Backward Pair Sharing Analysis

Lunjin Lu[1] and Andy King[2]

[1] Oakland University, MI 48309, USA.
[2] University of Kent, CT2 7NF, UK.

Abstract. This paper presents a backward sharing analysis for logic programs. The analysis computes pre-conditions for a query that guarantee a given post-condition is satisfied after the query is successfully executed. The analysis uses a pair sharing domain and is capable of inferring pre-conditions that ensure the absence of sharing. This, in turn, has many applications in logic programming. The work is unique in that it demonstrates that backward analysis is applicable even for properties that are not closed under instantiation.
Keywords: Abstract interpretation; Backward analysis; Pair-Sharing

1 Introduction

Sharing analysis is useful in specialising, optimising, compiling and parallelising logic programs and thus sharing analysis is an important topic of both abstract interpretation and logic programming. Sharing domains track possible sharing between program variables since optimisations and transformations can typically only be applied in the absence of sharing. Conventionally, sharing is traced in the direction of the control-flow in a query-directed fashion from an initial state. This paper considers the dual problem: the problem of inferring a set of initial states for which an optimisation or transformation is applicable. Specifically, the paper presents a novel backward sharing analysis that propagates information against the control-flow to infer pre-conditions on the variable sharing of a query. The pre-conditions are inferred from a given post-condition which encodes the sharing requirement. The analysis guarantees that if the inferred pre-condition holds for a query, then any successful computation will satisfy the post-condition, thereby ensuring that the optimisation or transformation is applicable.

This paper presents a novel, backward sharing analysis that is realised with abstract interpretation [2]. It is constructed as a suite of abstract operations on the classic pair-sharing domain [1, 14, 24] which captures information about linearity and variable independence. These operations instantiate a backward analysis framework which, in turn, takes care of the algorithmic concerns associated with a fixpoint calculation. This paper focuses on the two key abstract operations: the backward abstract unification and the backward abstract composition operations. The backward abstract unification operation computes a pre-condition for a given equation and its post-condition. The backward abstract composition operation calculates a pre-condition for a call from its post-condition and a description of its answer substitutions. The other abstract operations are much simpler and are more or less straightforward to construct. These operations are omitted from the paper for brevity.

The remainder of the paper is organised as follows. Section 2 introduces basic concepts used throughout the paper. Section 3 contains a brief description of the abstract interpretation framework within which the backward sharing analysis sits. Sections 4-6 introduce the abstract domain and the abstract operations. Section 7 reviews related work and section 8 concludes. Proofs are omitted due to space limitation.

2 Preliminaries

This section recalls some basic concepts in logic programming and abstract interpretation. The reader is referred to [17] and [2] for more detailed exposition.

Let Σ be a set of function symbols, $\mathcal{V}$ a denumerable set of variables. We assume that Σ contains at least one function symbol of arity 0. *Term* denotes the set of terms that can be constructed from Σ and $\mathcal{V}$.

An equation is a formula of the form $t_1 = t_2$ with $t_1, t_2 \in \mathit{Term}$. The set of all equations is denoted as *Eqn* whereas the set of all substitutions is denoted *Sub*. Let $dom(\theta)$ be the domain of a substitution θ, and $\mathbf{V}(o)$ the set of variables in the syntactic object o. Let $\mathit{Sub}_{fail} = \mathit{Sub} \cup \{\mathit{fail}\}$. Given $e \in \mathit{Eqn}$, $mgu : \mathit{Eqn} \mapsto \mathit{Sub}_{fail}$ returns either a most general unifier for e if e is unifiable or *fail* otherwise. For brevity, let $mgu(t_1, t_2) = mgu(t_1 = t_2)$. The function composition operation $\circ$ is defined as $f \circ g = \lambda x.f(g(x))$. Denote the size of a term t by $|t|$ and the number of elements in a set S by $|S|$.

Let $\langle C, \sqsubseteq^C, \sqcup^C, \sqcap^C, \top^C, \bot^C \rangle$ be a complete lattice and $S \subseteq C$. S is a Moore family iff $\top^C \in S$ and $s_1 \sqcap^C s_2 \in S$ for any $s_1, s_2 \in S$. Let $\langle D, \sqsubseteq^D \rangle$ be a poset. A function $\gamma : D \mapsto C$ is a concretization function iff γ is a monotone and $\gamma(D)$ is a Moore family. A concretization function from D to C induces a Galois connection between D and C [2]. The induced adjoint, called an abstraction function, is $\alpha(c) = \sqcap^D \{d \in D \mid c \sqsubseteq^C \gamma(d)\}$.

3 Framework for Backward Analysis

The backward sharing analysis is based on a novel abstract semantics [18]. The abstract semantics is sketched below so that the paper is self-contained. It is a (lower) approximation to a collecting semantics that maps a call $p(\boldsymbol{x})$ and a set Θ of substitutions into a set Ξ of substitutions such that, for any $\xi \in \Xi$, if δ is a computed answer for $\xi(p(\boldsymbol{x}))$ then $\delta \circ \xi \in \Theta$, i.e., $\{\Xi\}p(\boldsymbol{x})\{\Theta\}$ is a valid partial correctness formula. Note that $\Xi = \emptyset$ is a valid solution. In a more precise solution, Ξ contains more substitutions without compromising correctness. The collecting semantics is defined on the concrete domain $\langle \wp(\mathit{Sub}), \subseteq \rangle$. It is defined in terms of a suite of concrete operations. The two most important operators are $\mathit{uf}^{-1} : \mathit{Eqn} \times \wp(\mathit{Sub}) \mapsto \wp(\mathit{Sub})$ and $\circ^{-1} : \wp(\mathit{Sub}) \times \wp(\mathit{Sub}) \mapsto \wp(\mathit{Sub})$ defined

$$\begin{aligned} \mathit{uf}^{-1}(e, \Theta) &= \{\xi \in \mathit{Sub} \mid mgu(\xi(e)) \circ \xi \in \Theta\} \\ \Psi \circ^{-1} \Theta &= \{\omega \in \mathit{Sub} \mid \forall \psi \in \Psi.(\psi \circ \omega \in \Theta)\} \end{aligned}$$

The concrete operation uf^{-1} reverses unification. For a given equation e and a set Θ of success substitutions, it returns the set of those initial substitutions ξ such that the unification of e under ξ results in a success substitution in Θ. The concrete operation $\circ^{-1}$ reverses the composition of one substitution with another. Given a set Θ of success substitutions and a set Ψ of computed answer substitutions, it calculates the set of those initial substitutions ω such that the composition of any $\psi \in \Psi$ with ω obtains a success substitution in Θ.

