Abstract Matching can improve on Abstract Unification

Andy King
Computing Laboratory,
University of Kent at Canterbury,
Canterbury, CT2 7NF, UK.
a.m.king@ukc.ac.uk

Mark Longley
Dept. of Electronics and Computer Science,
University of Southampton,
Southampton, S09 5NH, UK.
ml@ecs.soton.ac.uk

March 3, 1995

Abstract

Analyses for sharing and freeness are important in the optimisation and the parallelisation of logic programs. By using a standard fixed-point framework, sharing and freeness analysis can be constructed by defining suitable abstract analogs for concrete operations like renaming, restriction, unification and extension. Extension is required in the clause exit mechanisms and is typically formulated in terms of restriction and matching. Matching also arises as goal-head unification in normalised programs in which the (formal) arguments of each clause head are distinct variables. Abstract matching, however, is rarely given special attention and is usually implemented by abstract unification. This paper remedies this, contributing a series of useful, practical and formally-justified abstract matching algorithms for the popular domains $Share$, $Share \times Free$ and $Share \times Free \times Lin$. The matching algorithms are useful and important because they can outperform their corresponding unification algorithms in both precision and speed.

1 Introduction

Analyses for sharing and freeness are important topics of logic programming with applications which include: the sound removal of the occur-check [31]; optimisation of backtracking [5]; the specialisation of unification [33]; and the identification [13, 35] and efficient exploitation [14, 28, 29] of independent and-parallelism [4].

Following the approach of abstract interpretation [9], sharing and freeness analyses are usually constructed by tracing possible program executions with descriptions of the data values (the abstract data) rather than using actual data values (the concrete data). The construction usually divides into domain and framework related issues. For the domain, suitable abstract analogs for concrete operations like renaming, unification, composition and restriction are specified and proven safe for a particular description of substitutions. For example, unification would be mimicked by an abstract unification algorithm in which substitutions are finitely represented by sharing and freeness abstractions, the abstract substitutions. The framework traces the control-flow of Prolog, the concrete semantics, calculating abstract substitutions at various points of a program thereby characterising the actual substitutions which can possibly arise at those program points. Frameworks [1, 11, 16, 17, 23, 26, 29, 30, 34] are usually parameterised by the domain operations and basically solve a set of fixed-point equations.

1.1 Abstract matching in standard frameworks

Although the concrete semantics of logic programs are formulated in terms of unification, both matching and unification usually require to be abstracted in a framework. Frameworks typically trace the values of substitutions which, for finiteness, are restricted to sets of program variables. To describe the sharing and freeness at a certain point in a clause, for example, it is only necessary to characterise the sharing and freeness between the variables of the clause. For finiteness, however, frameworks have to introduce explicit clause entry and exit mechanisms, the latter of which is formulated in terms of restriction and matching. The restriction and matching operations `RESTRG` and `EXTG`, for example, are used in the framework of [23].

`RESTRG(`l`,` $\phi_{in}^{\mathcal{A}}$`)` takes as input a literal, l, of the form $p(u_{i_1}, \ldots, u_{i_m})$ taken from a clause with variables $U = \{u_1, \ldots, u_n\}$; and an abstract substitution on U, $\phi_{in}^{\mathcal{A}}$. `RESTRG` projects $\phi_{in}^{\mathcal{A}}$ onto $\{u_{i_1}, \ldots, u_{i_m}\}$ returning, as output, the projection renamed by $\{u_{i_1} \mapsto u_1, \ldots, u_{i_m} \mapsto u_m\}$. The resulting abstract substitution, $\phi_{entry}^{\mathcal{A}}$ say, is thus expressed in terms of $\{x_1, \ldots, x_m\}$. The result of executing p, $\phi_{exit}^{\mathcal{A}}$ say, is also formulated in terms of $\{x_1, \ldots, x_m\}$

`EXTG(`l`,` $\phi_{in}^{\mathcal{A}}$`,` $\phi_{exit}^{\mathcal{A}}$`)` takes as input, like before, a literal $l = p(u_{i_1}, \ldots, u_{i_m})$ from a clause with variables $U = \{u_1, \ldots, u_n\}$; an abstract substitution on U, $\phi_{in}^{\mathcal{A}}$; and, in addition, an abstract substitution on $\{u_{i_1}, \ldots, u_{i_m}\}$, $\phi_{exit}^{\mathcal{A}}$, the result of executing p with input $\phi_{entry}^{\mathcal{A}} =$ `RESTRG(`l`,` $\phi_{in}^{\mathcal{A}}$`)`. For output, `EXTG` returns the abstract substitution obtained by instantiating (abstractly) $\phi_{in}^{\mathcal{A}}$ to take into account the result $\phi_{exit}^{\mathcal{A}}$ of p.

Operations like `EXTG`, sometimes called extension [29], boil down to renaming and matching. Although implementation details are rarely reported in the literature, extension is usually implemented by an abstract unification algorithm [29]. This is a convenient fix, saving on the design and implementation of an abstract matching algorithm. The saving, however, comes at the cost of both inefficiency and imprecision. Implementation note 48 in appendix B of [23] explains "`EXTG` loses some information because it uses unification instead of matching. To achieve more precision (and we have seen that it makes a difference in precision on a real application), it is necessary to define the matching version of [the abstract unification algorithms] `UNIF1` and `UNIF` as it is known that the second substitution is always an instance of the first one." In addition, since matching is simpler than unification, abstract matching can be faster than abstract unification and therefore should be used wherever possible.

