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 \( \alpha \) 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 \( \sigma \). For each such abstraction function \( \alpha \), an abstract fixpoint semantics of a definite logic program \( P \) is given as the least fixpoint of a function \( \Phi_{P, \alpha} \). The soundness of using \( \Phi_{P, \alpha} \) as an approximation of van Emden and Kowalski's \( \Phi_P \) 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.

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 [6, 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 \( \Phi_P : \wp(B_P) \mapsto \wp(B_P) \) defined as follows.

\[
\Phi_P(I) = \begin{cases} 
A_0 \sigma | \exists \sigma . A_0 \sigma \leftarrow A_1, \ldots, A_m \sigma \in P, A_0 \sigma \in B_P \end{cases}
\]  

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 Terms be the set of terms, \( \alpha \) be a function from Terms to Terms, called the set of abstract terms. The concrete domain we consider is a sub-lattice of \((p(\text{Terms}), \subseteq)\) and the abstract domain a sub-lattice of \((p(Terms), \subseteq)\).

The abstraction function \( \alpha \) induces an equivalence relation \( \equiv_{\alpha} \) on Terms, 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 \( \alpha \)-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 \( \theta_1 \) and \( \theta_2 \) such that \( \theta_1 t \equiv_{\alpha} \theta_2 t \).

**Definition 1.3** The fact that \( \theta_1 t \equiv_{\alpha} \theta_2 t \) is denoted as \( \theta_1 \equiv_{\alpha} \theta_2 \).

We generalise \( \alpha \) to be a function \( \alpha : p(\text{Terms}) \rightarrow p(\text{Terms}) \) as \( \alpha(S) = \{ \alpha(s) | s \in S \} \). We define a concretisation function \( \gamma : \text{Terms} \rightarrow p(\text{Terms}) \) as \( \gamma(t) = \{ t | t \in \text{Terms}, t = \alpha(t) \} \) and generalise \( \gamma \) to be \( \gamma : p(\text{Terms}) \rightarrow p(\text{Terms}) \) 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 \( \alpha \), denoted by \( \alpha \)-unification. In standard unification, two terms \( t_1 \) and \( t_2 \) are considered unifiable if there exists a substitution \( \sigma \) such that \( t_1 \sigma = t_2 \sigma \). In \( \alpha \)-unification, the identity relation \( \equiv_{\alpha} \) 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 \( \sigma \) be a substitution. \( \sigma \) is called an \( \alpha \)-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 \( \alpha \)-unifiable if they have at least one \( \alpha \)-unifier.

A term \( t \) is said to subsume a term \( s \) if there is a substitution \( \sigma \) such that \( t \sigma \equiv_{\alpha} s \).

**Definition 2.2** An \( \alpha \)-unifier \( \sigma \) of term \( t_1 \) and term \( t_2 \) is said to be most general if for any other \( \alpha \)-unifier \( \sigma' \) of term \( t_1 \) and \( t_2 \), \( \tau \sigma' \) is not subsumed by \( \tau \sigma \), where \( \tau \alpha \sigma' \) is \( \alpha \)-unifiable if they have at least one \( \alpha \)-unifier.

**Definition 2.3** An abstraction function is called stable, \( t \sigma \), if for any substitution \( \theta \) 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 \( \alpha \) is stable, then:

1. For any term \( t_1 \), any ground term \( t_2 \), and any \( \theta \), if \( t_1 \equiv_{\alpha} t_2 \) then \( t_1 \theta \equiv_{\alpha} t_2 \theta \).
2. If \( t_1 \in \gamma(t_1) \) and \( t_2 \in \gamma(t_2) \) are \( \alpha \)-unifiable with an \( \alpha \)-unifier \( \sigma \), then, for any \( t'_1 \in \gamma(t_1) \) and any \( t'_2 \in \gamma(t_2) \), \( t'_1 \sigma \) and \( t'_2 \sigma \) are \( \alpha \)-unifiable and \( \sigma \) is an \( \alpha \)-unifier of \( t'_1 \sigma \) and \( t'_2 \sigma \).
3. If a term \( t \) is \( \alpha \)-unifiable with a term \( t_1 \in \gamma(t_1) \) with \( \alpha \)-mgu \( \sigma \), then \( t \) is \( \alpha \)-unifiable with any term \( t'_1 \in \gamma(t_1) \) with \( \alpha \)-mgu \( \sigma \).