The abstract semantics is parameterised by an abstract domain $\langle Z, \sqsubseteq \rangle$ but actually operates on the disjunctive completion of $\langle Z, \sqsubseteq \rangle$. Let $S \subseteq Z$ and define $\downarrow(S) = \{z_1 \in Z \mid \exists z_2 \in S.z_1 \sqsubseteq z_2\}$. The set of order-ideals of Z, denoted $\wp^{\downarrow}(Z)$, is defined by $\wp^{\downarrow}(Z) = \{S \subseteq Z \mid S = \downarrow(S)\}$. Note that each order-ideal can be represented by the collection of its maximal elements. This representation of an order-ideal will be used in the sequel. The abstract semantics operates over $\wp^{\downarrow}(Z)$ to express pre-conditions which are disjunctive [18]. The semantics essentially computes a denotation for each call which maps a single post-condition (in Z) to a disjunction of pre-conditions (in $\wp^{\downarrow}(Z)$). The abstract semantics is defined in terms of a suite of abstract operations - one for each concrete operation. Implementing these operations instantiates the abstract semantics to obtain a backward analysis. The two abstract operations that mimic uf^{-1} and $\circ^{-1}$ are $\overline{uf}^{-1} : Eqn \times Z \mapsto \wp^{\downarrow}(Z)$ and $\overline{\circ}^{-1} : Z \times Z \mapsto \wp^{\downarrow}(Z)$. The backward abstract unification operation $\overline{uf}^{-1}$ computes a pre-condition for a given equation and its post-condition. The backward abstract composition operation $\overline{\circ}^{-1}$ calculates a pre-condition for an atom from its post-condition and a description of its answer substitutions. These abstract operations are obtained by inverting the corresponding abstract operations from a forward sharing analysis. Let $\gamma : Z \mapsto \wp(Sub)$ be a concretization function. Define $\gamma^{\cup}(Y) = \bigcup_{y \in Y} \gamma(y)$. These abstract operations are required to satisfy their local safety requirements.

(a) $\gamma^{\cup}(\overline{uf}^{-1}(e, z)) \subseteq uf^{-1}(e, \gamma(z))$ for any $e \in Eqn$ and any $z \in Z$, and
(b) $\gamma^{\cup}(z \overline{\circ}^{-1} z') \subseteq \gamma(z) \circ^{-1} \gamma(z')$ for any $z, z' \in Z$.

These requirements state that each abstract operation faithfully lower approximates its corresponding concrete operation.

The following three sections present the backward sharing analysis. The sharing domain captures information about linearity and dependencies between variables of interest. The abstract operations are obtained by inverting abstract operations from a forward sharing analysis.

4 Abstract Domain

A term t is linear iff it does not contain multiple occurrences of any variable. Let the predicate $linear(t)$ hold iff t is linear. Two terms s and t share a variable iff $\mathbf{V}(s) \cap \mathbf{V}(t) \neq \emptyset$. Two variables x and y share under a substitution θ if $\theta(x)$ and $\theta(y)$ share. The possible sharing and possible non-linearity of variables under a substitution θ are represented as a symmetric relation $\pi \subseteq VI \times VI$ [24] where VI is the set of variables in the program. Let PS be the set of symmetric relations over VI. The abstract domain for sharing and linearity, dubbed pair sharing, is $\langle PS, \subseteq, \emptyset, VI^2, \cap, \cup\rangle$ which is a complete lattice. A Galois connection between $\langle PS, \subseteq\rangle$ and $\langle \wp(Sub), \subseteq\rangle$ is obtained as follows [1].

$$\begin{aligned}
\alpha &: \wp(Sub) \mapsto PS \\
\gamma &: PS \mapsto \wp(Sub) \\
\alpha(\Theta) &= \bigcup_{\theta \in \Theta} \left\{ \langle x, y\rangle \in VI^2 \;\middle|\; \begin{array}{c} (x \neq y \wedge \mathbf{V}(\theta(x)) \cap \mathbf{V}(\theta(y)) \neq \emptyset) \\ \vee \\ (x = y \wedge \neg linear(\theta(x))) \end{array} \right\} \\
\gamma(\pi) &= \bigcup \{\Theta \subseteq Sub \mid \alpha(\Theta) \subseteq \pi\}
\end{aligned}$$

We will write $(u \leftrightarrow v) \in \pi$ to stand for $\{\langle u, v\rangle, \langle v, u\rangle\} \subseteq \pi$. Thus, the set $\{x_1 \leftrightarrow y_1, \cdots, x_n \leftrightarrow y_n\}$ abbreviates $\cup_{i=1}^{n}\{\langle x_i, y_i\rangle, \langle y_i, x_i\rangle\}$. If $(u \leftrightarrow v) \in \pi$ then $(u \leftrightarrow v)$ is called a link in π. We will also use $u \overset{\pi}{\leftrightarrow} v$ to abbreviate $(u \leftrightarrow v) \in \pi$ and $u \overset{\pi}{\Leftrightarrow} v$ to indicate $(u = v \vee u \overset{\pi}{\leftrightarrow} v)$. Define $X \otimes Y = X \times Y \cup Y \times X$ where $X \times Y$ is the Cartesian product of X and Y. $X \otimes Y$ is used to generate a link between each variable of X and each variable of Y. For instance $\{x\} \otimes \{y, z\} = \{x \leftrightarrow y, x \leftrightarrow z\}$. Define $\pi_x = \{y \mid (x \leftrightarrow y) \in \pi\}$. The set π_x includes all the variables that share with x in π. Note that $x \in \pi_x$ if $\langle x, x\rangle \in \pi$. As a further example, $\pi_x = \{y, z\}$ and $\pi_y = \{x, y\}$ where $\pi = \{x \leftrightarrow y, x \leftrightarrow z, y \leftrightarrow y\}$. Define $\phi_X = \bigcup_{x \in X} \phi_x$ where $X \subseteq VI$.

The next stage in the design of the backward sharing analysis is to construct the abstract operations and argue their correctness.

5 Abstract Operation $\overline{uf}^{-1}$

The backward abstract unification operation $\overline{uf}^{-1}$ computes a pre-condition for a given equation and a given post-condition. It is constructed by inverting a forward abstract unification operation given below. The following predicate $\chi : Term \times PS \mapsto \{true, false\}$ will be used in the definition of the forward abstract unification operation: $\chi(t, \pi) = \neg linear(t) \vee ((\mathbf{V}(t))^2 \cap \pi \neq \emptyset)$. The predicate $\chi(t, \pi)$ holds if $\theta(t)$ is non-linear for some $\theta \in \gamma(\pi)$ [1]. We abbreviate $\chi(t, \pi) = true$ as $\chi(t, \pi)$. The forward abstract unification operation is derived from an operation given in [14].

$$\overline{uf}(s = t, \pi) = \begin{cases} \pi \setminus (\mathbf{V}(s) \otimes VI) & \text{if } \mathbf{V}(t) = \emptyset \\ \pi \setminus (\mathbf{V}(t) \otimes VI) & \text{if } \mathbf{V}(s) = \emptyset \\ \pi \cup link(s, t, \pi) \cup (\chi(t, \pi) \rhd link(s, s, \pi)) \cup (\chi(s, \pi) \rhd link(t, t, \pi)) & \text{otherwise} \end{cases}$$

where $link(s, t, \pi) = \{u \leftrightarrow v \mid x \in \mathbf{V}(s) \wedge x \overset{\pi}{\Leftrightarrow} u \wedge v \overset{\pi}{\Leftrightarrow} y \wedge y \in \mathbf{V}(t)\}$ and $\rhd$ is defined $B \rhd \pi = (\text{if } B \text{ then } \pi \text{ else } \emptyset)$. The forward abstract unification operation safely upper approximates the forward concrete unification operation uf [14] where

$$uf(e, \Theta) = \{mgu(\theta(e)) \circ \theta \mid \theta \in \Theta\}$$

The following lemma justifies the construction of the backward abstract unification operation by inverting the forward abstract unification operation.