1.2 Abstract matching in normalised frameworks

Both the concrete and abstract semantics of a logic program are simplified if the program is normalised. Normalisation, suggested first in [3], involves transforming the input program into a more restricted syntactic form which preserves the semantics yet simplifies the design and implementation of an analyser. There are various degrees of normalisation, but for brevity, just two will be distinguished. In the first form of normalisation, unification between sub-goals and clause heads is made explicit by making the (formal) arguments of each clause head distinct variables. This enables an (entry and exit) substitution for a clause, both in the concrete and the abstract, to be expressed in terms of its head variables. This is an important simplification, in turn, factoring out some of the predicate entry and exit calculations and streamlining the lub [23]. Interestingly, the normalisation transforms goal-head unification into a combination of goal-head matching and explicit unifications in the body of a clause. Thus, although abstract unification is still required for the unification builtins, abstract matching is more appropriate for clause entry.

In the second, more aggressive form of normalisation, body atoms, other than the equality builtins, are additionally transformed so that the (actual) arguments are distinct variables. This has the chief advantage of simplifying the matching of the sub-goal with clause head to just renaming. Other types of normalisations derive from the form of equation that can be processed by the abstract unification algorithm. For example, the `Pat` algorithm of [23] requires syntactic equations of the form $x_i = x_j$ or $x_i = f(x_{j_1}, \ldots, x_{j_n})$ where f is an n-ary functor; whereas algorithms which apply preunification [6, 18, 19, 20] can manipulate more general equations like $t_i = t_j$ where t_i and t_j are arbitrary terms.

Although the rôle and affect of normalisation has only been partially explored [15, 23], it is clear that normalisation can induce a loss of precision [23] and perhaps more significantly, can increase the number of variables in a clause. Since the size of an abstract substitution is at least polynomial [6, 23] and is often exponential [2, 8, 14, 19, 18, 20, 28, 32] in the number of variables in a clause, normalisation can adversely affect both the time and the memory performance of an analyser [15, 27]. Fast renaming is thus not free. It is not yet clear, however, which has most affect on performance: renaming or the representation of substitutions. The potential speedup from using abstract matching techniques to implement goal-head unification (in the case of first degree normalisation) makes the performance of an analyser even harder to predict. To summarise, abstract matching algorithms need to be synthesised to quantitatively compare the speed and accuracy of the various normalisation strategies. Only then can the implementer make informed design decisions.

1.3 Abstract matching in sharing and freeness analysis

Sharing (or aliasing) analysis conventionally infers which program variables are definitely grounded and which variables can possibly be bound to terms containing a common variable. Freeness analysis usually infers which program variables are free, that is, which variables can never be bound to non-variable terms. Early proposals for sharing and freeness analyses include [36, 12, 25] and [28].

This paper contributes a series of useful, practical and formally-justified abstract matching algorithms for a number of popular sharing and freeness domains that can be used as part of extension operator or applied to abstract goal-head matching. As far as is known, the rôle of abstract matching has not been studied before very carefully, and certainly no abstract matching algorithms have been reported for the popular sharing and freeness domains.

The exposition is structured as follows. Section 2 describes the notation and preliminary definitions which will be used throughout. To fit with the trend of constructing a composite domain from proven and well-tried domain units [2, 7, 18, 22], abstract matching algorithms will be presented for the domains $Share$ [14], $Share \times Free$ [28] and a variant of the [2] domain, $Share \times Free \times Lin$ [32]. In section 3, the focus is on the domains. $Share$ captures possible sharing and definite groundness, whereas the $Free$ and Lin components denote sets of free variables and variables bound to linear terms. Linearity relates to the number of times a variable occurs in a term [6, 19, 31]. A term is linear if it definitely does not contain multiple occurrences of a variable; otherwise it is non-linear. The significance of linearity is that the unification of linear terms only yields restricted forms of aliasing. Specifically, by tracking linearity, a sharing analysis does not always have to assume that aliasing is transitive [6]. The structure of $Share$ is particularly rich, implicitly encoding covering information [8]. Covering, in short, permits groundness to interact nicely with sharing to remove redundant aliasing. For finiteness, $Share$, $Free$ and Lin are parametrised by a finite set of program variables, $Pvar$, which typically equate to the variables of a clause. To be precise, $Share_{Pvar} = \wp(\wp(Pvar))$ and $Free_{Pvar} = Lin_{Pvar} = \wp(Pvar)$.

In section 4, the emphasis changes to abstracting matching for these domains. An abstract matching algorithm is described for $Share_{Pvar} \times Free_{Pvar} \times Lin_{Pvar}$. Abstract matching algorithms $Share_{Pvar} \times Free_{Pvar}$ and $Share_{Pvar}$ follow straightforwardly from the $Share_{Pvar} \times Free_{Pvar} \times Lin_{Pvar}$ algorithm. The matching algorithms can outperform their corresponding unification algorithms [2, 14, 28, 32] in both precision and speed. Abstract matching is potentially faster than the abstract unification because certain forms of aliasing that arise in unification cannot occur in matching. Aliasing is tracked by calculating a union closure $\star$ operation [14, 28]. Closure has an exponential time and space complexity in the size of the abstract substitution and therefore closure calculations should be avoided, wherever possible. Since abstract matching computes fewer closures than abstract unification it is potentially faster. Moreover, since excess closures also reduce precision, the matching algorithms can also improve accurate. In addition, matching need make less conservative assumptions about the freeness and linearity of variables which, in turn, can further improve the quality of the aliasing information.

Although correctness is proved, for reasons of brevity and continuity, the proofs are relegated to an appendix, section 9. In addition, to shorten the presentation and avoid repetition, the abstract matching algorithms for $Share_{Pvar} \times Free_{Pvar}$ and $Share_{Pvar}$ are respectively presented in appendices 7 and 8. Sections 5 and 6 present the related work and the concluding discussion.

2 Notation and preliminaries

To introduce the analysis some notation and preliminary definitions are required. The reader is assumed to be familiar with the standard constructs used in logic programming [24] such as a universe of variables $(u, v \in) Uvar$; the set of terms $(t \in) Term$ formed from $Uvar$ and the set of functors $(f, g, h \in) Func$ (of the first-order language underlying the program), and the set of program atoms $Atom$. $Func$ is considered to include the set of constants $Const$. It is sometimes convenient to abbreviate $f(t_1, \ldots, t_n)$ to $f(\underline{t_i})$. Let $Pvar$ denote a finite set of program variables – the variables that are in the text of the program; and let $var(o)$ denote the set of variables in a syntactic object o.

