Boolean Functions for Pair Sharing Analysis

Lunjin Lu and Xuan Li

Oakland University
Rochester, MI 48309, USA
{l2lu,x2li}@oakland.edu

Abstract. This paper presents an encoding scheme for pair sharing. Pair sharing relations and abstract operations required for a pair sharing analysis are encoded as Boolean functions and operations on Boolean functions. Preliminary experimental results with a prototype implementation are presented.

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.

Sharing analyses make use of two abstract domains: set sharing [17] and pair sharing [24]. Pair sharing captures sharing and linearity while set sharing captures sharing and groundness dependencies. Both set and pair sharing domains have been widely studied. Most applications require pair sharing though this information can be extracted from set sharing. Pair sharing has much simpler domain structure and abstract operations, which renders a possibility for an efficient implementation of pair sharing. This paper explores this possibility. An encoding scheme is presented that encodes pair sharing relations in Boolean functions and abstract operations in operations on Boolean functions. To the best of our knowledge, this is the first encoding of pair sharing relations in terms of Boolean functions.

Encoding pair sharing in Boolean functions makes it easy to compose a pair sharing analysis with other analyses. Many analyses for logic programs and constraint logic programs use Boolean functions to express dependencies between program variables. In groundness analysis [11, 22, 15, 8, 1, 19, 14, 6], the formula $x \wedge (y \leftrightarrow z)$ describes a program state in which x is bound to a ground term, and there is a dependency between y and z such that whenever y becomes ground so does z and vice versa. Boolean functions have also been used to express other properties such as definiteness [3], finiteness [2], termination [13] and set sharing [7]. Using Boolean functions as a uniform underlying representation for pair sharing and those other properties provides a leverage for composing a pair sharing analysis with those analyses using a standard domain composition technique [12] such as reduced product and functional dependency [10].

Encoding pair sharing in Boolean functions also enables implementors to benefit from efficient data structures for Boolean functions such as binary decision diagrams (BDD in short) [4] that have been proven to have excellent performance in practice.

The remainder of the paper is organised as follows. Section 2 contains basic concepts in logic programming, abstract interpretation and Boolean functions. Section 3 recalls the pair sharing domain and presents an encoding of pair sharing relations in Boolean functions. Section 4 recalls operations on pair sharing relations and encode these operations in operations on Boolean functions. The correctness of the encoding is proved. Section 5 concludes with a discussion on related work.

2 Preliminaries

This section recalls some basic concepts in logic programming and abstract interpretation. The reader is referred to [20] and [9] for more detailed exposition.

Assume that there are a set Σ of function symbols and a denumerable set $\mathcal{V}$ of variables. Let $\mathbf{V}(o)$ be the set of variables in the syntactic object o. Let $V \subseteq \mathcal{V}$. $\mathsf{Term}(V)$ denotes the set of terms that can be constructed from Σ and V. An equation over V is a formula of the form $t_1 = t_2$ with $t_1, t_2 \in \mathsf{Term}(V)$. Let Eqn_V denote the set of all equations over V and Sub_V the set of substitutions θ such that $\mathbf{V}(\theta) \subseteq V$. Given $E \in \wp(Eqn_V)$, $mgu : \wp(Eqn_V) \mapsto Sub_V \cup \{fail\}$ returns either a most general unifier for E if E is unifiable or *fail* otherwise. For brevity, let $mgu(t_1, t_2) = mgu(\{t_1 = t_2\})$. We will sometimes treat a substitution as a set of equations. The function composition operation $\circ$ is defined as $f \circ g = \lambda x.f(g(x))$. Let P be the program to analyse and $\mathcal{V}_P$ be the set of the variables occurring in P. We will use a fixed renaming substitution Ψ such that $\Psi(\mathcal{V}_P) \cap \mathcal{V}_P = \emptyset$. Ψ, called a tagging substitution in [23], is used to rename both concrete and abstract objects over $\mathcal{V}_P$. Let $\mathcal{V}_P^{\ddagger} = \mathcal{V}_P \cup \Psi(\mathcal{V}_P)$.

Let $p(s)$ be a call. We say that the program point left to $p(s)$ is the call-site of $p(s)$ and that the program point right to $p(s)$ is the return-site of $p(s)$. Let C be a clause. The leftmost program point associated with C is called the entry point of C and the rightmost program point associated with C the exit point of C.

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 $\langle A, \sqsubseteq^A \rangle$ be a poset. A function $\gamma : A \mapsto C$ is a concretization function iff γ is a monotone and $\gamma(A)$ is a Moore family. A concretization function from A to C induces a Galois connection between A and C [9]. The induced adjoint, called an abstraction function, is $\alpha(c) = \sqcap^A \{a \in A \mid c \sqsubseteq^C \gamma(a)\}$. In an analysis by abstract interpretation, there are two semantics: concrete and abstract semantics. The concrete semantics on a concrete domain $\langle C, \sqsubseteq^C \rangle$ is defined in terms of a number of concrete operations $\mathcal{C}$ on C. The abstract semantics on an abstract domain $\langle A, \sqsubseteq^A \rangle$ is usually derived from the concrete semantics such that there is an abstract operation $\mathcal{A}$ on A corresponding to each concrete operation $\mathcal{C}$ on C. The correctness of the analysis

is guaranteed by requiring each abstract operation $\mathcal{A}$ safely approximates its corresponding concrete operation $\mathcal{C}$ with respect to a concretization function γ from A to C, i.e., $\mathcal{C}(\gamma(a_1), \cdots, \gamma(a_k)) \sqsubseteq^C \gamma(\mathcal{A}(a_1, \cdots, a_k))$ for all $a_1, \cdots, a_k \in A$.

3 Encoding Sharing Relations

This section first recalls the pair sharing domain and then presents an encoding of pair sharing relations in Boolean functions. The benefits of express pair sharing relations in Boolean functions are twofold. Firstly, integration of pair sharing analysis with groundness analysis [11, 22, 15, 8, 1, 19, 14, 6], set sharing analysis [7], definiteness analysis [3] and finiteness [2] is facilitated because Boolean functions have been used to encode those properties. Secondly, encoding pair sharing relations in Boolean functions make available to implementator efficient data structures such as BDD.

3.1 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 $\theta(s)$ and $\theta(t)$ share. The possible sharing and linearity of variables under a substitution θ over a set V of variables are represented as a symmetric relation $\pi \subseteq V \times V$ [24]. We follow [8] to parameterise abstract domains with finite sets of variables. Let PS_V be the set of symmetric relations over V. The abstract domain for sharing and linearity, dubbed pair sharing, is $\langle PS_V, \subseteq, \emptyset, V^2, \cap, \cup \rangle$ which is a complete lattice. A Galois connection between $\langle PS_V, \subseteq \rangle$ and $\langle \wp(Sub), \subseteq \rangle$ is obtained as follows [5].