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

Now consider (b). By definition 2.3, \( t_1 \sigma \equiv_{\alpha} t_2 \sigma \) and \( t_2 \sigma \equiv_{\alpha} t_2 \sigma \). So, \( t_1 \sigma \equiv_{\alpha} t_2 \sigma \) since \( t_1 \sigma \equiv_{\alpha} t_2 \sigma \).

**Definition 2.4** Let \( P \) be a definite logic program and \( \alpha(BP) \subseteq \text{Terms} \) be the \( \alpha \) abstraction of \( BP \). We define \( \Phi_{\alpha} : p(\alpha(BP)) \rightarrow p(\alpha(A\sigma(BP))) \) as follows.

\[
\Phi_{\alpha}(\{ A_{0} \}) = \{ \alpha(A_{0}), \alpha(A_{1}) \in BP, \ldots, \alpha(A_{m}) \in BP \}
\]

**Definition 2.5**

\[
\Phi_{\alpha}(I) = \{ A \in\{ A_{0}, \ldots, A_{m} \} \in I \}
\]

In this definition, we have relaxed the condition that the substitution \( \sigma \) makes the clause ground. The condition \( A \in \alpha(\{ A_{0}, \ldots, A_{m} \}) \) guarantees \( \Phi_{\alpha}(I) \subseteq \alpha(BP) \). We now prove that \( \Phi_{\alpha} = \Phi_{\alpha} \).

**Lemma 2.2** \( \Phi_{\alpha} = \Phi_{\alpha} \).

**Proof:** It suffices to show that \( \Phi_{\alpha}(I) \subseteq \Phi_{\alpha}(I) \) for any \( I \). This is done by proving \( \Phi_{\alpha}(I) \subseteq \Phi_{\alpha}(I) \) and \( \Phi_{\alpha}(I) \subseteq \Phi_{\alpha}(I) \) separately.

First, let \( A \in \Phi_{\alpha}(I) \). By definition 2.4, there exist some \( \alpha \) and some \( C = A_{0} \rightarrow A_{1}, \ldots, A_{m} \in BP \) such that \( C \sigma \) is ground, \( \alpha(A_{j} \sigma) \in \Phi_{\alpha}(I) \) for \( j \in \{ 1, \ldots, m \} \) and \( A = \alpha(A_{0} \sigma) \). Since \( A \sigma \) is ground, we have \( \{ A_{0} \sigma \} \subseteq \{ A_{0} \sigma \} \) and hence \( \alpha(\{ A_{0} \sigma \}) = \{ A_{0} \sigma \} \) and \( \alpha(\{ A_{0} \sigma \}) = \{ A \} \). Thus, \( A \in \alpha(\{ A_{0} \sigma \}) \), which, together with \( \alpha(A_{j} \sigma) \in \Phi_{\alpha}(I) \) for \( j \in \{ 1, \ldots, m \} \), implies that \( A \in \Phi_{\alpha}(I) \). So, \( \Phi_{\alpha}(I) \subseteq \Phi_{\alpha}(I) \).
Now suppose that \( A \in \Phi(p_{A}(I)) \). By definition 2.5, there exist some \( \tau \) and some \( C = A \rightarrow A_0 \rightarrow \ldots \rightarrow A_m \in p \) such that \( \alpha(A_j) \in \Phi(p_{A}(I)) \) for \( j \in \{1, \ldots, m\} \) and \( A \in \alpha(A_0) \). Let \( T \) be the substitution such that \( A_0 \tau = B \) and \( A = \alpha(A_0 \tau) \). Then, for any \( j \in \{1, \ldots, m\} \), \( \alpha(A_j \tau) \in I \) since \( A_j \in I \) and \( \tau \subseteq A_0 \). Let \( \tau \) be such that \( A_0 \tau \in (Bp) \). Let \( \tau \subseteq A_0 ] \). If \( C \tau \) is not ground, then \( \tau \) has an \( \tau \) such that \( C \tau \tau' \) is ground. We have \( \alpha(A_j \tau' \) \) and \( A = \alpha(A_0 \tau' \) \). So, \( A \in \Phi(p_{A}(I)) \) and hence \( \Phi(p_{A}(I)) \subseteq \Phi(p_{A}(I)) \). This completes the proof of the lemma. \( \square \)