Lemma 1. *If* $\overline{uf}(s=t,\pi) \subseteq \psi$ *then* $\gamma(\pi) \subseteq uf^{-1}(s=t,\gamma(\psi))$. ∎

According to lemma 1, a pre-condition for an equation and a post-condition can be obtained as follows. The forward abstract unification operator is run on the equation $s=t$ and each $\pi \in PS$. The pre-condition contains those π such that $\overline{uf}(s=t,\pi) \subseteq \psi$ which are also maximal. Therefore, the following is a correct specification for the backward abstract unification operation.

$$\overline{uf}^{-1}(s=t,\psi) = \{\pi \mid \overline{uf}(s=t,\pi) \subseteq \psi \land \forall \pi' \in PS.(\overline{uf}(s=t,\pi') \subseteq \psi \Rightarrow \pi \not\subset \pi')\}$$

Computing $\overline{uf}^{-1}(s=t,\psi)$ via membership checking is however not feasible. Suppose that VI contains n variables. The abstract domain then has $2^{\frac{n(n+1)}{2}}$ elements. Running $\overline{uf}$ on all these elements is practically impossible even for a relatively small n, say 7. The remainder of this subsection gives a polynomial method for computing $\overline{uf}^{-1}(s=t,\psi)$ starting with simple cases. Without loss of generality, we assume that s and t unify for otherwise, $\{VI^2\}$ is a valid pre-condition.

Case $\mathbf{V}(t)=\emptyset$. The effect of the forward abstract unification of $s=t$ in a pair sharing π is to remove those links that are incident to variables in $\mathbf{V}(s)$. Let ψ be the result of this pruning process – the post-condition. Then the unique maximal pre-condition is given by $\psi \cup (\mathbf{V}(s) \otimes VI)$. So, $\overline{uf}^{-1}(s=t,\psi) = \{\psi \cup (\mathbf{V}(s) \otimes VI)\}$.

Example 1. Let $\psi = \{w \leftrightarrow x, x \leftrightarrow x\}$ and $VI = \{w,x,y,z\}$. Then $\overline{uf}^{-1}(f(x,y) = f(a,b),\psi) = \{\{w \leftrightarrow x, w \leftrightarrow y, x \leftrightarrow x, x \leftrightarrow y, x \leftrightarrow z, y \leftrightarrow y, y \leftrightarrow z\}\}$. ∎

Case $\mathbf{V}(s)=\emptyset$. By symmetry to the above, $\overline{uf}^{-1}(s=t,\psi) = \{\psi \cup (\mathbf{V}(t) \otimes VI)\}$. When $\mathbf{V}(t)=\emptyset$ and $\mathbf{V}(s)=\emptyset$, both cases apply and $\overline{uf}^{-1}(s=t,\psi)=\{\psi\}$.

Case $\mathbf{V}(s)\neq\emptyset \land \mathbf{V}(t)\neq\emptyset$. By the definition of $\overline{uf}$, we have $\pi \subseteq \overline{uf}(s=t,\pi)$ for any π. Thus if $\overline{uf}(s=t,\pi) \subseteq \psi$ then $\pi \subseteq \psi$, hence π can be obtained by removing a symmetric subset τ of ψ. The problem is how to find τ. The forward abstract unification operation $\overline{uf}$ produces a link $u \leftrightarrow v$ from $s=t$ and a subset of the links in ψ. Such a subset of links justifies the presence of $u \leftrightarrow v$ in $\overline{uf}(s=t,\psi)$; and is henceforth called a support set for $u \leftrightarrow v$.

Example 2. Let $\psi = \{w \leftrightarrow x, x \leftrightarrow y, x \leftrightarrow z, y \leftrightarrow z\}$. We have $\chi(x,\psi) = \mathit{false}$ and $\chi(f(y,z),\psi) = \mathit{true}$. So, $\overline{uf}(x = f(y,z),\psi) = \psi \cup \mathit{link}(x, f(y,z),\psi) \cup (\mathit{true} \rhd \mathit{link}(x,x,\psi)) \cup (\mathit{false} \rhd \mathit{link}(f(y,z), f(y,z),\psi)) = \psi \cup \mathit{link}(x, f(y,z),\psi) \cup \mathit{link}(x,x,\psi)$. That $(w \leftrightarrow y) \in \mathit{link}(x, f(y,z),\psi)$ has two justifications: one is that $(w \leftrightarrow x) \in \psi$, $x \in \mathbf{V}(s)$ and $y \in \mathbf{V}(f(y,z))$; the other is that $(w \leftrightarrow x) \in \psi$, $x \in \mathbf{V}(x)$, $z \in \mathbf{V}(f(y,z))$ and $(y \leftrightarrow z) \in \psi$. The link $(w \leftrightarrow y)$ occurs in $\mathit{link}(x,x,\psi)$ because $(w \leftrightarrow x) \in \psi$, $x \in \mathbf{V}(x)$, $x \in \mathbf{V}(x)$ and $(x \leftrightarrow y) \in \psi$. Thus, there are three support sets for $w \leftrightarrow y$: $S_1 = \{w \leftrightarrow x\}$ and $S_2 = \{w \leftrightarrow x, y \leftrightarrow z\}$ and $S_3 = \{w \leftrightarrow x, x \leftrightarrow y\}$. ∎

In order to ensure that forward abstract unification cannot produce a link $u \leftrightarrow v$ that is not in ψ, all the support sets for $u \leftrightarrow v$ must be destroyed. A support set is destroyed if just one of its links is removed. Therefore, to prevent $u \leftrightarrow v$ from being produced, it is necessary to remove a set of links that contains one link from each of its support sets. Such a set is called a frontier for $u \leftrightarrow v$.

Definition 1. *Let $h : PS \mapsto PS$ be monotonic and $\psi \in PS$ and $\tau \subseteq \psi$. If $(u \leftrightarrow v) \in h(\psi)$ and $(u \leftrightarrow v) \notin h(\psi \setminus \tau)$, we say that (removal of) τ (from ψ) excludes $u \leftrightarrow v$ from $h(\psi)$ and that τ is a frontier for $u \leftrightarrow v$ in $h(\psi)$. Let $\phi \in PS$. We say that τ is a frontier for ϕ in $h(\psi)$ if, for each link $u \leftrightarrow v \in \phi$, τ is a frontier for $u \leftrightarrow v$ in $h(\psi)$.* ∎

The particular notions of frontier and exclusion that are required to define backward abstract unification are obtained by putting $h = \lambda\phi.\overline{uf}(s = t, \phi)$. Another instance of these concepts appears in section 6.

Example 3. There are four frontiers for the $w \leftrightarrow y$ link of example 2. They are $F_1 = \{w \leftrightarrow x\}$, $F_2 = \{w \leftrightarrow x, x \leftrightarrow y\}$, $F_3 = \{w \leftrightarrow x, y \leftrightarrow z\}$ and $F_4 = \{w \leftrightarrow x, x \leftrightarrow y, y \leftrightarrow z\}$. Removing any F_i for $1 \leq i \leq 4$ from ψ will prevent $w \leftrightarrow y$ from being produced. ∎

The above example demonstrates that one frontier for a link may be contained in another. Removing one frontier from ψ results in a pair sharing that is a superset of that obtained by removing another frontier from ψ. Since the precondition that is the object of the computation contains maximal pair sharings, only minimal frontiers for the link should be removed. The following example shows that a link may have more than one minimal frontiers.