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

Example 1. Let $V = \{x_1, x_2, x_3\}$, $\pi = \{\langle x_1, x_2 \rangle, \langle x_2, x_1 \rangle\}$, $\theta_0 = \{x_1 \mapsto g(u), x_2 \mapsto h(v)\}$, $\theta_1 = \{x_1 \mapsto f(u, v), x_2 \mapsto h(v, w)\}$ and $\theta_2 = \{x_2 \mapsto f(x_3, u)\}$. We have $\theta_0 \in \gamma_V(\pi)$ since $\alpha_V(\{\theta_0\}) = \emptyset \subseteq \pi$; and $\theta_1 \in \gamma_V(\pi)$ since $\alpha_V(\{\theta_1\}) = \{\langle x_1, x_2 \rangle, \langle x_2, x_1 \rangle\} = \pi$. Also, $\alpha_V(\{\theta_2\}) = \{\langle x_2, x_3 \rangle, \langle x_3, x_2 \rangle\} \not\subseteq \pi$, hence, $\theta_2 \notin \gamma_V(\pi)$. ∎

We will write $(u \leftrightarrow v) \in \pi$ to stand for $\{\langle u, v \rangle, \langle v, u \rangle\} \subseteq \pi$. 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 $\{x\} \otimes \{y, z\} = \{x \leftrightarrow y, x \leftrightarrow z\}$.

In a pair sharing analysis, two instances of PS_V will be used: $PS_{\mathcal{V}_P}$ and $PS_{\mathcal{V}_P^\sharp}$. $PS_{\mathcal{V}_P}$ contains sharing relations that are associated with textual points in the program while the sharing relations in $PS_{\mathcal{V}_P^\sharp}$ are used as intermediate results during the propagation of sharing across procedure boundaries.

3.2 Encoding Sharing Relations

We now present a scheme that encodes pair sharing relations in Boolean functions. Since the class of Boolean functions is isomorphic to that of Boolean formulae modulo logical equivalence, we will use Boolean functions and Boolean formulae interchangeably.

Two variables in a sharing pair are encoded using disjoint sets of Boolean variables. We first consider the encoding of the left component of the pair. Let b be a Boolean variable. Both b and $\neg b$ are Boolean literals. Let $k = \lceil \log_2 |\mathcal{V}_P| \rceil$ where $\lceil \cdot \rceil$ is the ceiling function. Program variables in $\mathcal{V}_P$ are encoded as min-terms of $k+1$ Boolean variables $b_0, b_1, \cdots, b_k$ with b_0 always occurring positively. Min-terms for different variables are different. Let $\mathcal{L}(x)$ be the code of the program variable x as the left component of a sharing pair.

Example 2. For instance, let $\mathcal{V}_P = \{x_1, x_2, x_3\}$. Then $k = 2$ and the variables in $\mathcal{V}_P$ may be encoded as follows: $\mathcal{L}(x_1) = b_0 \wedge \overline{b_1} \wedge \overline{b_2}$, $\mathcal{L}(x_2) = b_0 \wedge \overline{b_1} \wedge b_2$ and $\mathcal{L}(x_3) = b_0 \wedge b_1 \wedge \overline{b_2}$ where b_1 and b_2 are Boolean variables. ∎

The code of a tagged program variable $\Psi(x)$ is obtained from that of x by changing b_0 into $\overline{b_0}$. Thus, the code of $\Psi(x)$ is the same as that of x except that b_0 occurs positively in the code of x and negatively in the code of $\Psi(x)$. $\mathcal{L}(\Psi(x)) = \mathcal{L}(x)[b_0/\overline{b_0}]$ where $\mathcal{L}(x)[b_0/\overline{b_0}]$ results from replacing b_0 in $\mathcal{L}(x)$ with $\overline{b_0}$. Note that $\mathcal{L}(x) = \mathcal{L}(\Psi(x))[\overline{b_0}/b_0]$. Continuing with example 2, we obtain $\mathcal{L}(\Psi(x_2)) = \mathcal{L}(x_2)[b_0/\overline{b_0}] = \overline{b_0} \wedge \overline{b_1} \wedge b_2$.

The right component of a sharing pair is encoded as a Boolean formula over $c_0, c_1, \cdots, c_k$ and is defined $\mathcal{R}(x) = \mathcal{L}(x)[b_0/c_0 \cdots b_k/c_k]$ where $\{c_0, \cdots, c_k\}$ is disjoint with $\{b_0, \cdots, b_k\}$. Continuing with example 2, we have $\mathcal{R}(\Psi(x_2)) = \mathcal{L}(\Psi(x_2))[b_0/c_0, b_1/c_1, b_2/c_2] = (\overline{b_0} \wedge \overline{b_1} \wedge b_2)[b_0/c_0, b_1/c_1, b_2/c_2] = \overline{c_0} \wedge \overline{c_1} \wedge c_2$. Let π be a sharing relation and $\langle x, y \rangle \in \pi$. The code of $\langle x, y \rangle$ is $\mathcal{L}(x) \wedge \mathcal{R}(y)$. The code of a sharing relation π is the disjunction of the codes of the pairs contained in π: $\mathcal{E}(\pi) = \bigvee_{\langle x,y \rangle \in \pi} \mathcal{L}(x) \wedge \mathcal{R}(y)$.

Example 3. Continuing example 2. Let $\pi = \{\langle x_1, \Psi(x_3) \rangle, \langle \Psi(x_3), x_1 \rangle, \langle x_2, x_2 \rangle\}$. $\mathcal{E}(\pi) = b_0 \wedge \overline{b_1} \wedge \overline{b_2} \wedge \overline{c_0} \wedge c_1 \wedge \overline{c_2} \vee \overline{b_0} \wedge b_1 \wedge \overline{b_2} \wedge c_0 \wedge \overline{c_1} \wedge \overline{c_2} \vee b_0 \wedge \overline{b_1} \wedge b_2 \wedge c_0 \wedge \overline{c_1} \wedge c_2$. ∎

Since the code $\mathcal{L}(x) \wedge \mathcal{R}(y)$ of a pair $\langle x, y \rangle$ is a min-term of Boolean variables $b_0, \cdots, b_k, c_0, \cdots, c_k$, it has a unique model denoted by $\mathcal{M}(x, y)$. Continuing example 2, $\mathcal{R}(x_2) = c_0 \wedge \overline{c_1} \wedge c_2$ and the min-term $\mathcal{L}(x_1) \wedge \mathcal{R}(x_2) = b_0 \wedge \overline{b_1} \wedge \overline{b_2} \wedge c_0 \wedge \overline{c_1} \wedge c_2$ has the unique model $\mathcal{M}(x, y) = \{b_0 \mapsto \mathit{true}, b_1 \mapsto \mathit{false}, b_2 \mapsto \mathit{false}, c_0 \mapsto \mathit{true}, c_1 \mapsto \mathit{false}, c_2 \mapsto \mathit{true}\}$.