Lemma 2.2.2 and lemma 2.2 allow the use of the \( \alpha \)-unification algorithm in the computation of \( \Phi(p_{A}(I)) \) can be computed with the help of \( \alpha \)-unification. That is, \( \forall \tau \in \{1, \ldots, m\} \), \( \alpha(A_j \tau) \in I \) can be checked by verifying that \( A_j \in I \) and \( \tau \subseteq A_0 \). Let \( \tau \subseteq A_0 \). If \( C \tau \) is not ground, then \( \tau \) has an \( \tau \) such that \( C \tau \tau' \) is ground. We have \( \alpha(A_j \tau' \) \) and \( A = \alpha(A_0 \tau' \) \). So, \( A \in \Phi(p_{A}(I)) \) and hence \( \Phi(p_{A}(I)) \subseteq \Phi(p_{A}(I)) \). This completes the proof of the lemma. \( \square \)

Lemma 2.3 For any \( T \in p(\alpha(B)) \), \( T(\gamma(I)) \subseteq \gamma(\Phi(p_{A}(I))) \).

Proof: Let \( A \in T(\gamma(I)) \). Then, by definition 1.1, there exist a clause \( A \rightarrow A_0 \rightarrow \ldots \rightarrow A_m \) and some \( \tau \) such that \( A_0 \tau \in Bp \), \( A = A_0 \tau \) and \( A_j \tau \in \gamma(I) \), \( \tau \subseteq A_0 \). Let \( \tau \subseteq A_0 \). If \( C \tau \) is not ground, then \( \tau \) has an \( \tau \) such that \( C \tau \tau' \) is ground. We have \( \alpha(A_j \tau' \) \) and \( A = \alpha(A_0 \tau' \) \). So, \( A \in \Phi(p_{A}(I)) \). By the definition of \( \gamma \), \( A \in \gamma(\Phi(p_{A}(I))) \). So, \( T(\gamma(I)) \subseteq \gamma(\Phi(p_{A}(I))) \). \( \square \)

Lemma 2.3 states that \( \Phi(p_{A}(I)) \) safely approximates \( Tp \). That is, atomic ground goals outside \( \gamma(\Phi(p_{A}(I))) \) cannot succeed.

Lemma 2.4

\[
\forall i \geq 0 \forall A(A \in T^n(0) \Rightarrow \alpha(A) \in \Phi(p_{A}(0)))
\]

Proof: The proof is by induction on \( i \). First, the result holds for \( i = 0 \), since \( T^n(0) = \emptyset \). Next assume that the result holds for \( i < n \). Let \( A \in T^{n+1}(0) \). From definition 1.1, there exist \( A_0 \rightarrow A_1 \rightarrow \ldots \rightarrow A_m \) and \( \tau \) such that \( A = A_0 \tau \) and \( A_j \tau \in T^n(0) \) for \( j \in 1, \ldots, m \). By the induction assumption, \( A_j \tau \) is a-unifiable with some \( \alpha(A_j) \in \Phi(p_{A}(0)) \). Thus, the result holds for \( n + 1 \) and this completes the proof of the lemma. \( \square \)

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

\[
\alpha(MP) \subseteq \Phi(p_{A}(I))
\]

Proof: \( \alpha(M_P) = \alpha(lfp(T)) = \alpha(u_{i=0}^{\infty} T^i(0)) \). This and lemma 2.4 imply that \( \alpha(M_P) \subseteq u_{i=0}^{\infty} \Phi(p_{A}(0)) = \Phi(p_{A}(0)) \). \( \square \)

Lemma 2.5 states that \( lfp(\Phi(p_{A})) \) is an abstract interpretation of \( MP \).

Lemma 2.6 \( lfp(\Phi(p_{A})) = \alpha(MP) \) 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 \( \alpha \)-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 \( \alpha \)-SLD can derive is the same as \( lfp(\Phi(p_{A})) \).

Definition 3.1 Let \( G \) be a definite goal. If \( P \in \mathbb{A}(A) \), and \( C = B \rightarrow \ldots \rightarrow B_n \), called the input clause, be a \( \alpha \)-derivation of \( G \). Then \( G' \) is a \( \alpha \)-derivation of \( G \) using \( \alpha \)-mu \( \sigma \) if the following conditions hold:

\( a \) \( A_0 \) is an atom, called the selected atom, in \( G \); \( b \) \( \sigma \) is an \( \alpha \)-mu of \( A_j \) and \( B_0 \); \( c \) \( G' = \langle A_1, \ldots, A_{m+1}, B_0, B_1, \ldots, B_n \rangle \sigma \)

Definition 3.2 An \( \alpha \)-SLD derivation of \( PU(G) \) consists of a sequence \( G_0, G_1, \ldots \) of goals, a sequence \( C_1, C_2, \ldots \) of program clauses and a sequence \( \sigma_1, \sigma_2, \ldots \) of \( \alpha \)-mu's such that each \( G_i + 1 \) is \( \alpha \)-derived from \( G_i \) and \( C_{i+1} \) using \( \sigma_{i+1} \).

