{(n)}_{P,\alpha}(\emptyset)$.

Proof: By theorem 4.1, any $G \in \gamma(\mathcal{G})$ has an α-refutation of length n and $\sigma_1, \ldots, \sigma_n$ are the α-mgu's. Let $G = \leftarrow A_1, \ldots, A_m$. We have, $A_1 \in \gamma(\mathcal{A}_1), \ldots, A_m \in \gamma(\mathcal{A}_m)$. By theorem 3.1, $\forall j \in \{1..m\}.(\alpha([A_j\sigma_1 \ldots \sigma_n])) \subseteq \Phi^{(n)}_{P,\alpha}(\emptyset)$. So, $\forall j \in \{1..m\}.([\mathcal{A}_j\sigma_1 \ldots \sigma_n]) \subseteq \Phi^{(n)}_{P,\alpha}(\emptyset)$ from definitions 4.2 and 4.6. □

Theorem 4.3 (Completeness of abstract SLD) *Let* $\mathcal{A}$ *be an abstract atom. If* $\mathcal{A} \in lfp(\Phi_{P,\alpha})$*, then* $\mathcal{A}$ *has an* $\underline{\alpha}$*-SLD refutation.*

Proof: The proof has been omitted since it could be done by modifying the proof of theorem 3.2. □.

Theorems 4.2 and 4.3 state that the set of all abstract atoms that may be resolved by the abstract SLD resolution procedure is the same as the abstract fixpoint semantics. That is, the abstract SLD resolution gives an abstract procedural semantics for a definite logic program. It is to be noted that the abstract SLD resolution procedure differs from the standard SLD resolution procedure only in that it uses partial unification to unify two abstract terms while the standard resolution procedure uses standard unification to unify two concrete terms. By definition 4.1 and lemma 4.1, the unification of two abstract terms can be done by unifying two concrete terms using partial unification.

An abstract interpretation is obtained by defining an abstract domain, finding an stable abstract function α from the concrete domain to the abstract domain and designing a partial unification algorithm. The corresponding abstract SLD resolution can then be used to decide whether an abstract atom is included in the fixpoint semantics or not.

5 Depth abstractions

In this section, we illustrate how to approximate definite logic programs through depth abstractions. Sato *et al.* use depth abstractions to represent the success patterns of definite logic programs [11]. We prove the appropriateness of our framework for depth abstractions and show how an abstract SLD resolution can be used when it is necessary to increase the depth.

Definition 5.1 *Let* $t = f(x_1, x_2, \ldots, x_n)$ *be a term.* t *is a depth 0 sub-term of* t *and* t' *is a depth* k *sub-term of* t *if* t' *is a depth* $(k-1)$ *sub-term of* x_i *for some* $i \in \{1..n\}$*.*

A term is a depth k *abstract term if all its depth* k *sub-terms are distinct variables and these variables* *do not occur at any level less than k. Each variable at level k will be denoted as an $\_$.*

A depth k abstract term t is canonical if t contains no variables at any level less than k.

Definition 5.2 (depth k abstraction) *Let t be a term. The depth k abstraction of t, denoted by $d_k(t)$, is obtained by replacing each depth k sub-term of t with an $\_$. Each occurrence of $\_$ corresponds to a distinct variable. It is to be noted that the term thus obtained is a depth k abstract term.*

The depth 2 abstraction of $f(g(X,Y),g(h(Z)))$ is $f(g(\_,\_),g(\_))$, and its depth 3 abstraction is $f(g(X,Y),g(h(\_)))$. $f(g(\_,\_),g(\_))$ is a depth 2 abstract term.

The depth k abstractions of the Herbrand base B_P, the Herbrand Universe U_P and the minimum model M_P, *i.e.* $d_k(B_P)$, $d_k(U_P)$ and $d_k(M_P)$ respectively, are all sets of canonical depth k abstract terms.

Let us first establish the appropriateness of our method by the following lemma.

Lemma 5.1 *For any $k \geq 0$, d_k is stable.*

Proof: The proof is done by induction on the depth of abstraction k.

The result holds for $k = 0$ since $d_0(t\theta) = \_$ for any substitution θ and any term t.