2.1 Substitutions

A substitution ϕ is a total mapping $\phi : Uvar \rightarrow Term$ such that its domain $dom(\phi) = \{u \in Uvar \mid \phi(u) \neq u\}$ is finite. The application of a substitution ϕ to a variable u is denoted by $\phi(u)$. Thus the codomain is given by $cod(\phi) = \cup_{u \in dom(\phi)} var(\phi(u))$. A substitution ϕ is sometimes represented as a finite set of variable and term pairs $\{u \mapsto \phi(u) \mid u \in dom(\phi)\}$. The identity mapping on $Uvar$ is called the empty substitution and is denoted by ϵ. Substitutions, sets of substitutions, and the set of substitutions on $Uvar$ are denoted by lower-case Greek letters, upper-case Greek letters, and Sub.

Substitutions are extended in the usual way from variables to functions, from functions to terms, and from terms to atoms. The restriction of a substitution ϕ to a set of variables $U \subseteq Uvar$ and the composition of two substitutions ϕ and φ, are denoted by $\phi \upharpoonright U$ and $\phi \circ \varphi$ respectively, and defined so that $(\phi \circ \varphi)(u) = \phi(\varphi(u))$. Restriction lifts to sets of substitutions by: $\Phi \upharpoonright U = \{\phi \upharpoonright U \mid \phi \in \Phi\}$. The preorder Sub $(\sqsubseteq)$, ϕ is more general than φ, is defined by: $\phi \sqsubseteq \varphi$ if and only if there exists a substitution $\psi \in Sub$ such that $\varphi = \psi \circ \phi$. The preorder induces an equivalence relation $\approx$ on Sub, that is: $\phi \approx \varphi$ if and only if $\phi \sqsubseteq \varphi$ and $\varphi \sqsubseteq \phi$.

2.2 Equations and most general unifiers

An equation is an equality constraint of the form $a = b$ where a and b are terms or atoms. Let $(e \in) Eqn$ denote the set of finite sets of equations. The equation set $\{e\} \cup E$, following [6], is abbreviated by $e : E$. There is a natural mapping from substitutions to equations, that is, $eqn(\phi) = \{u = t \mid u \mapsto t \in \phi\}$. Thus, when unambiguous, substitutions will be expressed as equations. The set of most general unifiers of E, $mgu(E)$, is defined operationally [14] in terms of a predicate mgu. The predicate $mgu(E, \phi)$ which is true if ϕ is a most general unifier of E.

Definition 2.1 (mgu) *The set of most general unifiers $mgu(E) \in \wp(Sub)$ is defined by: $mgu(E) = \{\phi \mid mgu(E, \phi)\}$ where*

$$\begin{array}{r}
mgu(\emptyset, \epsilon) \\
mgu(v = v : E, \zeta) \ \mathit{if}\, mgu(E, \zeta) \\
mgu(t = v : E, \zeta) \ \mathit{if}\, mgu(v = t : E, \zeta) \\
mgu(v = t : E, \zeta \circ \eta) \ \mathit{if}\, mgu(\eta(E), \zeta) \wedge v \notin var(t) \wedge \eta = \{v \mapsto t\} \\
mgu(f(\underline{t_i}) = f(\underline{t_i'}) : E, \zeta) \ \mathit{if}\, mgu(\{t_i = t_i'\}_{i=1}^{n} \cup E, \zeta)
\end{array}$$

By induction it follows that $dom(\phi) \cap cod(\phi) = \emptyset$ if $\phi \in mgu(E)$, or put another way, that the most general unifiers are idempotent [21].

Following [14], the semantics of a logic program is formulated in terms of a single $unify$ operator. To construct $unify$, and specifically to rename apart program variables, an invertible substitution [21], Υ, is introduced. It is convenient to let $Rvar$ denote a universe of renaming variables distinct from $Uvar$, $Uvar \cap Rvar = \emptyset$, and suppose that $\Upsilon : Uvar \rightarrow Rvar$.

Definition 2.2 (*unify*) *The partial mappings* $unify : Atom \times Sub \times Atom \times Sub \rightarrow Sub$ *is defined by:*

$$unify(a, \phi, b, \psi) = (\varphi \circ \phi) \restriction Pvar \text{ where } \varphi \in mgu(\{\phi(a) = \Upsilon(\psi(b))\})$$

2.3 Linearity

To be more precise about linearity, it is necessary to introduce the variable multiplicity of a term t, denoted $\chi(t)$.

Definition 2.3 (variable multiplicity, χ [6]) *The variable multiplicity operator* $\chi : Term \rightarrow \{0, 1, 2\}$ *is defined by:*

$$\chi(t) = \max(\{\chi_u(t) \mid u \in Uvar\}) \text{ where } \chi_u(t) = \begin{cases} 0 \text{ if } u \text{ does not occur in } t \\ 1 \text{ if } u \text{ occurs only once in } t \\ 2 \text{ if } u \text{ occurs many times in } t \end{cases}$$

If $\chi(t) = 0$, t is ground; if $\chi(t) = 1$, t is linear; and if $\chi(t) = 2$, t is non-linear. Note that if $\phi(u) \in Uvar$ then $\chi(\phi(u)) = 1$ so that free variables, like u, are linear. The unification of linear terms only yields restricted forms of aliasing. Lemma 2.1 states one restriction on a most general unifier which follows from unification with a linear term.