Proposition 1. *$\langle x, y\rangle \in \pi$ iff $\mathcal{M}(x, y) \models \mathcal{E}(\pi)$ for any sharing relation π.* ∎

An immediate corollary of proposition 1 is that no two different sharing relations have the same code, and hence a sharing relation can be extracted from its code.

4 Encoding Abstract Operations

This section first recalls abstract operations on pair sharing relations and then encodes these abstract operations in operations on Boolean functions.

4.1 Abstract Operations

We first recall abstract operations that a sharing analysis uses, beginning with simpler ones. The constant $\emptyset$ - the bottom element of $\langle PS_{\mathcal{V}_P}, \subseteq\rangle$ is the initial sharing relation associated with each program point when the least fixpoint algorithm commences. It also describes the sharing relation in the identity substitution. The least upper bound operation $\cup$ on $PS_{\mathcal{V}_P}$ is used to combine sharing relations that come to a program point from different control-flow paths. The equality test operation $=$ on $PS_{\mathcal{V}_P}$ is used to check if sharing information associated with a program point need be propagated to other program points along control-flow.

The abstract unification operation $amgu : \mathsf{Term}(\mathcal{V}_P^\ddagger) \times \mathsf{Term}(\mathcal{V}_P^\ddagger) \times PS_{\mathcal{V}_P^\ddagger} \mapsto PS_{\mathcal{V}_P^\ddagger}$ unifies two terms s and t in a sharing relation π and returns a sharing relation π' such that $mgu(\theta(s), \theta(t)) \circ \theta$ is described by π' whenever θ is described by π. The following predicate $\chi : \mathsf{Term}(\mathcal{V}_P^\ddagger) \times PS_{\mathcal{V}_P^\ddagger} \mapsto \{true, false\}$ will be used in the 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 in π for any $\theta \in \gamma(\pi)$ [5]. The formula $\chi(t, \pi) = true$ is abbreviated as $\chi(t, \pi)$. The following definition of *amgu* is derived from one given in [18].

$$amgu(s, t, \pi) =$$
$$\begin{cases} \{\langle x, y\rangle \in \pi \mid x \notin \mathbf{V}(s) \wedge y \notin \mathbf{V}(s)\} & \text{if } \mathbf{V}(t) = \emptyset \\ \{\langle x, y\rangle \in \pi \mid x \notin \mathbf{V}(t) \wedge y \notin \mathbf{V}(t)\} & \text{if } \mathbf{V}(s) = \emptyset \\ \pi \cup link(s, t, \pi) \cup (\chi(t, \pi) \rhd link(s, s, \pi)) \cup (\chi(s, \pi) \rhd link(t, t, \pi)) & \text{otherwise} \end{cases}$$

where $link : \mathsf{Term}(\mathcal{V}_P^\ddagger) \times \mathsf{Term}(\mathcal{V}_P^\ddagger) \times PS_{\mathcal{V}_P^\ddagger} \mapsto PS_{\mathcal{V}_P^\ddagger}$ is defined $link(s, t, \pi) = \{u \leftrightarrow v \mid x \in \mathbf{V}(s) \wedge x \overset{\pi}{\Leftrightarrow} u \wedge v \overset{\pi}{\Leftrightarrow} y \wedge y \in \mathbf{V}(t)\}$ and $\rhd : \{false, true\} \times PS_{\mathcal{V}_P^\ddagger} \mapsto PS_{\mathcal{V}_P^\ddagger}$ is defined $B \rhd \pi =$ (if B then π else $\emptyset$) . Observe that if $s, t \in \mathsf{Term}(\mathcal{V}_P)$ and $\pi \in PS_{\mathcal{V}_P}$ then $amgu(s, t, \pi) \in PS_{\mathcal{V}_P}$.

Example 4. Let $s = \Psi(x3)$, $t = [x1|x3]$ and $\pi = \{\Psi(x2) \leftrightarrow x1, \Psi(x2) \leftrightarrow x2, \Psi(x3) \leftrightarrow x1, \Psi(x3) \leftrightarrow x2, \Psi(x2) \leftrightarrow \Psi(x3)\}$. We have $\mathbf{V}(t) = \{x1, x3\} \neq \emptyset$, $\mathbf{V}(s) = \{\Psi(x3)\} \neq \emptyset$, $\chi(s, \pi) = false$, $\chi(t, \pi) = false$. So,

$$amgu(s, t, \pi) = \pi \cup link(s, t, \pi)$$

We have

$$\pi' = link(s,t,\pi) = \left\{ \begin{array}{c} x1 \leftrightarrow x1, x1 \leftrightarrow x2, x2 \leftrightarrow x3, x3 \leftrightarrow x1 \\ \Psi(x2) \leftrightarrow x1, \Psi(x2) \leftrightarrow x2, \Psi(x2) \leftrightarrow x3, \\ \Psi(x2) \leftrightarrow \Psi(x2), \Psi(x2) \leftrightarrow \Psi(x3), \\ \Psi(x3) \leftrightarrow x1, \Psi(x3) \leftrightarrow x2, \Psi(x3) \leftrightarrow x3, \Psi(x3) \leftrightarrow \Psi(x3), \end{array} \right\}$$

Note that $\pi \subseteq \pi'$. Thus, $amgu(s,t,\pi) = \pi'$. ∎

Abstract procedure-entry operation $entry : \mathsf{Term}(\mathcal{V}_P) \times PS_{\mathcal{V}_P} \times \mathsf{Term}(\mathcal{V}_P) \mapsto PS_{\mathcal{V}_P}$ propagates sharing information at a call-site to the entry point of a clause. Let π be the sharing relation at the call-site of a call $p(s)$ and C' a clause with head $p(t)$. The sharing relation at the entry point of C' is obtained by $entry(s,\pi,t)$. Abstract procedure-exit operation $exit : \mathsf{Term}(\mathcal{V}_P) \times PS_{\mathcal{V}_P} \times \mathsf{Term}(\mathcal{V}_P) \times PS_{\mathcal{V}_P} \mapsto PS_{\mathcal{V}_P}$ propagates sharing information at the exit point of a clause to the return-site of a call. Let σ be the sharing relation at the call-site of a call $p(t)$ and C' a clause with head $p(s)$ and π be sharing relation at the exit point of C'. The sharing relation at the return-site of $p(t)$ is obtained by $exit(s,\pi,t,\sigma)$.