Now suppose that the result hold for $k = n-1$. Let $d_n(t_1) = d_n(t_2)$. There are two cases to consider: (a). $t_1 = t_2 = X$, X is a variable; (b). $t_1 = f(x_1,\ldots,x_m)$ and $t_2 = f(y_1,\ldots,y_m)$. For case (a), the result holds since $t_1 = t_2$ implies $t_1\theta = t_2\theta$ for any θ. For case (b), we have $d_{n-1}(x_j) = d_{n-1}(y_j)$ for any $1 \leq j \leq m$ since $d_n(f(x_1,\ldots,x_m)) = d_n(f(y_1,\ldots,y_m))$. By the induction hypothesis, we have $d_{n-1}(x_j\theta) = d_{n-1}(y_j\theta)$ for any θ. So, $d_n(f(x_1,\ldots,x_m)\theta) = d_n(f(y_1,\ldots,y_m)\theta)$ for any θ. This completes the induction and hence the proof of the lemma. $\Box$

Now we give a partial unification algorithm for depth k abstraction and prove its correctness. The following algorithm for d_k-unification results from modifying Robinson's unification algorithm [10]. The ancillary function $occur(k,X,t)$ is true iff X occurs in t at any depth $j < k$.

Algorithm 5.1 (d_k-unification algorithm)
Given two terms t_1 and t_2 to be partially unified and the depth k, this algorithm decides if t_1 and t_2 are d_k-unifiable and, if unifiable, returns a most general d_k-unifier of t_1 and t_2.

01 **function** *du(k,t_1,t_2) $\Rightarrow$ (uni, σ)*
02 **begin**
03 **if** $k = 0$ *then* $(uni,\sigma) \leftarrow (true,\emptyset)$
04 **else if** t_1 *or* t_2 *is a variable* **then**
05 **begin**
06 **let** X *be the variable*
07 **let** t *be the other term*
08 **if** $X = t$ **then** $(uni,\sigma) \leftarrow (true,\emptyset)$
09 **else if** $occur(k,X,t)$ **then** $uni \leftarrow false$
10 **else** $(uni,\sigma) \leftarrow (true,\{X \leftarrow d_k(t)\})$
11 **end else**
12 **begin**
13 **let** $t_1 = f(x_1,\ldots,x_n)$
14 **let** $t_2 = g(x_1,\ldots,x_m)$
15 **if** $f \neq g$ *or* $m \neq n$ **then** $uni \leftarrow false$ **else**
16 **begin** $j \leftarrow 0, \quad (uni,\sigma_0) \leftarrow (true,\emptyset)$
17 **while** $j < m$ *and uni* **do**
18 **begin** $j \leftarrow j+1$
19 $(uni,\tau_j) \leftarrow du(k-1, x_j\sigma_{j-1}, y_j\sigma_{j-1})$
20 **if** *uni* **then** $\sigma_j \leftarrow \sigma_{j-1}\tau_j$
21 **end**
22 $\sigma \leftarrow \sigma_m$
23 **end**
24 **end**
25 **return** (uni,σ)
26 **end**

Lemma 5.2 (d_k-unification correctness)
Let t_1 and t_2 be terms. If t_1 and t_2 are d_k-unifiable, then algorithm 5.1 terminates and gives the unique d_k-$mgu(t_1,t_2)$. Otherwise, the algorithm terminates and reports the fact.

Proof: The algorithm always terminates since the first argument decreases by 1 whenever a recursive invocation occurs. If t_1 and t_2 are not d_k-unifiable then the algorithm certainly stops at line 09 or 15.

Now we show that if t_1 and t_2 are d_k-unifiable, then the algorithm finds a d_k-*mgu*. This is done by induction on the depth of abstraction k.

The result holds for $k = 0$ because $\emptyset$ is the d_0-mgu of any two terms since the depth 0 abstraction of any term t is $\_$ and we have $\sigma = \emptyset\sigma$ for any σ.

Now assume that the result holds for $k = n-1$. Suppose that t_1 and t_2 be d_n-unifiable. There are two cases to consider: (a). one of t_1 and t_2 is a variable; (b) both of them are functions with the same main functor and arity.

First consider case (a). If $t_1 = t_2$, line 08 gives the answer d_n-$mgu(t_1,t_2) = \emptyset$. Otherwise, $t_1 \neq t_2$. Let X be the variable and t the other term. Line 10 gives the answer d_n-$mgu(t_1,t_2) = \{X \leftarrow d_n(t)\}$, because $occur(n,X,t)$ cannot be true since t_1 and t_2 are d_n-unifiable. Thus, the result holds for case (a).

Now consider case (b). Let $t_1 = f(x_1,\ldots,x_m)$ and $t_2 = f(y_1,\ldots,y_m)$. Lines 16-23 in algorithm 5.1 result in $\sigma_0,\sigma_1,\ldots,\sigma_m$. We now prove that $\sigma_m = d_n$-$mgu(t_1,t_2)$. It suffices to prove that for any d_n-unifier θ of t_1 and t_2, and any $1 \leq j \leq m$ there exists a θ_j such that $\theta \simeq_{d_n,t'} \sigma_j\theta_j$, where $t' = t_1$ or $t' = t_2$.

For $j = 0$, let $\theta_0 = \theta$. We have $\theta = \sigma_0\theta_0$ since $\sigma_0 = \emptyset$. So, $\theta \simeq_{d_n,t'} \sigma_0\theta_0$.