Lemma 2.1 $\chi(b) \neq 2 \wedge var(a) \cap var(b) = \emptyset \wedge \phi \in mgu(\{a = b\}) \Rightarrow$

1. $\forall u, u' \in Uvar \,.\, u \neq u' \wedge var(\phi(u)) \cap var(\phi(u')) \neq \emptyset \Rightarrow u \notin var(a) \vee u' \notin var(a)$.

Lemma 2.1 represents one case of a three part result which is formally established in [19]. The lemma differs from the corresponding lemma in [6] (lemma 2.2) because lemma 2.1 requires that a and b do not share variables. This is essentially a work-around for a subtle mistake in lemma 2.2 [10].

3 Abstracting substitutions

Abstract interpretation clarifies how data is represented in the abstract by requiring the relationship between the data and the abstract data to be made explicit. To keep the paper self-contained the $Share_{Pvar} \times Free_{Pvar} \times Lin_{Pvar}$ domain and its abstraction and concretisation mappings will be briefly reviewed. The mappings for $Share_{Pvar} \times Free_{Pvar}$ are presented in appendix 7.

3.1 On the domain $Share_{Pvar} \times Free_{Pvar} \times Lin_{Pvar}$

$Share_{Pvar}$ is formulated in terms of sharing groups [14, 29] which record which program variables potentially share variables. A sharing group is a (possibly empty) set of program variables. $Free_{Pvar}$ and Lin_{Pvar}, on the other hand, represents the free and linear program variables as sets.

Definition 3.1 (*$Share_{Pvar}$*, *$Free_{Pvar}$* **and** *Lin_{Pvar}*) *The domains $Share_{Pvar}$, $Free_{Pvar}$ and Lin_{Pvar} are defined by:*

$$Share_{Pvar} = \wp(\wp(Pvar)), \quad Free_{Pvar} = \wp(Pvar), \quad Lin_{Pvar} = \wp(Pvar)$$

The intuition is that a sharing group records which program variables are bound to terms that share a variable. $Share_{Pvar} \times Free_{Pvar} \times Lin_{Pvar}$ is finite since $Pvar$ is finite.

3.2 On the abstraction and concretisation mappings $\alpha^{\mathcal{SFL}}_{Pvar}$ and $\gamma^{\mathcal{SFL}}_{Pvar}$

In the spirit of [29], the abstraction and concretisation mappings are constructed by lifting three mappings, sh_{Pvar}, fr_{Pvar} and ln_{Pvar}, to sets of substitutions. The mappings sh_{Pvar}, fr_{Pvar} and ln_{Pvar} detail how a single substitution is abstracted.

Definition 3.2 (sh_{Pvar}**,** fr_{Pvar} **and** ln_{Pvar}) *The abstraction mappings* $sh_{Pvar} : Sub \rightarrow Share_{Pvar}$, $fr_{Pvar} : Sub \rightarrow Free_{Pvar}$ *and* $ln_{Pvar} : Sub \rightarrow Lin_{Pvar}$ *are defined by:*

$$sh_{Pvar}(\phi) = \{occ_{Pvar}(u, \phi) \mid u \in Uvar\}, \quad occ_{Pvar}(u, \phi) = \{v \in Pvar \mid u \in var(\phi(v))\}$$

$$fr_{Pvar}(\phi) = \{v \in Pvar \mid var(\phi(v)) \in Uvar\}$$

$$ln_{Pvar}(\phi) = \{v \in Pvar \mid \chi(\phi(v)) \leq 1\}$$

The abstraction sh_{Pvar} is analogous to the abstraction $\mathcal{A}$ used in [29]. Observe that for $\phi \approx \varphi$, $sh_{Pvar}(\phi) = sh_{Pvar}(\varphi)$, $fr_{Pvar}(\phi) = fr_{Pvar}(\varphi)$ and $ln_{Pvar}(\phi) = ln_{Pvar}(\varphi)$. The mapping $\alpha^{\mathcal{SFL}}_{Pvar}$ and $\gamma^{\mathcal{SFL}}_{Pvar}$ follow directly from sh_{Pvar}, fr_{Pvar} and ln_{Pvar}.

Definition 3.3 ($\alpha^{\mathcal{SFL}}_{Pvar}$ **and** $\gamma^{\mathcal{SFL}}_{Pvar}$) *The abstraction and concretisation mappings* $\alpha^{\mathcal{SFL}}_{Pvar} : \wp(Sub) \rightarrow Share_{Pvar} \times Free_{P}$ *and* $\gamma^{\mathcal{SFL}}_{Pvar} : Share_{Pvar} \times Free_{Pvar} \times Lin_{Pvar} \rightarrow \wp(Sub)$ *are defined by:*

$$\alpha^{\mathcal{SFL}}_{Pvar}(\Phi) = \langle \alpha^{\mathcal{S}}_{Pvar}(\Phi), \alpha^{\mathcal{F}}_{Pvar}(\Phi), \alpha^{\mathcal{L}}_{Pvar}(\Phi)\rangle, \quad \gamma^{\mathcal{SFL}}_{Pvar}(\phi^{\mathcal{SFL}}) = \gamma^{\mathcal{S}}_{Pvar}(\phi^{\mathcal{S}}) \cap \gamma^{\mathcal{F}}_{Pvar}(\phi^{\mathcal{F}}) \cap \gamma^{\mathcal{L}}_{Pvar}(\phi^{\mathcal{L}})$$

where

$$\begin{array}{ll} \alpha^{\mathcal{S}}_{Pvar}(\Phi) = \cup_{\phi \in \Phi} sh_{Pvar}(\phi), & \gamma^{\mathcal{S}}_{Pvar}(\phi^{\mathcal{S}}) = \{\phi \mid sh_{Pvar}(\phi) \subseteq \phi^{\mathcal{S}}\} \\ \alpha^{\mathcal{F}}_{Pvar}(\Phi) = \cap_{\phi \in \Phi} fr_{Pvar}(\phi), & \gamma^{\mathcal{F}}_{Pvar}(\phi^{\mathcal{F}}) = \{\phi \mid \phi^{\mathcal{F}} \subseteq fr_{Pvar}(\phi)\} \\ \alpha^{\mathcal{L}}_{Pvar}(\Phi) = \cap_{\phi \in \Phi} ln_{Pvar}(\phi), & \gamma^{\mathcal{L}}_{Pvar}(\phi^{\mathcal{L}}) = \{\phi \mid \phi^{\mathcal{L}} \subseteq ln_{Pvar}(\phi)\} \end{array}$$

Note that $\alpha^{\mathcal{S}}(\emptyset) = \emptyset$ whereas $\alpha^{\mathcal{S}}(\Phi) = \{\emptyset\}$ if Φ is a set of substitutions which all ground $Pvar$. This distinction is preserved in $Share_{Pvar} \times Free_{Pvar} \times Lin_{Pvar}$.