We first introduce two auxiliary operations before giving definitions for *entry* and *exit*. The tagging operation $\Psi : PS_{\mathcal{V}_P} \mapsto PS_{\mathcal{V}_P^\ddagger}$ tags each variable in each pair in its argument: $\Psi(\pi) = \{\langle \Psi(x), \Psi(y)\rangle \mid \langle x,y\rangle \in \pi\}$. The tagging operation is used as the first step in both *entry* and *exit*. The projection operation $proj : PS_{\mathcal{V}_P^\ddagger} \mapsto PS_{\mathcal{V}_P}$ removes from its argument all the sharing links incident to variables outside $\mathcal{V}_P$: $proj(\pi) = \pi \cap (\mathcal{V}_P \times \mathcal{V}_P)$. The projection operation is used as the last step in both *entry* and *exit*. The following definitions are adapted from [21]

$$\begin{aligned} entry(s,\pi,t) &= proj \circ amgu(mgu(\Psi(s),t), \Psi(\pi)) \\ exit(s,\pi,t,\sigma) &= proj \circ amgu(mgu(\Psi(s),t), \Psi(\pi) \cup \sigma) \end{aligned}$$

where $amgu : \wp(Eqn_{\mathcal{V}_P^\ddagger}) \times PS_{\mathcal{V}_P^\ddagger} \mapsto PS_{\mathcal{V}_P^\ddagger}$ is by $amgu(\emptyset,\pi) = \pi$ and $amgu(\{s = t\} \cup E, \pi) = amgu(E, amgu(s,t,\pi))$.[1]

Example 5. Let $\mathcal{V}_P = \{x_1, x_2, x_3\}$, $s_0 = p(x2, x3)$ and $t_0 = p([x1|x2], [x1|x3])$ and $\pi_0 = \{x2 \leftrightarrow x3\}$. Then $\Psi(s_0) = p(\Psi(x2), \Psi(x3))$, $mgu(\Psi(s_0), t_0) = \{\Psi(x2) = [x1|x2], \Psi(x3) = [x1|x3]\}$ and $\Psi(\pi_0) = \{\Psi(x2) \leftrightarrow \Psi(x3)\}$. So,

$$\begin{aligned} &entry(s_0, \pi_0, t_0) \\ &= proj \circ amgu(\{\Psi(x2) = [x1|x2], \Psi(x3) = [x1|x3]\}, \{\Psi(x2) \leftrightarrow \Psi(x3)\}) \\ &= proj \circ amgu(\Psi(x3), [x1|x3], \pi_1) \end{aligned}$$

where

$$\begin{aligned} \pi_1 &= amgu(\Psi(x2), [x1|x2], \{\Psi(x2) \leftrightarrow \Psi(x3)\})) \\ &= \left\{ \begin{array}{c} (\Psi(x2) \leftrightarrow x1), (\Psi(x2) \leftrightarrow x2), \\ (\Psi(x3) \leftrightarrow x1), (\Psi(x3) \leftrightarrow x2), (\Psi(x2) \leftrightarrow \Psi(x3)) \end{array} \right\} \end{aligned}$$

[1] The operations *entry* and *exit* are applied only when $mgu(\Psi(s),t) \neq fail$.

and π_1 is equal to π in example 4. By example 4, $entry(s_0, \pi_0, t_0) = proj(\pi')$ with π' given therein. Therefore, $entry(s_0, \pi_0, t_0) = \{x1 \leftrightarrow x1, x1 \leftrightarrow x2, x2 \leftrightarrow x3, x3 \leftrightarrow x1\}$. ∎

4.2 Encoding Abstract Operations

The code of an operation $\mathcal{O}$ is denoted $\mathcal{O}^\sharp$ which is an operation over Boolean functions. $\mathcal{O}^\sharp$ is a correct code of $\mathcal{O}$ iff $\mathcal{E} \circ \mathcal{O}(\pi_1, \cdots, \pi_m) = \mathcal{O}^\sharp(\mathcal{E}(\pi_1), \cdots, \mathcal{E}(\pi_m))$.

The lattice operations over PS_V have straightforward codes: $\emptyset^\sharp = false$, $\cup^\sharp = \vee$ and $\cap^\sharp = \wedge$ and $=^\sharp$ is $\Leftrightarrow$. The remainder of this section constructs codes for other operations commencing with simpler ones. Let $f = \mathcal{E}(\pi)$.

$$\begin{aligned}\mathcal{E}(\chi(t,\pi)) &= \mathcal{E}(\neg linear(t) \vee ((\mathbf{V}(t)^2 \cap \pi) \neq \emptyset)) \\ &= \neg linear(t) \vee (\mathcal{E}(\mathbf{V}(t)^2 \cap \pi) \not\Leftrightarrow false) \\ &= \neg linear(t) \vee (\mathcal{E}(\mathbf{V}(t)^2) \wedge \mathcal{E}(\pi) \not\Leftrightarrow false) \\ &= \neg linear(t) \vee (\mathcal{E}(\mathbf{V}(t)^2) \wedge f \not\Leftrightarrow false)\end{aligned}$$

Thus, $\chi^\sharp(t, f) = linear(t) \vee (\mathcal{E}(\mathbf{V}(t)^2) \wedge f) \not\Leftrightarrow false)$ is a correct code of χ. The code $\otimes^\sharp$ of $\otimes$ is $X \otimes^\sharp Y = \mathcal{E}(X \otimes Y) = (\bigvee_{u \in X, v \in Y} \mathcal{L}(u) \wedge \mathcal{R}(v)) \vee (\bigvee_{u \in Y, v \in X} \mathcal{L}(u) \wedge \mathcal{R}(v))$.

Now consider $link(s, t, \pi)$. By rewriting, $link(s, t, \pi) = \sigma \cup (\pi \bowtie \sigma) \cup (\sigma \bowtie \pi) \cup (\pi \bowtie \sigma \bowtie \pi)$ where $\sigma = \mathbf{V}(s) \otimes \mathbf{V}(t)$ and $\pi_1 \bowtie \pi_2 = \{\langle u, v\rangle \mid \exists w.(\langle u, w\rangle \in \pi_1 \wedge \langle w, v\rangle \in \pi_2)\}$. Observe that $\bowtie$ is associative. Encoding *link* reduces to encoding $\bowtie$ since $\cup^\sharp = \vee$. Let $\{d_0, \cdots, d_k\}$ be disjoint with $\{b_0, \cdots, b_k\}$ and $\{c_0, \cdots, c_k\}$. Define

$$f \bowtie^\sharp g = \exists d_0. \cdots \exists d_k.(f[c_0/d_0 \cdots c_k/d_k] \wedge g[b_0/d_0 \cdots b_k/d_k])$$

It can be readily proved that $\bowtie^\sharp$ is associative. We now prove that $\bowtie^\sharp$ is a correct code of $\bowtie$.

Lemma 1. *For any $\pi_1, \pi_2 \in PS_V$ where $V \in \{\mathcal{V}_P, \mathcal{V}_P^\ddagger\}$, $\mathcal{E}(\pi_1 \bowtie \pi_2) = \mathcal{E}(\pi_1) \bowtie^\sharp \mathcal{E}(\pi_2)$.* ∎

The following lemma provides a correct code of *link*.