Definition 3.3 An \( \alpha \)-SLD refutation of \( PU(G) \) is a finite \( \alpha \)-SLD derivation of \( P \cup G \) with the empty goal \( \boxempty \) as the last goal in the derivation. If \( G_n = \boxempty \), then the refutation has length \( n \).

Definition 3.4 An unrestricted \( \alpha \)-SLD refutation is an \( \alpha \)-SLD refutation except that we drop the requirement that the substitutions \( \sigma_i \) be \( \alpha \)-mu's. They only need to be \( \alpha \)-unifiers.

Lemma 3.1 Let \( P \) be a definite logic program and \( G \) be a definite goal. If \( PU(G) \) has an unrestricted \( \alpha \)-SLD refutation, then \( PU(G) \) has an \( \alpha \)-SLD refutation such that if \( \theta_1, \ldots, \theta_n \) are the \( \alpha \)-unifiers for the unrestricted \( \alpha \)-SLD refutation and \( \theta'_1, \ldots, \theta'_n \) are \( \alpha \)-mu's for the \( \alpha \)-SLD refutation, then there exists some \( \lambda \) such that \( \theta_1, \ldots, \theta_n \cong \theta'_1, \ldots, \theta'_n \).

Proof: The proof is by induction on the length of the unrestricted \( \alpha \)-SLD refutation.

Suppose first that \( n = 1 \). Thus \( PU(G) \) has an unrestricted \( \alpha \)-SLD refutation \( G_0 = G, G_1 = \boxempty \) with
input atom in $A.$ Furthermore, $P \cup \{G\}$ has an a-SLD refutation $G_0 = G, G_1 = \Box$ with input clause $C_1$ and a-mgu $\theta_1.$

Next suppose that the result holds for $n=1.$ Let $P \cup \{G\}$ have an unrestricted a-SLD refutation $G_0 = G, G_1, \ldots, G_n = \Box$ of length $n$ with input clauses $C_1, \ldots, C_n$ and a-unifiers $\theta_1, \ldots, \theta_n.$ There exists an a-mgu $\theta'_1$ for the selected atom in $C_1$ such that $\theta'_1 \equiv G, G_1, \ldots, G_n \in \Box$ and $\Box$ has a length $n$ and $\Box$ is a-unifiable with an a-unifier $\theta'_{i, 2}, \ldots, \theta'_{i, \lambda}$ for some $\lambda.$ Thus, $P \cup \{G\}$ has an a-SLD refutation $G'_0 = G, G'_1, \ldots, G'_n = \Box$ with input clauses $C_1, \ldots, C_n$ and a-unifiers $\theta'_1, \theta'_2, \ldots, \theta'_{i, \lambda}$ such that $\theta_1, \ldots, \theta_n, \theta'_1, \theta'_2, \ldots, \theta'_{i, \lambda} \equiv G, G_1, \ldots, G_n \in \Box.$

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 a-SLD refutation of length $n$ and $\sigma_1, \ldots, \sigma_n$ be a-mgu's, then $\forall j \in \{1, m\}, \alpha([A_1], [\ldots, \sigma_n]) \subseteq \Phi_{P,a}(\sigma).$

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

Suppose first that $n = 1.$ Then $G$ must be the form of $\neg A_1$ and the program $P$ has a unit clause of form $A_1$. Let $\alpha$ be an a-SLD refutation of $P \cup \{G\}$ of length $n = 1.$ By the definition of $\alpha, \alpha([A_1]) \subseteq \Phi_{P,a}(\Box).$ So, $\alpha([A_1]) \subseteq \Phi_{P,a}(\Box)$ since $\alpha([A_1]) \equiv \alpha.$