4 Abstracting matching

Abstract interpretation can help to focus the development of an analysis by illuminating the connection between an operation (like matching) and its abstract counterpart.

4.1 On the abstract matching relation $mgu^{\mathcal{SFL}}$

Matching is abstracted by tracing the steps of a standard unification algorithm [21]. To trace unification, the abstract algorithm mimics the recursive simplification steps of mgu in a relation $mgu^{\mathcal{SFL}}$, relegating the solution of simplified equations of the form $u = t$ or $t = u$ to a mapping $mgu^{\mathcal{SFL}}$. Unification, or more precisely pre-unification [6], cannot be used to implement the simplification steps. Instead a simplification algorithm like that of [28] is used.

The relation $mgu^{\mathcal{SFL}}$ is defined to abstract a slight variant of mgu. Specifically, if $\varphi \in mgu(\{\phi(t) = \phi(t')\})$, $\varphi(\phi(t)) = \phi(t')$ and $\phi \in \gamma^{\mathcal{SFL}}_{Pvar}(\phi^{\mathcal{SFL}})$ then $mgu^{\mathcal{SFL}}(t, t', \phi^{\mathcal{SFL}})$ abstracts the composition $\varphi \circ \phi$ (rather than φ), that is, $\varphi \circ \phi \in \gamma^{\mathcal{SFL}}_{Pvar}(mgu^{\mathcal{SFL}}(t, t', \phi^{\mathcal{SFL}}))$. This spares the need to define an extra (composition) operator.

Definition 4.1 ($mgu^{\mathcal{SFL}}$) *The relation* $mgu^{\mathcal{SFL}} : Eqn \times (Share_{Pvar} \times Free_{Pvar} \times Lin_{Pvar}) \times (Share_{Pvar} \times Free_{Pvar} \times Lin_{Pvar})$ *is defined by:*

$$\begin{array}{rl} mgu^{\mathcal{SFL}}(\emptyset, \phi^{\mathcal{SFL}}, \phi^{\mathcal{SFL}}) & \\ mgu^{\mathcal{SFL}}(u = u : E, \phi^{\mathcal{SFL}}, \varphi^{\mathcal{SFL}}) & \mathit{if}\; mgu^{\mathcal{SFL}}(E, \phi^{\mathcal{SFL}}, \varphi^{\mathcal{SFL}}) \\ mgu^{\mathcal{SFL}}(t = u : E, \phi^{\mathcal{SFL}}, \varphi^{\mathcal{SFL}}) & \mathit{if}\; mgu^{\mathcal{SFL}}(E, mgu^{\mathcal{SFL}}(t, u, \phi^{\mathcal{SFL}}), \varphi^{\mathcal{SFL}}) \wedge u \notin var(t) \\ mgu^{\mathcal{SFL}}(u = t : E, \phi^{\mathcal{SFL}}, \varphi^{\mathcal{SFL}}) & \mathit{if}\; mgu^{\mathcal{SFL}}(E, mgu^{\mathcal{SFL}}(u, t, \phi^{\mathcal{SFL}}), \varphi^{\mathcal{SFL}}) \wedge u \notin var(t) \\ mgu^{\mathcal{SFL}}(f(\underline{t_i}) = f(\underline{t'_i}) : E, \phi^{\mathcal{SFL}}, \varphi^{\mathcal{SFL}}) & \mathit{if}\; mgu^{\mathcal{SFL}}(\{t_i = t'_i\}_{i=1}^{n} \cup E, \phi^{\mathcal{SFL}}, \varphi^{\mathcal{SFL}}) \end{array}$$

4.2 On the auxiliary operations

To define the mapping $mgu^{\mathcal{SFL}}$ (and thus the relation $mgu^{\mathcal{SFL}}$) a number of standard auxiliary operators are required [14, 29]. First, $rel(t, \phi^{\mathcal{S}})$ represents the sharing groups of $\phi^{\mathcal{S}}$ which are relevant to the term t, that is, those sharing groups of $\phi^{\mathcal{SFL}}$ which share variables with t. Second, in the absence of useful freeness and linearity information worst-case aliasing is assumed. Thus, as in [14, 29], a closure under union operator, *, is employed to enumerate all the possible sharing groups that can possibly arise in unification. Third, to succinctly define $mgu^{\mathcal{SFL}}$, it is convenient to lift $\cup$ to sets of sharing groups with a pair-wise union operator, denoted $\Box$.

Definition 4.2 (*rel*, * **[14, 29] and** $\Box$)

$$rel(t, \phi^{\mathcal{S}}) = \{U \in \phi^{\mathcal{S}} \mid U \cap var(t) \neq \emptyset\}$$

$$\phi^{\mathcal{S}\star} = \phi^{\mathcal{S}} \cup \{U \cup U' \mid U, U' \in \phi^{\mathcal{S}\star}\}, \quad \phi^{\mathcal{S}} \,\Box\, \phi'^{\mathcal{S}} = \{U \cup U' \mid U \in \phi^{\mathcal{S}} \wedge U' \in \phi'^{\mathcal{S}}\}$$

The mappings $share^{\mathcal{SFL}}$, $free^{\mathcal{SFL}}$ and $lin^{\mathcal{SFL}}$ apply different analysis strategies according to the freeness and linearity of $\phi(t)$ and $\phi(t')$ for $\phi \in \gamma^{\mathcal{SFL}}_{Pvar}(\phi^{\mathcal{SFL}})$.