Example 4. Let $\psi = \{w \leftrightarrow x, y \leftrightarrow z\}$. Then $(w \leftrightarrow z) \in \overline{uf}(x = g(y), \psi)$. The link has one support set: $\{w \leftrightarrow x, y \leftrightarrow z\}$. Two minimal frontiers for $w \leftrightarrow z$ are $\{w \leftrightarrow x\}$ and $\{y \leftrightarrow z\}$ which are incomparable. ∎

Some links have no frontiers at all. For example, let $\psi = \emptyset$. Then $\overline{uf}(x, y, \psi) = \{x \leftrightarrow y\}$. This indicates that the post-condition ψ is unsatisfiable.

Definition 2. *Let $h : PS \mapsto PS$ be monotonic, $\psi \in PS$ and $\Pi \subseteq PS$. Π is a complete set of frontiers for a link $u \leftrightarrow v$ (a set ϕ of links respectively) in $h(\psi)$ if*

(i) every $\pi \in \Pi$ is a frontier for $u \leftrightarrow v$ (ϕ respectively) in $h(\psi)$; and
(ii) every minimal frontier for $u \leftrightarrow v$ (ϕ respectively) in $h(\psi)$ is in Π. ∎

Observe that a complete set of frontiers may contain a non-minimal frontier.

5.1 Minimal Frontier Function

By the definition of $\overline{uf}$, a link $(u \leftrightarrow v) \notin \psi$ occurs in $\overline{uf}(s = t, \psi)$ iff it occurs in $link(s, t, \pi)$, or $\chi(t, \pi) \rhd link(s, s, \pi)$ or $\chi(s, \pi) \rhd link(t, t, \pi)$. Thus, it is excluded from $\overline{uf}(s = t, \psi)$ iff it is excluded from $link(s, t, \pi)$, and $\chi(t, \pi) \rhd link(s, s, \pi)$ and $\chi(s, \pi) \rhd link(t, t, \pi)$.

We first consider how to exclude a link $u \leftrightarrow v$ from $link(s, t, \psi)$. We rewrite the definition of $link(s, t, \psi)$ into $link(s, t, \psi) = \bigcup_{i=1}^{4} \sigma_i(s, t, \psi)$ where

$$\begin{aligned}
\sigma_1(s, t, \psi) &= \{u \leftrightarrow v \mid u \in \mathbf{V}(s) \wedge v \in \mathbf{V}(t)\}\\
\sigma_2(s, t, \psi) &= \{u \leftrightarrow v \mid u \in \mathbf{V}(s) \wedge v \notin \mathbf{V}(t) \wedge (\psi_v \cap \mathbf{V}(t)) \neq \emptyset\}\\
\sigma_3(s, t, \psi) &= \{u \leftrightarrow v \mid u \notin \mathbf{V}(s) \wedge v \in \mathbf{V}(t) \wedge (\psi_u \cap \mathbf{V}(s)) \neq \emptyset\}\\
\sigma_4(s, t, \psi) &= \{u \leftrightarrow v \mid u \notin \mathbf{V}(s) \wedge v \notin \mathbf{V}(t) \wedge (\psi_u \cap \mathbf{V}(s)) \neq \emptyset \wedge (\psi_v \cap \mathbf{V}(t)) \neq \emptyset\}
\end{aligned}$$

Observe that $(u \leftrightarrow v) \in link(s, t, \psi)$ iff $(u \leftrightarrow v) \in \sigma_i(s, t, \psi)$ for some $1 \leq i \leq 4$. Note that $(u \leftrightarrow v) \in \sigma_i(s, t, \psi)$ implies $(u \leftrightarrow v) \notin \sigma_j(s, t, \psi)$ for $j \neq i$. The following computes the set of minimal frontiers for $u \leftrightarrow v$ in $link(s, t, \psi)$.

$$mf_{link}(u, v, s, t, \psi) = \begin{cases} \emptyset & \text{if } u \in \mathbf{V}(s) \wedge v \in \mathbf{V}(t)\\ \{(\{v\} \otimes \mathbf{V}(t)) \cap \psi\} & \text{if } u \in \mathbf{V}(s) \wedge v \notin \mathbf{V}(t)\\ \{(\{u\} \otimes \mathbf{V}(s)) \cap \psi\} & \text{if } u \notin \mathbf{V}(s) \wedge v \in \mathbf{V}(t)\\ \{(\{u\} \otimes \mathbf{V}(s)) \cap \psi, (\{v\} \otimes \mathbf{V}(t)) \cap \psi\} & \text{if } u \notin \mathbf{V}(s) \wedge v \notin \mathbf{V}(t) \end{cases}$$

Each element in $mf_{link}(u, v, s, t, \psi)$ excludes $u \leftrightarrow v$ from $link(s, t, \psi)$. The empty set in the first branch indicates that the presence of $u \leftrightarrow v$ in $\sigma_1(s, t, \psi)$ is independent of ψ and hence cannot be excluded. The second contains one frontier that consists of links between v and variables in $\mathbf{V}(t)$. The third is dual to the second. The fourth returns a set of two minimal frontiers. One consists of links between v and variables in $\mathbf{V}(t)$; and the other consists of links between u and variables in $\mathbf{V}(s)$.

Lemma 2. *$mf_{link}(u, v, s, t, \psi)$ is a complete set of frontiers for $u \leftrightarrow v$ in $link(s, t, \psi)$.* ∎

We now consider how to exclude $u \leftrightarrow v$ from $(\chi(t, \psi) \rhd link(s, s, \psi))$. By the definition of $\rhd$, $\chi(t, \psi) \rhd link(s, s, \psi) = (\text{if } \chi(t, \psi) \text{ then } link(s, s, \psi) \text{ else } \emptyset)$. Thus, we can either make the condition $\chi(t, \psi)$ false or exclude $u \leftrightarrow v$ from $link(s, s, \psi)$. The latter can be accomplished by removing from ψ any element in $mf_{link}(u, v, s, s, \psi)$. Note that $\chi(t, \psi) = \neg linear(t) \vee ((\mathbf{V}(t))^2 \cap \psi \neq \emptyset)$. If $\neg linear(t)$ holds then $\chi(t, \psi)$ cannot be falsified by removing any part of ψ. In this case, we can exclude $u \leftrightarrow v$ from $\chi(t, \psi) \rhd link(s, s, \psi)$ only by removing an element in $mf_{link}(u, v, s, s, \psi)$. Otherwise, $linear(t)$ holds. We can alternatively choose to falsify $((\mathbf{V}(t))^2 \cap \psi \neq \emptyset)$. This can be done by removing all the links in $(\mathbf{V}(t))^2 \cap \psi$. Thus, each element in $(linear(t) \wedge ((\mathbf{V}(t))^2 \cap \psi \neq \emptyset) \rhd \{(\mathbf{V}(t))^2 \cap \psi\}) \cup mf_{link}(u, v, s, s, \psi)$ excludes $u \leftrightarrow v$ from $\chi(t, \psi) \rhd link(s, s, \psi)$. Excluding $u \leftrightarrow v$ from $(\chi(s, \psi) \rhd link(t, t, \psi))$ is symmetric.