Next suppose that the result holds for any a-SLD refutation of length $n-1.$ Consider an a-SLD refutation of $P \cup \{G\}$ of length $n.$ Let $A_1$ be an atom of $G.$ If $A_1$ is not the selected atom, then $A_1, \sigma_1$ is an atom of $G_1$, the second goal of the a-SLD refutation. $\alpha([A_1], [\ldots, \sigma_n]) \subseteq \Phi_{P,a}(\Box)$ by the induction hypothesis. So, $\alpha([A_1], [\ldots, \sigma_n]) \subseteq \Phi_{P,a}(\Box)$ by the monotonicity of $\Phi_{P,a}.$ If $A_1$ is the selected atom, letting $B = B_1, B_2, \ldots, B_q$ be the first input clause, then $A_1, \sigma_1 = B_1, \sigma_1$ if $q = 0.$ Let $\alpha([B_1]) \subseteq \Phi_{P,a}(\Box)$ by the definition of $\Phi_{P,a}.$ Thus, $\alpha([A_1], [\ldots, \sigma_n]) \subseteq \alpha([A_1], [\ldots, \sigma_n]) \subseteq \alpha([B_1]) \subseteq \Phi_{P,a}(\Box).$ If $q > 0,$ by the induction hypothesis, $\alpha([B_1, [\ldots, \sigma_n]) \subseteq \Phi_{P,a}(\Box).$ For $i = 1, 2, \ldots, q.$ By the definition of $\Phi_{P,a},$ we have that $\alpha([A_1, [\ldots, \sigma_n]) \subseteq \Phi_{P,a}(\Box).$ Because $\alpha$ is stable, we have $\alpha([A_1, [\ldots, \sigma_n]) \subseteq \Phi_{P,a}(\Box)$ since $A_1, \sigma_1 = B_1, \sigma_1.$

Theorem 3.2 (Completeness of partial SLD) If $\alpha(A) \subseteq \Phi_{P,a}(\Box),$ then $A$ has an a-SLD refutation.

Proof: It suffices to show that for any atom $A$ and any $n, \alpha(A) \subseteq \Phi_{P,a}(\Box)$ implies that $A$ has an a-SLD refutation. The result is proved by induction on $n.$

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

Next suppose that the result is true for $n-1.$ That $\alpha(A) \subseteq \Phi_{P,a}(\Box)$ implies that there exist some $\sigma$ and some clause $C = A_0 - A_1, \ldots, A_n$ such that $C$ is ground, $A \equiv A_0 \sigma$ and $\alpha(A_1 \sigma), \ldots, \alpha(A_n \sigma) \in \Phi_{P,a}(\Box).$ The induction hypothesis implies that $A_1 \sigma$ has an a-SLD refutation for $j = 1, \ldots, q.$ So, $(A_1, \ldots, A_q) \sigma$ has an a-SLD refutation since $A_1 \sigma$ is ground for $j = 1, \ldots, q.$ That $A_0 \sigma$ is ground and $A \equiv A_0 \sigma$ implies that $A \sigma \equiv A_0 \sigma.$ Thus $A$ has an unrestricted a-SLD refutation with $G_0 = A, G_1 = (A_1, \ldots, A_q) \sigma$ and input clause $C_1 = C$ and $\sigma = \sigma.$ Hence, $A$ has an a-SLD refutation by lemma 3.1.

Lemma 3.2 If $A \in E(A)$ and $P \cup \{A\}$ has an a-SLD refutation of length $n$ with input clauses $C_1, C_2, \ldots, C_n$ of variants of program clauses and $\alpha(A_1), \ldots, \alpha(A_n)$ of input clauses $C_1, C_2, \ldots, C_n$ of variants of program clauses and $\alpha(A_1), \ldots, \alpha(A_n)$ with input clauses $C_1, C_2, \ldots, C_n$ of variants of program clauses and $\alpha(A_1), \ldots, \alpha(A_n)$

Proof: Since, $\sigma_1$ is the a-mgu of $A$ and the head of $C_1$ is the a-mgu of $A'$ and the head of $C_1$ by lemma 2.1.d. Thus, we have $G_1 \equiv G_1.$

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 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 $\alpha$-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 $\alpha$-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 $t_1$ and $t_2$ are $\alpha$-unifiable with an $\alpha$-unifier $\sigma$ iff there exist some $t_1 \in \gamma(t_1)$ and $t_2 \in \gamma(t_2)$ such that $\sigma$ is an $\alpha$-unifier of $t_1$ and $t_2.$ An $\alpha$-unifier of $t_1$ and $t_2$ is an $\alpha$-mgu of $t_1$ and $t_2,$ iff for any $\alpha$-unifier $\tau$ of $t_1$ and $t_2,$ there exists some $\theta$ such that $\sigma \equiv \alpha, \tau \theta,$ where $\theta = t_1$ or $\theta = t_2.$

Lemma 4.1 Two abstract terms $t_1$ and $t_2$ are $\alpha$-unifiable iff any element of $\gamma(t_1)$ is $\alpha$-unifiable with any element of $\gamma(t_2).$
Proof: The proof results from definition 4.1 and lemma 2.1.b.