Lemma 2. *For any $s, t \in \mathsf{Term}(V)$ and any $\pi \in PS_V$ where $V \in \{\mathcal{V}_P, \mathcal{V}_P^\ddagger\}$,*

$$link^\sharp(s, t, f) = \begin{cases} let & g = \mathbf{V}(s) \otimes^\sharp \mathbf{V}(t) \\ in & g \vee (f \bowtie^\sharp g) \vee (g \bowtie^\sharp f) \vee (f \bowtie^\sharp g \bowtie^\sharp f) \end{cases}$$

∎

Example 6. Let $\mathcal{V}_P = \{x1, x2\}$, $s = \Psi(x1)$, $t = h(x1, x2)$ and $\pi = (\Psi(x1) \leftrightarrow \Psi(x1))$. Let $\mathcal{L}(x1) = b_0 \wedge \overline{b_1}$ and $\mathcal{L}(x2) = b_0 \wedge b_1$ and $f = \mathcal{E}(\pi) = \overline{b_0} \wedge \overline{b_1} \wedge \overline{c_0} \wedge \overline{c_1}$. Then $link^\sharp(s, t, f) = g \vee (f \bowtie^\sharp g) \vee (g \bowtie^\sharp f) \vee (f \bowtie^\sharp g \bowtie^\sharp f)$ where $g =$

$\mathbf{V}(s) \otimes^\sharp \mathbf{V}(t) = \overline{b_0} \wedge \overline{b_1} \wedge c_0 \wedge \overline{c_1} \vee b_0 \wedge \overline{b_1} \wedge \overline{c_0} \wedge \overline{c_1} \vee \overline{b_0} \wedge \overline{b_1} \wedge c_0 \wedge c_1 \vee b_0 \wedge b_1 \wedge \overline{c_0} \wedge \overline{c_1}$. We obtain $f \bowtie^\sharp g$ as follows.

$$\begin{aligned} f \bowtie^\sharp g &= \exists d_0.\exists d_1.\left(\overline{b_0} \wedge \overline{b_1} \wedge \overline{d_0} \wedge \overline{d_1} \wedge \begin{pmatrix} \overline{d_0} \wedge \overline{d_1} \wedge c_0 \wedge \overline{c_1} \vee d_0 \wedge \overline{d_1} \wedge \overline{c_0} \wedge \overline{c_1} \\ \vee\ \overline{d_0} \wedge \overline{d_1} \wedge c_0 \wedge c_1 \vee d_0 \wedge d_1 \wedge \overline{c_0} \wedge \overline{c_1} \end{pmatrix}\right) \\ &= \overline{b_0} \wedge \overline{b_1} \wedge c_0 \wedge \overline{c_1} \ \vee\ \overline{b_0} \wedge \overline{b_1} \wedge c_0 \wedge c_1 \end{aligned}$$

Similarly, $g \bowtie^\sharp f = b_0 \wedge \overline{b_1} \wedge \overline{c_0} \wedge \overline{c_1} \vee b_0 \wedge b_1 \wedge \overline{c_0} \wedge \overline{c_1}$ and $f \bowtie^\sharp g \bowtie^\sharp f = \mathit{false}$. So, $\mathit{link}^\sharp(s,t,f) = \overline{b_0} \wedge \overline{b_1} \wedge c_0 \wedge \overline{c_1} \vee b_0 \wedge \overline{b_1} \wedge \overline{c_0} \wedge \overline{c_1} \vee \overline{b_0} \wedge \overline{b_1} \wedge c_0 \wedge c_1 \vee b_0 \wedge b_1 \wedge \overline{c_0} \wedge \overline{c_1}$. ∎

The code of the tagging operation Ψ simply changes b_0 in its argument into $\overline{b_0}$ and c_0 into $\overline{c_0}$: $\Psi^\sharp(f) = f[b_0/\overline{b_0}, c_0/\overline{c_0}]$. The code of the projection operation *proj* falsifies min-terms with a negative occurrence of b_0 or c_0 in its argument: $\mathit{proj}^\sharp(f) = b_0 \wedge c_0 \wedge f$. The correctness of $\Psi^\sharp$ and $\mathit{proj}^\sharp$ follows immediately from the definitions for Ψ, *proj* and $\mathcal{E}$. Observe that $B \wedge \mathcal{E}(\pi) = \mathcal{E}(B \rhd \pi)$ for any sharing relation π, implying that $B \rhd^\sharp f = B \wedge f$ is a correct code of $\rhd$. The following theorem provides correct codes of the abstract operations *amgu*, *entry* and *exit*.

Theorem 1. *For any $s', t' \in \mathsf{Term}(\mathcal{V}_P^\ddagger)$, any $s, t \in \mathsf{Term}(\mathcal{V}_P)$, any code h of a sharing relation in $PS_{\mathcal{V}_P^\ddagger}$, any codes f, g of sharing relations in $PS_{\mathcal{V}_P}$,*

$$\mathit{amgu}^\sharp(s',t',h) = \begin{cases} h \wedge ((\mathcal{V}_P^\ddagger \setminus \mathbf{V}(s')) \otimes^\sharp (\mathcal{V}_P^\ddagger \setminus \mathbf{V}(s'))) & \text{if } \mathbf{V}(t') = \emptyset \\ h \wedge ((\mathcal{V}_P^\ddagger \setminus \mathbf{V}(t')) \otimes^\sharp (\mathcal{V}_P^\ddagger \setminus \mathbf{V}(t'))) & \text{if } \mathbf{V}(s') = \emptyset \\ h \vee \mathit{link}^\sharp(s',t',h) \vee (\chi^\sharp(t',h) \wedge \mathit{link}^\sharp(s',s',h)) \vee (\chi^\sharp(s',h) \wedge \mathit{link}^\sharp(t',t',h)) & \text{otherwise} \end{cases}$$

$$\mathit{entry}^\sharp(s,f,t) = \mathit{proj}^\sharp \circ \mathit{amgu}^\sharp(\mathit{mgu}(\Psi(s),t), \Psi^\sharp(f))$$

$$\mathit{exit}^\sharp(s,f,t,g) = \mathit{proj}^\sharp \circ \mathit{amgu}^\sharp(\mathit{mgu}(\Psi(s),t), \Psi^\sharp(f) \vee g)$$

where $\mathit{amgu}^\sharp(\emptyset, f) = f$ and $\mathit{amgu}^\sharp(\{s = t\} \cup E, f) = \mathit{amgu}^\sharp(E, \mathit{amgu}^\sharp(s,t,f))$. ∎