Lemma 3. *$(linear(t) \wedge ((\mathbf{V}(t))^2 \cap \psi \neq \emptyset) \rhd \{(\mathbf{V}(t))^2 \cap \psi\}) \cup mf_{link}(u, v, s, s, \psi)$ is a complete set of frontiers for $u \leftrightarrow v$ in $\chi(t, \psi) \rhd link(s, s, \psi)$.* ∎

In order to exclude a link $u \leftrightarrow v$ from $h_1(\psi) \cup h_2(\psi)$, it is necessary to exclude $u \leftrightarrow v$ from both $h_1(\psi)$ and $h_2(\psi)$. This can be accomplished by removing from ψ a frontier for $u \leftrightarrow v$ in $h_1(\psi)$ and a frontier for $u \leftrightarrow v$ in $h_2(\psi)$. The union of a frontier for $u \leftrightarrow v$ in $h_1(\psi)$ and a frontier for $u \leftrightarrow v$ in $h_2(\psi)$ is a frontier for $u \leftrightarrow v$ in $h_1(\psi) \cup h_2(\psi)$. To this end, define $\Psi \uplus \Phi = min(\{\psi \cup \phi \mid \psi \in \Psi \wedge \phi \in \Phi\})$ where $min(\Pi)$ returns the set of the elements in Π that are minimal with respect to $\subseteq$. The operation $\uplus$ is commutative and associative.

Lemma 4. *Let $\psi \in PS$, $h_1, h_2 : PS \mapsto PS$ monotonic functions, and Φ_i a complete set of frontiers for a link $u \leftrightarrow v$ in $h_i(\psi)$ for $1 \leq i \leq 2$. Then $\Phi_1 \uplus \Phi_2$ is a complete set of frontiers for $u \leftrightarrow v$ in $h_1(\psi) \cup h_2(\psi)$.* ∎

The function $mf(u, v, s, t, \psi)$ below returns the set of all minimal frontiers for $u \leftrightarrow v$ in $\overline{uf}(s = t, \psi)$.

$$
\begin{aligned}
& mf(u, v, s, t, \psi) = \\
& \quad mf_{link}(u, v, s, t, \psi) \\
& \uplus ((linear(t) \wedge ((\mathbf{V}(t))^2 \cap \psi \neq \emptyset) \rhd \{(\mathbf{V}(t))^2 \cap \psi\}) \ \cup \ mf_{link}(u, v, s, s, \psi)) \\
& \uplus ((linear(s) \wedge ((\mathbf{V}(s))^2 \cap \psi \neq \emptyset) \rhd \{(\mathbf{V}(t))^2 \cap \psi\}) \ \cup \ mf_{link}(u, v, t, t, \psi))
\end{aligned}
$$

Note that non-minimal frontiers for a link are removed by the *min* operation employed in the $\uplus$ operation and that minimal frontiers for a link are computed without computing support sets for the link.

Lemma 5. *$mf(u, v, s, t, \psi)$ is a complete set of frontiers for $u \leftrightarrow v$ in $\overline{uf}(s = t, \psi)$.* ∎

Example 5. Let $\psi = \{w \leftrightarrow x, y \leftrightarrow z\}$. Then $(w \leftrightarrow z) \in \overline{uf}(x = g(y), \psi)$. The set $mf(w, z, x, g(y), \psi)$ of minimal frontiers for $w \leftrightarrow z$ in $\overline{uf}(x = g(y), \psi)$ is computed as follows. We calculate $linear(g(y)) = true$ and $\mathbf{V}(g(y))^2 \cap \psi = \{y \leftrightarrow y\} \cap \psi = \emptyset$. Thus, $(linear(g(y)) \wedge (\mathbf{V}(g(y))^2 \cap \psi \neq \emptyset)) = false$ and hence $(linear(g(y)) \wedge (\mathbf{V}(g(y))^2 \cap \psi \neq \emptyset)) \rhd \{\mathbf{V}(g(y))^2 \cap \psi\} = \emptyset$. We can also obtain $(linear(x) \wedge (\mathbf{V}(x)^2 \cap \psi \neq \emptyset)) \rhd \{\mathbf{V}(x)^2 \cap \psi\} = \emptyset$. Thus, $mf(w, z, x, g(y), \psi) = mf_{link}(w, z, x, g(y), \psi) \uplus (\emptyset \cup mf_{link}(w, z, x, x, \psi)) \uplus (\emptyset \cup mf_{link}(w, z, g(y), g(y), \psi))$. We first calculate $mf_{link}(w, z, x, g(y), \psi) = \{(\{w\} \otimes \{x\}) \cap \psi, (\{z\} \otimes \{y\}) \cap \psi\} = \{\{w \leftrightarrow x\}, \{y \leftrightarrow z\}\}$ since $w \notin \{x\}$ and $z \notin \{y\}$. We can also obtain $mf_{link}(w, z, x, x, \psi) = \{\{w \leftrightarrow x\}\}$ and $mf_{link}(w, z, g(y), g(y), \psi) = \{\{y \leftrightarrow z\}\}$. So, $mf(w, z, x, g(y), \psi) = \{\{w \leftrightarrow x\}, \{y \leftrightarrow z\}\} \uplus \{\{w \leftrightarrow x\}\} \uplus \{\{y \leftrightarrow z\}\} = \{\{w \leftrightarrow x\}, \{y \leftrightarrow z\}\}$. ∎

Suppose that two links $u \leftrightarrow v$ and $u' \leftrightarrow v'$ need be excluded from $\overline{uf}(s = t, \psi)$. Removing from ψ a frontier for one link will exclude the link. Both links will be excluded if the union of a frontier for $u \leftrightarrow v$ and a frontier for $u' \leftrightarrow v'$ is removed from ψ. A frontier for a set of n links is the union of n frontiers - one for each link.

Lemma 6. *Let $\psi \in PS$, $h : PS \mapsto PS$ a monotonic function, $\phi_i \in PS$ and $\Pi_i \subseteq PS$ for $1 \leq i \leq 2$. If Π_i is a complete set of frontiers for ϕ_i in $h(\psi)$ then $\Pi_1 \uplus \Pi_2$ is a complete set of frontiers for $\phi_1 \cup \phi_2$ in $h(\psi)$.* ∎

The following lemma provides a constructive method for computing $\overline{uf}^{-1}(s = t, \psi)$ for the case $\mathbf{V}(s) \neq \emptyset \wedge \mathbf{V}(t) \neq \emptyset$. The forward abstract unification operation $\overline{uf}$ is first employed to compute $\psi' = \overline{uf}(s = t, \psi)$. The set of minimal frontiers for $\psi' \setminus \psi$ is then computed. It is $\uplus_{(u \leftrightarrow v) \in (\psi' \setminus \psi)} mf(u, v, s, t, \psi)$. Each pair sharing in $\overline{uf}^{-1}(s = t, \psi)$ is obtained by removing one of these minimal frontiers from ψ.