The Lin_{Pvar} component of the domain encodes the variable multiplicity of a substitution. More significantly, if $\phi \in \gamma^{\mathcal{SFL}}_{Pvar}(\phi^{\mathcal{SFL}})$ then the variable multiplicity of $\phi(t)$ can be (partially) deduced from t and $\phi^{\mathcal{SFL}}$. The precise relationship between $\phi(t)$, t and $\phi^{\mathcal{SFL}}$ is formalised in definition 4.3 and lemma 4.1. Proof of lemma 4.1 is given in proof 9.1.

Definition 4.3 ($\chi^{\mathcal{SFL}}$) *The abstract variable multiplicity operator* $\chi^{\mathcal{SFL}} : Term \times (Share_{Pvar} \times Free_{Pvar} \times Lin_{Par}) \to \{0, 1, 2\}$ *is defined by:*

$$\chi^{\mathcal{SFL}}(t, \phi^{\mathcal{SFL}}) = \begin{cases} 0 \; if \, var(t) \cap var(\phi^{\mathcal{S}}) = \emptyset \\ 1 \; else \; if \; \forall u \in var(\phi^{\mathcal{S}}).\chi_u(t) \leq 1 \; \wedge \\ \qquad var(t) \subseteq \phi^{\mathcal{L}} \qquad\qquad \wedge \\ \qquad \forall u, v \in var(t).rel(u, \phi^{\mathcal{S}}) \cap rel(v, \phi^{\mathcal{S}}) = \emptyset \\ 2 \; otherwise \end{cases}$$

Lemma 4.1

$$var(t) \subseteq Pvar \wedge \phi \in \gamma^{\mathcal{SFL}}_{Pvar}(\phi^{\mathcal{SFL}}) \Rightarrow \chi(\phi(t)) \leq \chi^{\mathcal{SFL}}(t, \phi^{\mathcal{SFL}})$$

4.3 On the abstract matching mapping $mgu^{\mathcal{SFL}}$

The mapping $mgu^{\mathcal{SFL}}(t, t', \phi^{\mathcal{SFL}})$ abstracts the matching of two terms, $\phi(t)$ and $\phi(t')$, where $\phi \in \phi^{\mathcal{SFL}}$. The mapping assumes that $\phi(t)$ is more general than $\phi(t')$. The different cases of $mgu^{\mathcal{SFL}}$ apply different analysis strategies according to whether $\phi(t)$ is free or $\phi(t')$ is free or linear. Simplification ensures that the equation $t = t'$ assumes the form of either $u = t'$ or $t = u$.

Definition 4.4 ($mgu^{\mathcal{SFL}}$)

$$mgu^{\mathcal{SFL}}(t, t', \phi^{\mathcal{SFL}}) = \mu^{\mathcal{SFL}} \; where$$

$$\begin{aligned} \mu^{\mathcal{S}} = \; & \phi^{\mathcal{S}} \setminus (rel(t, \phi^{\mathcal{S}}) \cup rel(t', \phi^{\mathcal{S}})) \cup \\ & \begin{cases} rel(t, \phi^{\mathcal{S}}) \;\; \Box \, rel(t', \phi^{\mathcal{S}}) if \, t \in \phi^{\mathcal{F}} \vee \chi^{\mathcal{SFL}}(t', \phi^{\mathcal{SFL}}) \leq 1 \\ rel(t, \phi^{\mathcal{S}})^\star \, \Box \, rel(t', \phi^{\mathcal{S}}) otherwise \end{cases} \end{aligned}$$

$$\mu^{\mathcal{F}} = \begin{cases} \phi^{\mathcal{F}} \cup \{t\} & if \, t' \in \phi^{\mathcal{F}} \\ \phi^{\mathcal{F}} \setminus var(rel(t, \phi^{\mathcal{S}})) & otherwise \end{cases}$$

$$\mu^{\mathcal{L}} = \begin{cases} \phi^{\mathcal{L}} \cup var(t) & if \, \chi^{\mathcal{SFL}}(t', \phi^{\mathcal{SFL}}) \leq 1 \\ \phi^{\mathcal{L}} \setminus var(rel(t, \phi^{\mathcal{S}})) & otherwise \end{cases}$$

Note that $rel(t, \phi^{\mathcal{S}}) \Box rel(t', \phi^{\mathcal{S}}) = \emptyset$ and $rel(t, \phi^{\mathcal{S}})^{\star} \Box rel(t', \phi^{\mathcal{S}}) = \emptyset$ if $rel(t, \phi^{\mathcal{S}}) = \emptyset$. Thus, in the first case of $\mu^{\mathcal{S}}$, $rel(t, \phi^{\mathcal{S}})$ need not be calculated if $rel(u, \phi^{\mathcal{S}}) = \emptyset$ and similarly in case two, $rel(t, \phi^{\mathcal{S}})$ need not be computed or closed under union if $rel(u, \phi^{\mathcal{S}}) = \emptyset$. Analogous refinements follow if $rel(t, \phi^{\mathcal{S}}) = \emptyset$.

Observe that $mgu^{\mathcal{SFL}}$ improves on a refinement suggested in [28]. In abstract unification, the calculation of a closure can be avoided if either t or t' are free. If neither t nor t' are free, two closure calculations are required. Abstract matching, however, requires at most one closure computation. This follows from the restricted forms of aliasing that can arise from matching. Moreover, if t or t' are free, or $\phi(t')$ is linear, no closures need be calculated.