Example 7. Continue with example 6. We have $\chi^\sharp(s,f) = \mathit{true}$, $\chi^\sharp(t,f) = \mathit{false}$. So, $\mathit{amgu}^\sharp(s,t,f) = f \vee \mathit{link}^\sharp(s,t,f) \vee \mathit{link}^\sharp(t,t,f)$. $\mathit{link}(s,t,f)$ is calculated in example 6. Omitting details, we obtain $\mathit{link}^\sharp(t,t,f) = (b_0 \wedge \overline{b_1} \wedge c_0 \wedge \overline{c_1}) \vee (b_0 \wedge \overline{b_1} \wedge c_0 \wedge c_1) \vee (b_0 \wedge b_1 \wedge c_0 \wedge \overline{c_1}) \vee (b_0 \wedge b_1 \wedge c_0 \wedge c_1) \vee (\overline{b_0} \wedge \overline{b_1} \wedge \overline{c_0} \wedge \overline{c_1})$. Thus,

$$\mathit{amgu}^\sharp(s,t,h) = \begin{pmatrix} \overline{b_0} \wedge \overline{b_1} \wedge \overline{c_0} \wedge \overline{c_1} \ \vee\ \overline{b_0} \wedge \overline{b_1} \wedge c_0 \wedge \overline{c_1} \ \vee b_0 \wedge \overline{b_1} \wedge \overline{c_0} \wedge \overline{c_1} \\ \vee\ \overline{b_0} \wedge \overline{b_1} \wedge c_0 \wedge c_1) \vee\ b_0 \wedge b_1 \wedge \overline{c_0} \wedge \overline{c_1} \ \vee b_0 \wedge \overline{b_1} \wedge c_0 \wedge \overline{c_1} \\ \vee\ b_0 \wedge \overline{b_1} \wedge c_0 \wedge c_1 \ \vee b_0 \wedge b_1 \wedge c_0 \wedge \overline{c_1}) \vee b_0 \wedge b_1 \wedge c_0 \wedge c_1 \end{pmatrix}$$

that is the code of $\{\Psi(x1) \leftrightarrow \Psi(x1), x1 \leftrightarrow \Psi(x1), x2 \leftrightarrow \Psi(x1), x1 \leftrightarrow x1, x1 \leftrightarrow x2, x2 \leftrightarrow x2\}$. ∎

4.3 Prototype Implementation

We have implemented in C++ a prototype pair sharing analyzer in order to demonstrate the viability of the encoding scheme. The Boolean functions are implemented using a BDD package developed by Schaechte. The analyzer has been run on benchmark programs. As expected, the analyzer produces the correct result on each program we have tested. Table 1 contains timing data for medium sized programs. Each row corresponds to a program. The first two entries are the name of the program and the number of program points in the program. Next two entries are analysis time in seconds and analysis time per program point in seconds. The average analysis time per porgram point is 0.0025 seconds, which is fast. The same table also shows that the encoding results in better time performance. For the sake of comparison, we also implemented another analyzer which is identical to the prototype analyzer except that pair sharing relations are expressed as bitmaps. The last two entries contain the analysis time and analysis time per program point taken by the bitmap-based analyzer. The bitmap-based analyzer takes an average of 0.0061 seconds per program point which is more than two times slower than the BDD-based analyzer.

Table 1. Time Performance

CPU: Intel(R) Pentium(R) 4 CPU 2.40GHz. Operating System: Linux 3.2.8					
		BDD		Bit Map	
Program	Size	Time	PTime	Time	PTime
sdda.pl	171	0.3	0.0017	0.55	0.0032
kalah.pl	220	0.2	0.0009	0.21	0.0009
ann.pl	70	0.04	0.0005	0.04	0.0005
asm.pl	515	0.03	0.00005	0.02	0.00003
disj_r.pl	127	0.1	0.0007	0.17	0.0013
chat.pl	840	13.07	0.015	12.41	0.014
zebra.pl	43	0.01	0.0002	0.01	0.0002
tsp.pl	86	0.27	0.003	0.48	0.005
cs_r.pl	261	0.37	0.0014	1.13	0.004
life.pl	71	0.24	0.003	0.3	0.004
peep.pl	333	0.41	0.0012	0.72	0.002
nand.pl	343	3.07	0.0089	10.22	0.029
read.pl	386	0.95	0.002	1.48	0.0038
ga.pl	293	0.11	0.0003	0.19	0.0006

5 Discussion and Conclusion

We have presented an encoding sheme in which pair sharing relations are encoded in Boolean functions and abstract operations on pair sharing relations

in operations on Boolean functions. The encoding scheme is the first one for pair sharing despite the fact that Boolean functions have been used to express a variety of program properties for (constraint) logic programs.

In logic programming, Boolean functions were first used to trace groundness dependency between program variables. The domain *Def* of definite Boolean functions was first used in Dart [11] whilst the domain *Pos* of Positive Boolean functions first appeared in [22]. Both *Def* and *Pos* have since been widely studied [15, 8, 1, 19, 14, 6]. Boolean functions have also been used to express definiteness [3], finiteness [2] and termination [13] properties.

The most related work is by Codish, Søndergaard and Stuckey [7] who express set sharing in positive Boolean functions and encode abstract operations for set sharing analysis in operations on Boolean functions. They discover that the set sharing domain *Sharing* is isomorphic to *Pos*. Let X be the set of variables of interest. An element in *Sharing* is a set of sharing groups that are subsets of X. A positive Boolean function f is the code of a set sharing S iff $S = \{X \setminus G' \mid assign(G') \models f\}$ where $assign(G) = \lambda x \in X.$(if $x \in G$ then *true* else *false*). The code of the abstract unification operation makes use of an operation $\downarrow$ that mapes a positive Boolean function f to the smallest definite Boolean function g such that $f \models g$. A constructive definition of $\downarrow$ is given by Howe and King [16].

As further work, we would like to investigate how a pairing sharing analysis based on Boolean functions could be integrated with other analyses based on Boolean functions. We would also like to experimentally study how an implementation using Boolean functions to trace pair sharing compares with an implementation using a different representation of pair sharing.

Acknowledgements

Thanks are due to annonymous referees for their feedbacks on a previous draft of this paper. This work was supported by the National Science Foundation under grants CCR-0131862 and INT-0327760.

References

1. T. Armstrong, K. Marriott, P. Schachte, and H. Søndergaard. Two classes of Boolean functions for dependency analysis. *Science of Computer Programming*, 31(1):3–45, 1998.
2. R. Bagnara, R. Gori, P. M. Hill, and E. Zaffanella. Finite-tree analysis for constraint logic-based languages. In P. Cousot, editor, *Static Analysis: Proceedings of the 8th International Symposium*, volume 2126 of *Lecture Notes in Computer Science*, pages 165–184, Paris, France, 2001. Springer-Verlag.
3. P. Bigot, S. K. Debray, and K. Marriott. Understanding Finiteness Analysis Using Abstract Interpretation. In Krzysztof Apt, editor, *Proceedings of the Joint International Conference and Symposium on Logic Programming*, pages 735–749, Washington, USA, November 1992. The MIT Press.
4. R. Bryant. Symbolic Boolean manipulation with ordered binary-decision diagrams. *ACM Computing Survey*, 24(3):293–318, 1992.