Lemma 7. $\overline{uf}^{-1}(s = t, \psi) = \{\psi \setminus \tau \mid \tau \in \uplus_{(u \leftrightarrow v) \in (\psi' \setminus \psi)} mf(u, v, s, t, \psi)\}$ *where* $\psi' = \overline{uf}(s = t, \psi)$. ∎

Lemmas 1 and 7 imply the correctness of $\overline{uf}^{-1}$. We now show that $\overline{uf}^{-1}(s = t, \psi)$ is polynomial in $|s| + |t| + |VI|$. Operations $\cup$, $\cap$ and $\otimes$ are polynomial in $|VI|$. Let Π be a set of minimal frontiers and $\pi_1, \pi_2 \in \Pi$ such that $\pi_2 \neq \pi_1$. Then π_1 contains at least one link that does not belong to π_2. Thus, $|\Pi| \leq |VI|^2$ because $|\pi_1| \leq |VI|^2$. So, $\uplus$ is polynomial in $|VI|$. The forward abstract unification $\psi' = \overline{uf}(s = t, \psi)$ is polynomial in $|s| + |t| + |VI|$ [1]. All links $u \leftrightarrow v$ in $\psi' \setminus \psi$ invoke $mf(u, v, s, t, \psi)$ with the same s, t and ψ. Thus, $\mathbf{V}(s)$, $\mathbf{V}(t)$, $linear(s)$ and $linear(t)$ can be computed with their results being memoised for use in computing $mf(u, v, s, t, \psi)$ for different links $u \leftrightarrow v$. This takes $O(|s| + |t|)$ time. Using the memoised results, $mf_{link}(u, v, s, t, \psi)$ is polynomial in $|VI|$; so is $mf(u, v, s, t, \psi)$. The computation $\uplus_{(u \leftrightarrow v) \in (\psi' \setminus \psi)} mf(u, v, s, t, \psi)$ is polynomial in $|VI|$ since it invokes mf and $\uplus$ for $|\psi' \setminus \psi| \leq |VI|^2$ times and both mf and $\uplus$ are polynomial in $|VI|$. So, $\overline{uf}^{-1}(s = t, \psi)$ is polynomial in $|s| + |t| + |VI|$.

6 Abstract Operation $\overline{\circ}^{-1}$

The operator $\overline{\circ} : PS \times PS \mapsto PS$ was originally proposed in [1] for composing an abstract initial substitution for an atom with an abstract answer substitution (for the atom and the abstract initial substitution) to obtain an abstract success substitution. It will inverted to obtain $\overline{\circ}^{-1}$ and it is defined

$$\pi \overline{\circ} \phi = \{\langle u, v\rangle \mid u \overset{\phi}{\leftrightarrow} v \vee \exists x, y.(u \overset{\phi}{\Leftrightarrow} x \wedge x \overset{\pi}{\leftrightarrow} y \wedge y \overset{\phi}{\Leftrightarrow} v)\}$$

Note that $\overline{\circ}$ is not commutative. The following result is Lemma 4.4 in [1].

Proposition 1 (Codish, Dams and Yardeni). *Let $\sigma \in \gamma(\pi)$ and $\theta \in \gamma(\phi)$. Then $\sigma \circ \theta \in \gamma(\pi \overline{\circ} \phi)$.*

The following lemma justifies the construction tactic of inverting $\overline{\circ}$ to obtain $\overline{\circ}^{-1} : PS \times PS \mapsto PS$.

Lemma 8. *If $\pi \overline{\circ} \phi \subseteq \psi$ then $\gamma(\phi) \subseteq (\gamma(\pi) \circ^{-1} \gamma(\psi))$.* ∎

The above lemma implies that the following is a correct specification for $\overline{\circ}^{-1}$.

$$\pi \overline{\circ}^{-1} \psi = \{\phi \mid (\pi \overline{\circ} \phi \subseteq \psi) \wedge \forall \phi' \in PS.((\pi \overline{\circ} \phi' \subseteq \psi) \Rightarrow (\phi' \not\supset \phi))$$

Again, we need to find a practical method for computing $\pi \overline{\circ}^{-1} \psi$. For any $\pi, \phi \in PS$, we have $\phi \subseteq \pi \overline{\circ} \phi$. Thus, if $\pi \overline{\circ} \phi \subseteq \psi$ then $\phi \subseteq \psi$. A pair sharing in $\pi \overline{\circ}^{-1} \psi$ can be obtained by removing a set τ of links from ψ. The problem is thus again equivalent to finding τ. The notion of a support set and that of a frontier carry over with $\lambda \phi.(\pi \overline{\circ} \phi)$ taking the place of $\lambda \phi.\overline{uf}(s = t, \phi)$.

Example 6. Let $\psi = \{w \leftrightarrow x, x \leftrightarrow y, y \leftrightarrow z\}$ and $\pi = \{x \leftrightarrow y\}$. Then $\pi \overline{\circ} \psi = \pi \cup \psi \cup \{w \leftrightarrow y, w \leftrightarrow z, x \leftrightarrow x, x \leftrightarrow z, y \leftrightarrow y\}$. There is one support set for $w \leftrightarrow z$: $\{w \leftrightarrow x, y \leftrightarrow z\}$. Two minimal frontiers are obtained from the support set: $\{w \leftrightarrow x\}$ and $\{y \leftrightarrow z\}$. Removing either of them excludes $w \leftrightarrow z$ from $\pi \overline{\circ} \psi$. ∎

By expanding the definition of $\overline{\circ}$, we have $\pi \overline{\circ} \psi = \pi \cup \psi \cup (\psi \bowtie \pi) \cup (\pi \bowtie \psi) \cup (\psi \bowtie \pi \bowtie \psi)$ where $\pi_1 \bowtie \pi_2 = \{u \leftrightarrow v \mid \exists w.(u \overset{\pi_1}{\leftrightarrow} w \wedge w \overset{\pi_2}{\leftrightarrow} v)\}$ is associative. The following function computes the set of minimal frontiers for $u \leftrightarrow v$ in $\pi \overline{\circ} \psi$.

$$\begin{aligned} mftc(u, v, \pi, \psi) = \{(\{u\} \otimes \pi_v) \cap \psi\} &\uplus \{(\pi_u \otimes \{v\}) \cap \psi\} \\ &\uplus \{(\{u\} \otimes \pi_{\psi_v}) \cap \psi, (\{v\} \otimes \pi_{\psi_u}) \cap \psi\} \end{aligned}$$

Some explanation is in order. Assume that $(u \leftrightarrow v) \notin \psi$ and $(u \leftrightarrow v) \notin \pi$. Consider how to exclude $u \leftrightarrow v$ from $\pi \overline{\circ} \psi$. The link $u \leftrightarrow v$ belongs to $\psi \bowtie \pi$ if u is linked via ψ to any variable that is linked to v via π. Thus, it is necessary to remove all links in $(\{u\} \otimes \pi_v) \cap \psi$. Excluding $u \leftrightarrow v$ from $\pi \bowtie \psi$ is symmetric. Observe that π_{ψ_v} consists of those variables that are linked to v via a link in π followed by a link in ψ. In order to exclude $u \leftrightarrow v$ from $\psi \bowtie \pi \bowtie \psi$, we either remove the set of all links from u to variables in π_{ψ_v} or remove the set of all links from v to variables in π_{ψ_u}. The former is $(\{u\} \otimes \pi_{\psi_v}) \cap \psi$ and the latter is $(\{v\} \otimes \pi_{\psi_u}) \cap \psi$. Finally, the link $u \leftrightarrow v$ is excluded from $\pi \overline{\circ} \psi$ if it is excluded from $\psi \bowtie \pi$, $\pi \bowtie \psi$ and $\psi \bowtie \pi \bowtie \psi$.

Lemma 9. *For any $\pi \in PS$, $mftc(u, v, \pi, \psi)$ is a complete set of frontiers for $u \leftrightarrow v$ in $(\psi \bowtie \pi) \cup (\pi \bowtie \psi) \cup (\psi \bowtie \pi \bowtie \psi)$.* ∎

The following lemma provides a polynomial method for computing $\pi \overline{\circ}^{-1} \psi$. Together with lemma 8, it ensures the correctness of the abstract operation $\overline{\circ}^{-1}$. If $\pi \not\subseteq \psi$ then $\pi \overline{\circ} \phi \not\subseteq \psi$ for any $\phi \in PS$. In this case, the post-condition ψ is unsatisfiable and hence the pre-condition is the empty set of pair sharings. Otherwise, the pre-condition consists of those pair sharings that are obtained by removing minimal frontiers for $(\pi \overline{\circ} \psi) \setminus \psi$.