Suppose that $\theta \simeq_{d_n,t'} \sigma_i\theta_i$. $x_{i+1}\sigma_i$ and $y_{i+1}\sigma_i$ are d_{n-1}-unifiable since t_1 and t_2 are d_n-unifiable. We have $\tau_{i+1} = d_{n-1}$-$mgu(x_{i+1}\sigma_i, y_{i+1}\sigma_i)$ and $\sigma_{i+1} = \sigma_i\tau_{i+1}$. Without lose of generality, we assume that τ_{i+1} is the form of $\{V \leftarrow t\}$. Let V be the depth h sub-term of either t_1 or t_2 and t be the depth h sub-term of the other for some h.

We now define $\theta_{i+1} = \theta_i - \{V \leftarrow V\theta_i\}$. If θ_i has a binding for V, then $\theta_i = \theta_{i+1} \cup \{V \leftarrow V\theta_i\}$. $\theta_i \simeq_{d_n,t'} \theta_{i+1} \cup \{V \leftarrow t\theta_i\}$ since $d_{n-h}(V\theta_i) = d_{n-h}(t\theta_i)$ and both V and t are depth h sub-terms. $\theta_i \simeq_{d_n,t'} \theta_{i+1} \cup$

$\{V \leftarrow t\theta_{i+1}\}$ since $occur(n-h, V, t)$ is not true. So, $\theta_i \simeq_{d_n,t'} \{V \leftarrow t\}\theta_{i+1} = \tau_{i+1}\theta_{i+1}$.

If θ_i does not have a binding for V, then $\theta_{i+1} = \theta_i$ and $\theta_i \simeq_{d_n,t'} \{V \leftarrow t\}\theta_{i+1} = \tau_{i+1}\theta_{i+1}$, since $d_{n-h}(V\theta_i) = d_{n-h}(t\theta_i)$ and both V and t are depth h sub-terms.

Thus, $\theta \simeq_{d_n,t'} \sigma_i\theta_i \simeq_{d_n,t'} \sigma_i\tau_{i+1}\theta_{i+1} = \sigma_{i+1}\theta_{i+1}$ and hence there exists a θ_m such that $\theta \simeq_{d_n,t'} \sigma_m\theta_m$. So, $\sigma_m = d_n\text{-}mgu(t_1, t_2)$. This completes the proof for case (b).

The uniqueness results from the fact that the proof above shows $t'\sigma_m$ subsumes $t'\theta$ for any other d_k-unifier θ of t_1 and t_2. $\square$

We illustrate the theoretical results by the following example.

Example 5.1 *Let $\alpha = d_2$ and*

$$P = \left\{ \begin{array}{l} a(f(susan)).\quad b(f(h(susan))). \\ p(x) : -a(x), b(x). \end{array} \right\}$$

We have

$$d_2(B_P) = \left\{ \begin{array}{l} a(susan), a(f(\_)), a(h(\_)), \\ b(susan), b(f(\_)), b(h(\_)), \\ p(susan), p(f(\_)), p(h(\_)) \end{array} \right\}$$

$$\Phi^0_{P,d_2}(\emptyset) = \emptyset$$

$$\Phi^1_{P,d_2}(\emptyset) = \{a(f(\_)), b(f(\_))\}$$

$$\Phi^2_{P,d_2}(\emptyset) = \{a(f(\_)), b(f(\_)), p(f(\_))\}$$

$$\Phi^3_{P,d_2}(\emptyset) = \{a(f(\_)), b(f(\_)), p(f(\_))\}$$

$$lfp(\Phi_{P,d_2}) = \Phi^3_{P,d_2}(\emptyset) = \{a(f(\_)), b(f(\_)), p(f(\_))\}$$

while $d_2(M_P) = d_2(lfp(T_P)) = \{a(f(\_)), b(f(\_))\}$. So, $lfp(\Phi_{P,d_2}) \neq d_2(M_P)$ for the program. $p(f(\_)) \in lfp(\Phi_{P,d_2})$ because of the partial unification.