Definition 4.2 \( \sigma t = t' \) iff \( t' = \alpha(\sigma t) \) for some \( t \in \gamma(t) \).

Definition 4.3 Let \( G \) be \( \neg A_1, \ldots, \neg A_p \) and \( C = B - B_1, \ldots, B_q \), called input clause, be a variant of a clause of \( P \). Then \( G' \) is \( \alpha \)-derived from \( G \) and \( C \) using \( \alpha \)-mgu \( \sigma \) if the following conditions hold:

\begin{itemize}
  \item \( A_j \) is an atom, called selected atom, in \( G \);
  \item \( \sigma \) is a \( \alpha \)-mgu of \( A_j \) and \( \alpha(B) \);
  \item \( G' = \neg (A_1, \ldots, A_{j-1}, \alpha(B_1), \ldots, \alpha(B_q), A_{j+1}, \ldots, A_p) \sigma \)
\end{itemize}

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

Definition 4.4 

\( \alpha(-A_1, \ldots, A_m) = -\alpha(A_1), \ldots, \alpha(A_m) \)

Definition 4.5 Let \( G \) be \( \neg A_1, \ldots, A_m \). We define 

\( \gamma(G) = \{ -A_1, \ldots, A_m | A_1 \in \gamma(A_1), \ldots, A_m \in \gamma(A_m) \} \)

Lemma 4.2 If \( G' \) is \( \alpha \)-derived from \( G \) and \( C \) using \( \alpha \)-mgu \( \sigma \), then \( \alpha(G') \) is \( \alpha \)-derived from \( \alpha(G) \) and \( C \) using \( \alpha \)-mgu \( \sigma \).

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

Theorem 4.1 \( G' \) is \( \alpha \)-derived from \( G \) if \( \alpha \) is the unifier of \( a(\alpha G) \) and \( B \in -B_1, \ldots, B_q \) iff for any \( G' \in \gamma(G') \), \( G' \) is \( \alpha \)-derived from \( G \) using \( \alpha \)-mgu \( \sigma \) and \( B' = -B_1, \ldots, B_q \).

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

Suppose that \( G' \) is \( \alpha \)-derived from \( G \). Let \( A_j \) be the selected atom. Then \( G' = \neg (A_1, \ldots, A_{j-1}, \alpha(B_1), \ldots, \alpha(B_q), A_{j+1}, \ldots, A_p) \sigma \)

Let \( G' \in \gamma(G) \), then \( G' = \neg A_1, \ldots, A_{j-1}, B_1, \ldots, B_q, A_{j+1}, \ldots, A_p \) and \( A_i \in \gamma(A_i) \) for \( i \in \{1..(j-1), (j+1..p}\} \). By the transitivity of \( \equiv \), \( A_i \equiv \alpha A_i \sigma \) for all \( A_i \in \gamma(A_i) \).

This completes the proof.

In parallel to definitions 3.2, 3.3 and 3.4, we can define the notions of \( \alpha \)-derivation and \( \alpha \)-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 

\( \{ A \} = \alpha(\gamma(A)) \)

Theorem 4.2 (Soundness of abstract SLD) Let \( P \) be a definite logic program and \( \mathcal{G} \) be a definite goal of form \( \neg A_1, \ldots, A_m \). If \( \mathcal{P} \cup \{ \mathcal{G} \} \) has an \( \alpha \)-SLD refutation of length \( n \) and \( \sigma_1, \ldots, \sigma_m \) be \( \alpha \)-mgu's, then 

\( \forall j \in \{1..m\}.(\{A_j \sigma_1 \ldots \sigma_n\} \subseteq \Phi_{P,\alpha}(\emptyset)) \)