5. 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.
6. Michael Codish. Worst-case groundness analysis using positive Boolean functions. *The Journal of Logic Programming*, 41(1):125–128, 1999.
7. Michael Codish, Harald Søndergaard, and Peter Stuckey. Sharing and groundness dependencies in logic programs. *ACM Transactions on Programming Languages and Systems*, 21(5):948–976, 1999.
8. A. Cortesi, G. Filé, and W. Winsborough. Optimal groundness analysis using propositional logic. *The Journal of Logic Programming*, 27(2):137–168, 1996.
9. 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.
10. P. Cousot and R. Cousot. Systematic design of program analysis frameworks. In *Principles of Programming Languages*, pages 269–282. The ACM Press, 1979.
11. P.W. Dart. On derived dependencies and connected databases. *The Journal of Logic Programming*, 11(2):163–188, 1991.
12. G. Filé, R. Giacobazzi, and F. Ranzato. A unifying view on abstract domain design. *ACM Computing Surveys*, 28(2), 1996.
13. 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.
14. Andy Heaton, Muhamed Abo-Zaed, Michael Codish, and Andy King. Simple, efficient and scalable groundness analysis of logic programs. *The Journal of Logic Programming*, 45(1-3):143–156, 2000.
15. P. Van Hentenryck, A. Cortesi, and B. Le Charlier. Evaluation of the Domain `PROP`. *The Journal of Logic Programming*, 23(3):237–278, 1995.
16. J. M. Howe and A. King. Efficient Groundness Analysis in Prolog. *Theory and Practice of Logic Programming*, 3(1):95–124, January 2003.
17. 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.
18. A. King. Pair-sharing over rational trees. *The Journal of Logic Programming*, 46(1–2):139–155, 2000.
19. A. King, J. Smaus, and P. Hill. Quotienting Share for dependency analysis. In S. D. Swierstra, editor, *Proc. of the 8th European Symposium on Programming*, volume 1576 of *Lecture Notes in Computer Science*, pages 59–73. Springer, 1999.
20. J.W. Lloyd. *Foundations of Logic Programming*. Springer-Verlag, 1987.
21. L. Lu. A mode analysis of logic programs by abstract interpretation. In D. Bjørner, M. Broy, and I.V. Pottosin, editors, *Perspectives of System Informatics*, volume 1181 of *Lecture Notes in Computer Science*, pages 362–373. Springer, 1996.
22. 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.
23. K. Muthukumar and M. Hermenegildo. Compile-time derivation of variable dependency using abstract interpretation. *The Journal of Logic Programming*, 13(1–4):315–347, 1992.

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.

A Proofs

LEMMA 1: For any $\pi_1, \pi_2 \in PS_V$ where $V \in \{\mathcal{V}_P, \mathcal{V}_P^\ddagger\}$, $\mathcal{E}(\pi_1 \bowtie \pi_2) = \mathcal{E}(\pi_1) \bowtie^\sharp \mathcal{E}(\pi_2)$.

Proof. We only prove the case for $V = \mathcal{V}_P^\ddagger$ for the case as $V = \mathcal{V}_P$ is similar. We have $\mathcal{E}(\pi_1) = \bigvee_{\langle x,z_1\rangle\in\pi_1} \mathcal{L}(x) \wedge \mathcal{R}(z_1)$, $\mathcal{E}(\pi_2) = \bigvee_{\langle z_2,y\rangle\in\pi_2} \mathcal{L}(z_2) \wedge \mathcal{R}(y)$ and $h = (\mathcal{E}(\pi_1)[c_0/d_0 \cdots c_k/d_k] \wedge \mathcal{E}(\pi_2)[b_0/d_0 \cdots b_k/d_k])$. Then

$$\begin{aligned}
\mathcal{E}(\pi_1)[c_0/d_0, \cdots, c_k/d_k] &= \bigvee_{\langle x,z_1\rangle\in\pi_1} \mathcal{L}(x) \wedge (\mathcal{R}(z_1)[c_0/d_0, \cdots, c_k/d_k]) \\
&= \bigvee_{\langle x,z_1\rangle\in\pi_1} \mathcal{L}(x) \wedge (\mathcal{L}(z_1)[b_0/d_0, \cdots, b_k/d_k]) \\
\mathcal{E}(\pi_2)[b_0/d_0, \cdots, b_k/d_k] &= \bigvee_{\langle z_2,y\rangle\in\pi_2} (\mathcal{L}(z_2)[b_0/d_0, \cdots, b_k/d_k]) \wedge \mathcal{R}(y)
\end{aligned}$$

Since $\mathcal{L}(z_1) \wedge \mathcal{L}(z_2) \Leftrightarrow \mathit{false}$ for $z_1 \neq z_2$, we obtain

$$\begin{aligned}
&\mathcal{E}(\pi_1)[c_0/d_0 \cdots c_k/d_k] \wedge \mathcal{E}(\pi_2)[b_0/d_0 \cdots b_k/d_k] \\
&= \bigvee_{\langle x,z\rangle\in\pi_1, \langle z,y\rangle\in\pi_2} \mathcal{L}(x) \wedge \mathcal{R}(y) \wedge (\mathcal{L}(z)[b_0/d_0, \cdots, b_k/d_k])
\end{aligned}$$

Thus,

$$\begin{aligned}
\mathcal{E}(\pi_1) \bowtie^\sharp \mathcal{E}(\pi_2) &= \exists d_0. \cdots \exists d_k. \mathcal{E}(\pi_1)[c_0/d_0 \cdots c_k/d_k] \wedge \mathcal{E}(\pi_2)[b_0/d_0 \cdots b_k/d_k] \\
&= \bigvee_{\langle x,z\rangle\in\pi_1, \langle z,y\rangle\in\pi_2} \mathcal{L}(x) \wedge \mathcal{R}(y) \\
&= \mathcal{E}(\pi_1 \bowtie \pi_2)
\end{aligned}$$

∎

LEMMA 2: For any $s, t \in \mathsf{Term}(V)$ and any $\pi \in PS_V$ where $V \in \{\mathcal{V}_P, \mathcal{V}_P^\ddagger\}$,

$$\mathit{link}^\sharp(s,t,f) = \begin{cases} \mathit{let} & g = \mathbf{V}(s) \otimes^\sharp \mathbf{V}(t) \\ \mathit{in} & g \vee (f \bowtie^\sharp g) \vee (g \bowtie^\sharp f) \vee (f \bowtie^\sharp g \bowtie^\sharp f) \end{cases}$$

Proof. Let $f = \mathcal{E}(\pi)$.