Lemma 10. *For any $\pi, \psi \in PS$,*

$$\pi \overline{\circ}^{-1} \psi = \begin{cases} \emptyset & \text{if } \pi \not\subseteq \psi \\ \{\psi \setminus \tau \mid \tau \in \biguplus_{(u \leftrightarrow v) \in (\pi \overline{\circ} \psi) \setminus \psi} mftc(u, v, \pi, \psi)\} & \text{otherwise} \end{cases}$$ ∎

Example 7. Continuing with example 6, $(\pi \overline{\circ} \psi) \setminus \psi = \{w \leftrightarrow y, w \leftrightarrow z, x \leftrightarrow x, x \leftrightarrow z, y \leftrightarrow y\}$. We have $\pi_w = \pi_z = \emptyset$, $\pi_{\psi_z} = \pi_y = \{x\}$ and $\pi_{\psi_w} = \pi_x = \{y\}$. Thus, $mftc(w, z, \pi, \psi) = \{\emptyset\} \uplus \{\emptyset\} \uplus \{(\{w\} \otimes \pi_{\psi_z}) \cap \psi, (\{z\} \otimes \pi_{\psi_w}) \cap \psi\} = \{\{w \leftrightarrow x\}, \{y \leftrightarrow z\}\}$. Omitting details, we obtain other sets of minimal frontiers: $mftc(w, y, \pi, \psi) = \{\{w \leftrightarrow x\}\}$, $mftc(x, z, \pi, \psi) = \{\{y \leftrightarrow z\}\}$ and $mftc(x, x, \pi, \psi) = mftc(y, y, \pi, \psi) = \{\{x \leftrightarrow y\}\}$. The set of minimal frontiers for $(\pi \overline{\circ} \psi) \setminus \psi$ is $\uplus_{(u \leftrightarrow v) \in (\pi \overline{\circ} \psi) \setminus \psi} mftc(u, v, \pi, \psi) = mftc(w, y, \pi, \psi) \uplus mftc(w, z, \pi, \psi) \uplus mftc(x, x, \pi, \psi) \uplus mftc(x, z, \pi, \psi) \uplus mftc(y, y, \pi, \psi) = \{\{w \leftrightarrow x, x \leftrightarrow y, y \leftrightarrow z\}\} = \{\psi\}$. Thus, $\pi \overline{\circ}^{-1} \psi = \{\emptyset\}$. ∎

We now turn to the time complexity of $\pi \overline{\circ}^{-1} \psi$. Observe that $\pi \overline{\circ} \psi$ is polynomial in $|\mathit{VI}|$ since $\bowtie$ and $\cup$ are polynomial in $|\mathit{VI}|$. Since $\uplus$, $\otimes$ and $\cap$ are polynomial in $|\mathit{VI}|$, $mftc(u, v, \pi, \psi)$ is polynomial in $|\mathit{VI}|$. Thus, $\uplus_{(u \leftrightarrow v) \in (\pi \overline{\circ} \psi) \setminus \psi} mftc(u, v, \pi, \psi)$ is polynomial in $|\mathit{VI}|$; so is $\pi \overline{\circ}^{-1} \psi$. Both $\overline{uf}^{-1}(s = t, \psi)$ and $\pi \overline{\circ}^{-1} \psi$ are polynomial; in contrast the widely-used set-sharing analysis has a forward abstract unification operator that is exponential [13].

7 Related work

Though backward analysis has been a subject of intense research in functional programming [25, 11, 5], backward analysis has until very recently [9, 15, 19, 22] been rarely applied in logic programming. One notable exception is the demand analysis of [4]. This analysis infers the degree of instantiation necessary to allow the guards of a concurrent constraint program to reduce: it is local analysis that does not consider the possible suspension of body calls. The information it infers is useful in detecting (uni-modal) predicates which can be implemented with relatively straightforward suspension machinery. A more elaborate backward analysis for concurrent constraint programs is [6]. This demand analysis infers how much input is necessary for a procedure to generate a certain amount of output. This information is useful for adding synchronisation constraints to a procedure to delay execution and thereby increase grain size, and yet not introduce deadlock.

Mazur, Janssens and Bruynooghe [21] present a kind of *ad hoc* backward analysis to derive reuse conditions from a goal-independent reuse analysis. The analysis propagates reuse information from a point where a structure is decomposed in a clause to the point where the clause is invoked in its parent clause. This is similar in spirit to how demand is passed from a callee to a caller in the backward analysis described in this paper. However, the reuse analysis does not propagate information right-to-left across a clause, resulting in a less precise analysis. The above backward analyses are not specialisations of any framework for logic program analysis. In [22], a backward analysis is proposed to infer specialisation conditions that decide whether a call to a predicate from a particular call site should invoke a specialised version of the predicate or an unoptimised version. Specifically, if these conditions are satisfied by an (abstract) call in a goal-dependent analysis then the call will possibly lead to valuable optimisations, and therefore it should not be merged with calls that lead to a lower level of optimisation. The specialisation conditions produced by the backward analysis are not sufficient conditions and need to be checked by an ensuing forward analysis. In contrast, the pre-conditions obtained by the backward sharing analysis are guaranteed to be sufficient and thus need not be checked by a forward analysis.

In [15], the authors of the current paper present an abstract semantics for backward analysis of logic programs and specialise it to infer safe modes for queries which ensure that the groundness assertions are not violated using the groundness domain *Pos* [20]. The backward groundness analysis is also used in termination inference by [9]. A backward analysis using the abstract semantics is performed by first computing an upper approximation to the success set of the program and then a lower approximation to the set of programs states (substitutions) that will not violate any assertion. The key operation that propagates information backwards the control-flow is the (intuitionistic) logical implication operation. Thus, the abstract domain of the analysis is required to condense. This, however, is a strong requirement for any domain. The abstract domain for the backward sharing analysis does not condense. The approach advocated in this paper is to found backward analysis on a novel abstract semantics for backward analysis which relaxes this requirement. An analysis using the new abstract semantics is a greatest fixpoint computation whilst an analysis with the abstract semantics in [15] additionally computes a least fixpoint computation.

The authors of this paper have also shown how backward analysis can be used to perform type inference [19]. Given type signatures for a collection of selected predicates such as builtin or library predicates, the analysis of [19] infers type signatures for other predicates such that the execution of any query satisfying the inferred type signatures will not violate the given type signatures. Thus, the backward type analysis generalises type checking in which the programmer manually specifies type signatures for all predicates that are checked for consistency by a type checker. The work of [19] is distinct from that reported in this paper. The property considered in [19] is closed under instantiation whilst that in this paper is not.

Very recently, Gallagher [8] has proposed a program transformation as a tactic for realising backward analysis in terms of forward analysis. Assertions are realised with a meta-predicate $\mathrm{d}(G,P)$. The meta-predicate $\mathrm{d}(G,P)$ expresses the relationship between an initial goal G and a property P to be checked at some program point. The meta-predicate $\mathrm{d}(G,P)$ holds if there is a derivation starting from G leading to the program point with which P is associated. The transformed program defining the predicate d can be seen as a realisation of the resultants semantics [7]. Backward analysis is performed by examining the meaning of d, which can be approximated using a standard forward analysis, to deduce goals G that imply that the property P holds. This work is both promising and intriguing because it finesses the requirement of calculating a greatest fixpoint. One interesting line of enquiry would be to compare the expressive power of transformation – the pre-conditions its infers – against those deduced via a be-spoke backward analysis framework [15, 19].