Proof: By theorem 4.1, any \( G \in \gamma(G) \) has an \( \alpha \)-refutation of length \( n \) and \( \sigma_1, \ldots, \sigma_m \) are the \( \alpha \)-mgu's. Let \( G = \neg A_1, \ldots, A_m \). We have, \( G, A_1 \in \gamma(A_1), \ldots, A_m \in \gamma(A_m) \). By theorem 3.1, \( \forall j \in \{1..m\}.(\alpha([A_j \sigma_1 \ldots \sigma_n])) \subseteq \Phi_{P,\alpha}(\emptyset) \). So, \( \forall j \in \{1..m\}.(\{A_j \sigma_1 \ldots \sigma_n\} \subseteq \Phi_{P,\alpha}(\emptyset) \) from definitions 4.2 and 4.6.

Theorem 4.3 (Completeness of abstract SLD) Let \( A \) be an abstract atom. If \( A \in \Phi_P(\emptyset) \), then \( A \) has an \( \alpha \)-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 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.

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 \( a_k \).

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 \( a_k \). Each occurrence of \( a_k \) 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 \( f(g(\ldots, g(\ldots, g(\ldots))) \). f(\ldots, g(\ldots)) \) 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 \( M_P \), \( d_k(B_P) \), \( d_k(U_P) \) 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 > 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 \( d_0(t(x)) = \sigma \) for any substitution \( \sigma \) 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 \( \sigma = \tau \sigma \) for any \( \sigma \). 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 \( d_{n-1}(x_j) = d_{n-1}(y_j) \) for any \( \sigma \). So, \( d_n(f(x_1, \ldots, x_m)) = d_n(f(y_1, \ldots, y_m)) \) for any \( \sigma \). This completes the induction and hence the proof of the lemma.