$\mathcal{E}(\mathit{link}(s,t,\pi))$

$$
\begin{aligned}
&= \begin{cases} let\ \sigma = \mathbf{V}(s) \otimes \mathbf{V}(t) \\ in\ \ \mathcal{E}(\sigma \cup (\pi \bowtie \sigma) \cup (\sigma \bowtie \pi) \cup (\pi \bowtie \sigma \bowtie \pi)) \end{cases} \\
&= \begin{cases} let\ \sigma = \mathbf{V}(s) \otimes \mathbf{V}(t) \\ in\ \ \mathcal{E}(\sigma) \vee (\mathcal{E}(\pi) \bowtie^\sharp \mathcal{E}(\sigma)) \vee (\mathcal{E}(\sigma) \bowtie^\sharp \mathcal{E}(\pi)) \vee (\mathcal{E}(\pi) \bowtie^\sharp \mathcal{E}(\sigma) \bowtie^\sharp \mathcal{E}(\pi))) \end{cases} \\
&= \begin{cases} let\ g = \mathbf{V}(s) \otimes^\sharp \mathbf{V}(t) \\ in\ \ g \vee (f \bowtie^\sharp g) \vee (g \bowtie^\sharp f) \vee (f \bowtie^\sharp g \bowtie^\sharp f) \end{cases} \\
&= \mathit{link}^\sharp(s,t,f) \\
&= \mathit{link}^\sharp(s,t,\mathcal{E}(\pi))
\end{aligned}
$$

∎

THEOREM 1: For any $s', t' \in \mathsf{Term}(\mathcal{V}_P^\ddagger)$, any $s, t \in \mathsf{Term}(\mathcal{V}_P)$, any code h of a sharing relation in $PS_{\mathcal{V}_P^\ddagger}$, any codes f, g of sharing relations in $PS_{\mathcal{V}_P}$,

$$
\mathit{amgu}^\sharp(s',t',h) = \begin{cases} h \wedge ((\mathcal{V}_P^\ddagger \setminus \mathbf{V}(s')) \otimes^\sharp (\mathcal{V}_P^\ddagger \setminus \mathbf{V}(s'))) & \text{if } \mathbf{V}(t') = \emptyset \\ h \wedge ((\mathcal{V}_P^\ddagger \setminus \mathbf{V}(t')) \otimes^\sharp (\mathcal{V}_P^\ddagger \setminus \mathbf{V}(t'))) & \text{if } \mathbf{V}(s') = \emptyset \\ h \vee \mathit{link}^\sharp(s',t',h) \vee (\chi^\sharp(t',h) \wedge \mathit{link}^\sharp(s',s',h)) \vee (\chi^\sharp(s',h) \wedge \mathit{link}^\sharp(t',t',h)) & \text{otherwise} \end{cases}
$$

$$
\begin{aligned}
\mathit{entry}^\sharp(s,f,t) &= \mathit{proj}^\sharp \circ \mathit{amgu}^\sharp(\mathit{mgu}(\Psi(s),t), \Psi^\sharp(f)) \\
\mathit{exit}^\sharp(s,f,t,g) &= \mathit{proj}^\sharp \circ \mathit{amgu}^\sharp(\mathit{mgu}(\Psi(s),t), \Psi^\sharp(f) \vee g)
\end{aligned}
$$

where $\mathit{amgu}^\sharp(\emptyset, f) = f$ and $\mathit{amgu}^\sharp(\{s = t\} \cup E, f) = \mathit{amgu}^\sharp(E, \mathit{amgu}^\sharp(s,t,f))$.

Proof. We first consider the correctness of $\mathit{amgu}^\sharp$. Let $h = \mathcal{E}(\pi)$. Then $\mathcal{E}(\{\langle x,y\rangle \in \pi \mid x \notin \mathbf{V}(s') \wedge y \notin \mathbf{V}(s')\}) = \mathcal{E}(\pi \cap (\mathcal{V}_P^\ddagger \setminus \mathbf{V}(s'))^2) = h \wedge ((\mathcal{V}_P^\ddagger \setminus \mathbf{V}(s')) \otimes^\sharp (\mathcal{V}_P^\ddagger \setminus \mathbf{V}(s')))$. Similarly, $\mathcal{E}(\{\langle x,y\rangle \in \pi \mid x \notin \mathbf{V}(t') \wedge y \notin \mathbf{V}(t')\}) = h \wedge ((\mathcal{V}_P^\ddagger \setminus \mathbf{V}(t')) \otimes^\sharp (\mathcal{V}_P^\ddagger \setminus \mathbf{V}(t')))$.

$$
\begin{aligned}
&\mathcal{E}(\mathit{amgu}(s',t',\pi)) \\
&= \begin{cases} \mathcal{E}(\{\langle x,y\rangle \in \pi \mid x \notin \mathbf{V}(s') \wedge y \notin \mathbf{V}(s')\}) & \text{if } \mathbf{V}(t') = \emptyset \\ \mathcal{E}(\{\langle x,y\rangle \in \pi \mid x \notin \mathbf{V}(t') \wedge y \notin \mathbf{V}(t')\}) & \text{if } \mathbf{V}(s') = \emptyset \\ \mathcal{E}(\pi \cup \mathit{link}(s',t',\pi) \cup (\chi(t',\pi) \rhd \mathit{link}(s',s',\pi)) \cup (\chi(s',\pi) \rhd \mathit{link}(t',t',\pi))) & \text{otherwise} \end{cases} \\
&= \begin{cases} h \wedge \neg(\bigvee_{x \in \mathbf{V}(s')} \mathcal{L}(x)) \wedge \neg(\bigvee_{y \in \mathbf{V}(s')} \mathcal{R}(y)) \wedge \top^\ddagger & \text{if } \mathbf{V}(t') = \emptyset \\ h \wedge \neg(\bigvee_{x \in \mathbf{V}(t')} \mathcal{L}(x)) \wedge \neg(\bigvee_{y \in \mathbf{V}(t')} \mathcal{R}(y)) \wedge \top^\ddagger & \text{if } \mathbf{V}(s') = \emptyset \\ h \vee \mathit{link}^\sharp(s',t',h) \vee (\chi^\sharp(t',h) \wedge \mathit{link}^\sharp(s',s',h)) \vee (\chi^\sharp(s',h) \wedge \mathit{link}^\sharp(t',t',h)) & \text{otherwise} \end{cases} \\
&= \mathit{amgu}^\sharp(s',t',f)
\end{aligned}
$$

The correctness of $\mathit{entry}^\sharp$ and $\mathit{exit}^\sharp$ follows from that of $\mathit{amgu}^\sharp$, $\Psi^\sharp$ and $\mathit{proj}^\sharp$. ∎