Giacobazzi [10] proposes a method for an abductive analysis of modular logic programs. From a specification of the success patterns of a predicate defined in one module which calls open predicates defined in other modules, the method derives a specification for the success patterns of the open predicates. In contrast, our method derives a specification for the call patterns of some (unspecified) predicates from a specification of the call patterns of other (specified) predicates.

Pedreschi and Ruggieri [23] develop a calculus of weakest pre-conditions and weakest liberal pre-conditions, the latter of which is essentially a reformulation of Hoare's logic. Weakest liberal pre-conditions are characterised as the greatest fixpoint of a co-continuous function on the space of interpretations. Our work takes these ideas forward to show how abstract interpretation can infer weakest liberal pre-conditions.

Cousot and Cousot [3] explain how a backward collecting semantics can be used to precisely characterise states that arise in finite SLD-derivations. They present both a forward collecting semantics that records the descendant states that arise from a set of initial states and a backward collecting semantics that records those states which occur as ascendant states of the final states. By combining both semantics, they characterise the set of descendant states of the initial states which are also ascendant states of the final states of the transition system. This use of backward analysis is primarily as a device to improve the precision of a goal-dependent analysis. Our work is more radical in the sense that it shows how a bottom-up analysis performed in a backward fashion, can be used to characterise initial queries. Moreover it is used for lower approximation rather than upper approximation.

Hughes and Launchbury [12] shows how and when to reverse an analysis based on abstract interpretation. Their work is concerned with analyses of functional programs. In fact, Hughes and Launchbury argue that ideally the direction of an analysis should be reversed without reference to the concrete semantics. Our work demonstrates that this can be accomplished in the analysis of logic programs.

A systematic comparison of the relative precision of forward and backward abstract interpretation of logic programs is given in [16].

8 Conclusions

A backward sharing analysis for logic programs has been presented. From a given post-condition for an atom, it derives a pre-condition. Any successful execution of the atom in any state satisfying the pre-condition ends in a state satisfying the post-condition. Abstract operations for the backward sharing analysis are constructed by inverting those for a forward sharing analysis. The work demonstrates that backward analysis is applicable for properties that are not closed under instantiation.

Acknowledgements

This work was supported by the National Science Foundation under grants CCR-0131862 and INT-0327760.

References

1. M. Codish, D. Dams, and E. Yardeni. Derivation and safety of an abstract unification algorithm for groundness and aliasing analysis. In K. Furukawa, editor, *Proceedings of the Eighth International Conference on Logic Programming*, pages 79–93. The MIT Press, 1991.
2. P. Cousot and R. Cousot. Abstract interpretation: a unified framework for static analysis of programs by construction or approximation of fixpoints. In *Principles of Programming Languages*, pages 238–252. The ACM Press, 1977.
3. 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.
4. S.K. Debray. QD-Janus: A sequential implementation of Janus in Prolog. *Software Practice and Experience*, 23(12):1337–1360, 1993.
5. P. Dyber. Inverse image analysis generalises strictness analysis. *Information and Computation*, 90(2):194–216, 1991.
6. M. Falaschi, P. Hicks, and W. Winsborough. Demand Transformation Analysis for Concurrent Constraint Programs. *The Journal of Logic Programming*, 41(3):185–215, 2000.
7. M. Gabbrielli, G. Levi, and M. C. Meo. Resultants Semantics for Prolog. *Journal of Logic and Computation*, 6(4):491–521, 1996.
8. J.P. Gallagher. A Program Transformation for Backwards Analysis of Logic Programs. In *Pre-Proceedings of LOPSTR 2003 - International Symposium on Logic-based Program Synthesis and Transformation*, 2003.
9. S. Genaim and M. Codish. Inferring termination conditions for logic programs using backwards analysis. In R. Nieuwenhuis and A. Voronkov, editors, *Proceedings of the Eighth International Conference on Logic for Programming, Artificial Intelligence and Reasoning*, volume 2250 of *Lecture Notes in Artificial Intelligence*, pages 681–690. Springer, 2001.
10. R. Giacobazzi. Abductive analysis of modular logic programs. *Journal of Logic and Computation*, 8(4):457–484, 1998.
11. C. Hall and D. Wise. Generating function versions with rational strictness patterns. *Science of Computer Programming*, 12(1):39–74, 1989.
12. J. Hughes and J. Launchbury. Reversing abstract interpretations. *Science of Computer Programming.*, 22:307–326, 1994.
13. D. Jacobs and A. Langen. Static analysis of logic programs for independent and parallelism. *The Journal of Logic Programming*, 13(1–4):291–314, 1992.
14. A. King. Pair-sharing over rational trees. *The Journal of Logic Programming*, 46(1–2):139–155, 2000.
15. A. King and L. Lu. A backward analysis for constraint logic programs. *Theory and Practice of Logic Programming*, 2(4&5):517–547, 2002.
16. A. King and L. Lu. Forward versus Backward Verification of Logic Programs. In C. Palamidessi, editor, *Proceedings of Nineteenth International Conference on Logic Programming*, volume 2916 of *Lecture Notes in Computer Science*, pages 315–330, 2003.

17. J.W. Lloyd. *Foundations of Logic Programming.* Springer-Verlag, 1987.
18. L. Lu and A. King. A Framework for Backward Analysis of Logic Programs. In Preparation.
19. L. Lu and A. King. Type inference generalises type checking. In M. Hermenegildo and G. Puebla, editors, *Proceedings of Ninth International Static Analysis Symposium*, volume 2477 of *Lecture Notes in Computer Science*, pages 85–101, 2002.
20. K. Marriott and H. Søndergaard. Precise and efficient groundness analysis for logic programs. *ACM Letters on Programming Languages and Systems*, 2(1–4):181–196, 1993.
21. N. Mazur, G. Janssens, and M. Bruynooghe. A module based analysis for memory reuse in Mercury. In J.W. Lloyd et. al, editor, *Proceedings of the First International Conference on Computational Logic*, volume 1861 of *Lecture Notes in Computer Science*, pages 1255–1269. Springer, 2000.
22. N. Mazur, G. Janssens, and W. Vanhoof. Collecting potential optimizations. In M. Leuschel and F. Bueno, editors, *LOPSTR 2002, Logic-based Program Synthesis and Transformation, Revised Selected Papers*, volume 2664 of *Lecture Notes in Computer Science*, pages 109–110. Springer, 2002.
23. D. Pedreschi and S. Ruggieri. Weakest preconditions for pure Prolog programs. *Information Processing Letters*, 67(3):145–150, 1998.
24. H. Søndergaard. An application of abstract interpretation of logic programs: occur check problem. In B. Robinet and R. Wilhelm, editors, *ESOP 86, European Symposium on Programming*, volume 213 of *Lecture Notes in Computer Science*, pages 324–338. Springer, 1986.
25. P. Wadler and R.J.M. Hughes. Projections for strictness analysis. In *Proceedings of the Third International Conference on Functional Programming Languages and Computer Architecture*, volume 274 of *Lecture Notes in Computer Science*, pages 385–407, 1987.