Suppose now we want to be more precise and decide to compute $lfp(\Phi_{P,d_3})$. It can be noted that the set of ground atoms that $lfp(\Phi_{P,d_3})$ approximates is a subset of the set of ground atoms that $lfp(\Phi_{P,d_2})$ approximates. Instead of computing the least fixpoint as defined in definition 2.4, we choose to compute $lfp(\Phi_{P,d_3})$ by a generate-and-test method in order to demonstrate the use of abstract SLD resolution. First we generate a set of candidate elements for $lfp(\Phi_{P,d_3})$ and then use abstract SLD resolution to eliminate false candidates. From each element in $lfp(\Phi_{P,d_2})$, we generate a set of candidates by substituting each occurrence of $\_$ with each element from $d_1(U_P) = \{susan, f(\_), h(\_)\}$. Thus, the set of candidates is

$$\left\{ \begin{array}{l} a(f(susan)), a(f(f(\_))), a(f(h(\_))), \\ b(f(susan)), b(f(f(\_))), \\ b(f(h(\_))), \\ p(f(susan)), p(f(f(\_))), p(f(h(\_))) \end{array} \right\}$$

After eliminating false candidates, we have

$$lfp(\Phi_{P,d_3}) = \{a(f(susan)), b(f(h(\_)))\}$$

which is the same as $d_3(M_P)$.

The $p(f(susan))$ has been eliminated by the following process. First, $p(f(susan))$ is resolved with the clause $p(x) : -a(x), b(x)$ resulting in $a(f(susan)), b(f(susan))$. Then subgoal $a(f(susan))$ is resolved with the unit clause $a(f(susan))$. However, $b(f(susan))$ cannot be resolved with $b(f(h(susan))$ because $d_3(b(f(susan))) = d_3(b(f(susan)))$ while $d_3(b(f(h(susan)))) = b(f(h(\_)))$.

6 Related work and further work

We have defined a class of abstract fixpoint semantics of definite programs and given a procedural characterisation of the abstract fixpoint semantics. A particular program analysis is obtained by identifying an abstract domain, an stable abstraction function and designing a partial unification algorithm. The requirement that the abstraction function is stable is reasonable because each element in the abstract domain is meant to represent a set of elements in the concrete domain sharing a certain property, i.e., the abstract domain partitions the concrete domain. It is natural that an identical instantiation process, when applied to terms of an equivalent class, should result in terms that belong to the same equivalent class.

In the frameworks proposed by Mellish [9], Jones *et al.* [5] and Bruynooghe [2], it is necessary to provide three to five ancillary functions for a program analysis and to prove the correctness of the functions. We only need a partial unification algorithm to be provided and proved. It simplifies the design of the program analysis methods.

The concrete semantics we have considered is the fixpoint semantics of definite logic programs [13]. The results in this paper would appear to be readily generalised resulting in a class of abstract fixpoint semantics of normal programs and corresponding partial SLDNF resolution procedures and abstract SLDNF procedures by using the concrete semantics of normal programs such as that in [7]. The authors also believe that the results in this paper can be generalised to give a procedural characterisation of the abstract interpretation of collecting semantics since the abstract SLD procedure can be used to resolve both abstract atoms and abstract goals.

References

[1] M. Bruynooghe and G. Janssens. An instance of abstract interpretation integrating type and mode inferencing. In R.A. Kowalski and K.A. Bowen, editors, *Proceedings of the fifth International Conference and Symposium on Logic Programming*, pages 669–683. The MIT Press, 1988.

[2] 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.

[3] P. Cousot and R. Cousot. Abstract interpretation: a unified framework for static analysis of

programs by construction or approximation of fixpoints. In *Proceedings of the fourth annual ACM symposium on Principles of programming languages*, pages 238–252, Los Angeles, California, 1977.

[4] P. Cousot and R. Cousot. Systematic design of program analysis frameworks. In *Proceedings of the sixth annual ACM symposium on Principles of programming languages*, pages 269–282, San Antonio, Texas, 1979.

[5] N.D. Jones and H. Søndergarrd. 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.

[6] J.W. Lloyd. *Foundations of Logic Programming*. Springer-Verlag, 1987.

[7] K. Marriott and H. Søndergaard. Bottom-up abstract interpretation of logic programs. In R.A. Kowalski and K.A. Bowen, editors, *Proceedings of the fifth International Conference and Symposium on Logic Programming*, pages 733–748. The MIT Press, 1988.

[8] C. Mellish. Some global optimisations for a prolog compiler. *The Journal of Logic Programming*, 2(1):43–66, 1985.

[9] C. Mellish. Abstract interpretation of prolog programs. In S. Abramsky and C. Hankin, editors, *Abstract interpretation of declarative languages*, pages 181–198. Ellis Horwood Limited, 1987.

[10] J. A. Robinson. A machine-oriented logic based on the resolution principle. *Journal of the ACM*, 12(1):23–41, 1965.

[11] T. Sato and H. Tamaki. Enumeration of success patterns in logic programs. *Theoretical Computer Science*, 34(1):227–240, 1984.

[12] H. Søndergaard. An application of abstract interpretation of logic programs: occur check problem. In **LNCS 213** *Proceedings of the European symposium on programming*. Springer-Verlag, 1986.

[13] M.H. van Emden and R.A. Kowalski. The semantics of predicate logic as a programming language. *Artificial Intelligence*, 23(10):733–742, 1976.