The correctness of the mapping $mgu^{\mathcal{SFL}}$ is stated as lemma 4.2. The corresponding proof is numbered 9.2.

Lemma 4.2

$$\begin{array}{l}
\phi \in \gamma^{\mathcal{SFL}}_{Pvar}(\phi^{\mathcal{SFL}}) \quad \wedge \\
var(\phi(t)) \cap var(\phi(t')) = \emptyset \wedge \\
var(t) \cup var(t') \subseteq Pvar \quad \wedge \\
\varphi \in mgu(\{\phi(t) = \phi(t')\}) \quad \wedge \\
\varphi(\phi(t)) = \phi(t') \quad \wedge \\
\qquad mgu^{\mathcal{SFL}}(t, t', \phi^{\mathcal{SFL}}) = \mu^{\mathcal{SFL}} \Rightarrow \varphi \circ \phi \in \gamma^{\mathcal{SFL}}_{Pvar}(\mu^{\mathcal{SFL}})
\end{array}$$

The correctness of the relation $mgu^{\mathcal{SFL}}$ follows from lemma 4.2 and is stated as theorem 4.1. The corresponding proof is numbered 9.3.

Theorem 4.1

$$\begin{array}{l}
\phi \in \gamma^{\mathcal{SFL}}_{Pvar}(\phi^{\mathcal{SFL}}) \quad \wedge \\
E = \{t_i = t_i'\}_{i=1}^{n} \quad \wedge \\
var(\phi(t_i)) \cap var(\phi(t_j)) = \emptyset \wedge \\
var(E) \subseteq Pvar \quad \wedge \\
\varphi \in mgu(\phi(E)) \quad \wedge \\
\varphi(\phi(t_i)) = \phi(t_i') \quad \wedge \\
\qquad mgu^{\mathcal{SFL}}(E, \phi^{\mathcal{SFL}}, \mu^{\mathcal{SFL}}) \Rightarrow \varphi \circ \phi \in \gamma^{\mathcal{SFL}}_{Pvar}(\mu^{\mathcal{SFL}})
\end{array}$$

It is convenient shorthand to regard $mgu^{\mathcal{SFL}}$ as a mapping, that is, $mgu^{\mathcal{SFL}}(E, \phi^{\mathcal{SFL}}) = \psi^{\mathcal{SFL}}$ if $mgu^{\mathcal{SFL}}(E, \phi^{\mathcal{SFL}}, \psi^{\mathcal{SFL}})$. Strictly, it is necessary to show that $mgu^{\mathcal{SFL}}(E, \phi^{\mathcal{SFL}}, \psi^{\mathcal{SFL}})$ is deterministic for $mgu^{\mathcal{SFL}}(E, \phi^{\mathcal{SFL}})$ to be well-defined. Like in [6], the conjecture is that $mgu^{\mathcal{SFL}}$ yields a unique abstract substitution $\psi^{\mathcal{SFL}}$ for $\phi^{\mathcal{SFL}}$ regardless of the order in which E is solved (though, in practice, any $\psi^{\mathcal{SFL}}$ is safe).

4.4 On the mappings $entry^{\mathcal{SFL}}$ and $exit^{\mathcal{SFL}}$

To finally define the matching versions of clause entry and exit, abstract restriction has to be introduced. An abstract substitution, $\phi^{\mathcal{SFL}}$ say, is implicitly defined in terms of a set of program variables $Pvar$. If $\phi^{\mathcal{SFL}'} = \phi^{\mathcal{SFL}} \restriction^{\mathcal{SFL}} Pvar'$ then $\phi^{\mathcal{SFL}'}$, the restricted abstract substitution, is defined in terms of the variables $Pvar \cap Pvar'$. Abstract restriction thus restricts the variable set of an abstract substitution and does not abstract concrete restriction. The precise relationship between $\phi^{\mathcal{SFL}}$ and $\phi^{\mathcal{SFL}'}$ is stated as lemma 4.3 and established in proof 9.4.

Definition 4.5 (abstract restriction) *The abstract restriction operator, $\cdot \restriction^{\mathcal{SFL}} \cdot$, is defined by:*

$$\phi^{\mathcal{SFL}} \restriction^{\mathcal{SFL}} U = \langle \phi^{\mathcal{S}} \restriction^{\mathcal{S}} U, \phi^{\mathcal{F}} \restriction^{\mathcal{F}} U, \phi^{\mathcal{L}} \restriction^{\mathcal{L}} U \rangle \; where \; \begin{array}{l} \phi^{\mathcal{S}} \restriction^{\mathcal{S}} U = \{U \cap U' \mid U' \in \phi^{\mathcal{S}}\} \\ \phi^{\mathcal{F}} \restriction^{\mathcal{F}} U = U \cap \phi^{\mathcal{F}} \\ \phi^{\mathcal{L}} \restriction^{\mathcal{L}} U = U \cap \phi^{\mathcal{L}} \end{array}$$

Lemma 4.3

$$\gamma_{Pvar}^{\mathcal{SFL}}(\phi^{\mathcal{SFL}}) \subseteq \gamma_{Pvar \cap Pvar'}^{\mathcal{SFL}}(\phi^{\mathcal{SFL}} \restriction^{\mathcal{SFL}} Pvar')$$

The definitions of $entry^{\mathcal{SFL}}$ and $exit^{\mathcal{SFL}}$ are given below with their safety stated as theorems 4.2 and 4.3. Clause entry abstracts the unification of a (renamed) goal atom $\Upsilon(\phi_{in}(a_{call}))$ and a head atom a_{head} where $\phi_{in} \in \gamma_{Pvar}^{\mathcal{SFL}}(\phi_{in}^{\mathcal{SFL}})$. The resulting abstract substitution is restricted to the variables of the clause to obtain the clause entry substitution $\phi_{enter}^{\mathcal{SFL}}$. To clarify, ϕ_{in} and ϕ_{out} represent the input and output pairs for goal, or equivalently a program literal; whereas ϕ_{enter} and ϕ_{exit} represent the entry and exit substitutions for a clause which invoked by the literal. Matching arises if the arguments of a_{head} are distinct variables.