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 if \( X \) 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 \( t_2 \).

```c
01 function du(t1, t2) \( \Rightarrow (\mu, \sigma) \)
02 begin
03 if k = 0 then (\( \mu, \sigma \)) \( \equiv (true, \emptyset) \)
04 else if t1 or t2 is a variable then
05 begin
06 let X be the variable
07 let t be the other term
08 if X = t then (\( \mu, \sigma \)) \( \equiv (true, \emptyset) \)
09 else if occurs\((k, X, t)\) then (\( \mu, \sigma \)) \( \equiv false \)
10 else (\( \mu, \sigma \)) \( \equiv (true, \{X \rightarrow d_k(t)\}) \)
11 end else
12 begin
13 let t1 = \( f(x_1, \ldots, x_n) \)
14 let t2 = \( g(x_1, \ldots, x_n) \)
15 if \( f \neq g \) or \( m \neq n \) then (\( \mu, \sigma \)) \( \equiv false \)
16 begin
17 j = 0
18 while j < m and \( \mu \neq \emptyset \) do
19 begin
20 j = j + 1
21 if \( \mu \neq \emptyset \) or \( j = 1 \) then
22 begin
23 (\( \mu, \sigma \)) \( \equiv d_k(k - 1, x_j, \sigma, y_j, \sigma, \ldots) \)
24 end
25 end
26 end
27 return (\( \mu, \sigma \))
28 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 returns the fact.

**Proof:** The algorithm always terminates since the first argument decreases by 1 whenever a recursion 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 \( \emptyset \). We have \( \sigma = \emptyset \) for any \( \sigma \).

Now assume that the result holds for \( k = n - 1 \). Suppose that \( t_1 \) and \( t_2 \) are \( d_k \)-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 \( d_k \)-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 \( d_k \)-mgu\((t_1, t_2) \) = \( \{X \rightarrow d_k(t)\} \), because occur\((n, X, t) \) cannot be true since \( t_1 \) and \( t_2 \) are \( d_k \)-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 11-16 in algorithm 5.1 result in \( \sigma_0, \sigma_1, \ldots, \sigma_m \). We now prove that \( \sigma_m = d_k \)-mgu\((t_1, t_2) \). It suffices to prove that for any \( d_k \)-unifier \( \theta \) of \( t_1 \) and \( t_2 \), and any \( 1 \leq j \leq m \) there exists a \( \theta_j \) such that \( \theta \equiv d_k(t_1, \theta_j) \), where \( \theta_j = t_j \) or \( \theta_j = t_2 \). For \( j = 0 \), let \( \theta_0 = \emptyset \). We have \( \theta = \sigma_0 \sigma_0 \) since \( \sigma_0 = \emptyset \). So, \( \theta \equiv d_k(t, \emptyset) \).

Suppose that \( \theta \equiv d_k(t, \emptyset) \). \( x_1, \sigma_1 \) and \( y_1, \sigma_2 \) are \( d_{n-1} \)-unifiable since \( t_1 \) and \( t_2 \) are \( d_k \)-unifiable. We have \( \tau_{n-1} = d_{n-1} \)-mgu\((x_1, \sigma_1, y_1, \sigma_1) \) and \( \sigma_1 = \sigma_1 \tau_{n-1} \). Without lose of generality, we assume that \( \tau_{n-1} \) is the form of \( \{V \rightarrow t\} \). 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 \( \theta_{n-1} = \theta_1 \). \( \theta_1 \) is \( \{V \rightarrow V \tau_{n-1} \}. \) If \( \theta_1 \) has a binding for \( V \), then \( \theta_1 = \theta_{n-1} \cup \{V \rightarrow V \theta_1 \} \). \( \theta_{n-1} \equiv d_k \theta_{n-1} \theta_{n-1} = \{V \rightarrow t \} \) since \( d_{n-1}(V \theta_1) = d_{n-1}(t) \) and both \( V \) and \( t \) are depth \( h \) sub-terms. \( \theta_1 \equiv d_k \theta_{n-1} \theta_{n-1} \cup \{V \rightarrow t \} \).
\{V \leftarrow \theta \delta_{i+1}\} \text{ since } \text{occurs}(n - h, V, t) \text{ is not true. So, } \\
\delta_i \subseteq \delta_{i+1}, \quad \{V \leftarrow t\} \theta_{i+1} = \tau_{i+1} \theta_{i+1}.

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

Thus, \theta \subseteq \delta_{i+1}, \quad \sigma \delta_i \subseteq \delta_{i+1}, \quad \sigma \tau_{i+1} \theta_{i+1} = \sigma \tau_{i+1} \theta_{i+1} \text{ and hence there exists a } \theta_m \text{ such that } \theta \subseteq \delta_{i+1}, \quad \sigma \theta_m.

So, \sigma_m = d_n - mgu(\delta_1, \delta_2). \text{ This completes the proof for case } (b).

The uniqueness results from the fact that the proof above shows that the program analysis should result in terms that belong to the same equivalent class.

We illustrate the theoretical results by the following example.

**Example 5.1** Let \( \alpha = d_3 \) and 

\[
P = \{ a(f(susan)), b(f(h(susan))) \} \text{ and } P(x) = -a(x), b(x).
\]

We have

\[
d_3(BP) = \{ a(susan), a(f(_)), a(h(_)), b(susan), b(f(_)), b(h(_)), p(susan), p(f(_)), p(h(_)) \}
\]

\[
\Phi_0^{P, d_3}(\theta) = \emptyset
\]

\[
\Phi_1^{P, d_3}(\theta) = \{ a(f(_)), b(f(_)) \}
\]

\[
\Phi_2^{P, d_3}(\theta) = \{ a(f(_)), b(f(_)), p(f(_)) \}
\]

\[
\Phi_3^{P, d_3}(\theta) = \{ a(f(_)), b(f(_)), p(f(_)) \}
\]

\[
\text{If } (\Phi_3^{P, d_3} = \Phi_3^{P, d_3}(\theta) = \{ a(f(_)), b(f(_)), p(f(_)) \})
\]

while \( d_3(MP) = d_3(\{a(f(_)), b(f(_))\}) \text{ and } d_3(MP) \neq d_3(MP) \text{ for the program.}

\[
\text{If } (\Phi_3^{P, d_3} = \{ a(f(_)), b(f(_)), p(f(_)) \})
\]

After eliminating false candidates, we have

\[
\text{If } (\Phi_3^{P, d_3} = \{ a(f(susan)), b(f(h(_))) \})
\]

which is the same as \( d_3(MP) \).

The \( p(f(susan)) \) has been eliminated by the following process. First, \( p(f(susan)) \) is resolved with the clause \( p(x) = -a(x), b(x) \text{ resulting in } a(f(susan)), b(f(h(susan))). \) Then subgoal \( a(f(susan)) \) is resolved with the unit clause \( a(f(susan)) \). However, \( b(f(h(susan))) \) cannot be resolved with \( b(f(h(susan))) \) because \( d_3(b(f(susan))) = d_3(b(f(h(susan)))) \text{ 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 abstract interpretation 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. [6] 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**