Clause exit abstracts the unification of a goal atom $\phi_{in}(a_{call})$ and a (renamed) head atom $\Upsilon(\phi_{exit}(a_{head}))$ where $\phi_{in} \in \gamma_{Pvar}^{\mathcal{SFL}}(\phi_{in}^{\mathcal{SFL}})$ and $\phi_{exit} \in \gamma_{Pvar}^{\mathcal{SFL}}(\phi_{exit}^{\mathcal{SFL}})$. Matching arises because $\phi_{in}(a_{call})$ is more general than $\Upsilon(\phi_{exit}(a_{head}))$ since ϕ_{enter} is more general than ϕ_{exit}.

Definition 4.6 ($entry^{\mathcal{SFL}}$ and $exit^{\mathcal{SFL}}$) *The $entry^{\mathcal{SFL}}$ and $exit^{\mathcal{SFL}}$ mappings are defined by:*

$$entry^{\mathcal{SFL}}(a_{call}, \phi_{in}^{\mathcal{SFL}}, a_{head}) = \phi_{enter}^{\mathcal{SFL}}, \quad exit^{\mathcal{SFL}}(a_{call}, \phi_{in}^{\mathcal{SFL}}, a_{head}, \phi_{exit}^{\mathcal{SFL}}) = \phi_{out}^{\mathcal{SFL}}$$

$$\phi_{enter}^{\mathcal{SFL}} = mgu^{\mathcal{SFL}}(\{\Upsilon(a_{call}) = a_{head}\}, \Upsilon(\phi_{in}^{\mathcal{SFL}}) \cup \epsilon^{\mathcal{SFL}}) \restriction^{\mathcal{SFL}} Pvar$$
$$\phi_{out}^{\mathcal{SFL}} = mgu^{\mathcal{SFL}}(\{a_{call} = \Upsilon(a_{head})\}, \phi_{in}^{\mathcal{SFL}} \cup \Upsilon(\phi_{exit}^{\mathcal{SFL}})) \restriction^{\mathcal{SFL}} Pvar$$

Theorems 4.2 and 4.3 assume $var(a_{call}) \cup var(a_{head}) \subseteq Pvar$ and are established by proofs 9.5 and 9.6.

Theorem 4.2 (local safety of $entry^{\mathcal{SFL}}$)

$$\begin{array}{ll}
\phi_{in} \in \gamma_{Pvar}^{\mathcal{SFL}}(\phi_{in}^{\mathcal{SFL}}) & \wedge \\
\varphi \in mgu(\{\Upsilon(\phi_{in}(a_{call})) = a_{head}\}) & \wedge \\
\Upsilon(\phi_{in}(a_{call})) = \varphi(a_{head}) & \Rightarrow \\
\quad unify(a_{head}, \epsilon, a_{call}, \phi_{in}) \in \gamma_{Pvar}^{\mathcal{SFL}}(entry^{\mathcal{SFL}}(a_{call}, \phi_{enter}^{\mathcal{SFL}}, a_{head})) &
\end{array}$$

Theorem 4.3 (local safety of $exit^{\mathcal{SFL}}$)

$$\begin{array}{ll}
\phi_{in} \in \gamma_{Pvar}^{\mathcal{SFL}}(\phi_{in}^{\mathcal{SFL}}) \wedge \phi_{exit} \in \gamma_{Pvar}^{\mathcal{SFL}}(\phi_{exit}^{\mathcal{SFL}}) & \wedge \\
\varphi \in mgu(\{\phi_{in}(a_{call}) = \Upsilon(\phi_{exit}(a_{head}))\}) & \wedge \\
\varphi(\phi_{in}(a_{call})) = \Upsilon(\varphi(a_{head})) & \Rightarrow \\
\quad unify(a_{call}, \phi_{in}, a_{head}, \phi_{exit}) \in \gamma_{Pvar}^{\mathcal{SFL}}(exit^{\mathcal{SFL}}(a_{call}, \phi_{in}^{\mathcal{SFL}}, a_{head}, \phi_{exit}^{\mathcal{SFL}})) &
\end{array}$$

5 Related and future work

Abstract unification algorithms for sharing and freeness have been studied in some detail [2, 6, 7, 8, 12, 14, 18, 19, 22, 25, 28, 32, 36] but, curiously, there is a dearth of work on abstract matching. Abstract matching, in fact, is rarely given special attention and is usually implemented by abstract unification.

Future work will focus on implementation and benchmarking (which is a non-trivial study within itself) to measure the speedup from substituting matching for unification. The rôle and affect of normalisation will also be explored particularly in regard to goal-head unification. Another direction for future work is in extending the abstract matching algorithms to trace sure structural information [2, 18, 23]. This would avoid any loss of precision that might be introduced through normalisation.

6 Conclusions

Most of the execution time of an analyser is typically spent, not in the framework, but on domain operations like unification and matching [15]. Thus, if the performance of sharing and freeness analysis is to be improved, it is crucial that operations like abstract matching are both precise and efficient. Improving the efficiency of abstract matching speeds up extension, clause exit, for arbitrary programs; and goal-head unification clause entry, for normalised programs with head arguments that are distinct variables.

A series of useful, practical and formally-justified abstract matching algorithms have been synthesised for the popular domains $Share$, $Share \times Free$ and $Share \times Free \times Lin$. The matching algorithms can outperform their corresponding unification algorithms in both precision and speed. The techniques are significant because they can under-pin a number of important optimisation and parallelisation techniques.

Acknowledgements

This work was supported, in part, by ESPRIT project (6707) "ParForce" and undertaken while Mark Longley was visiting the University of Kent at Canterbury.

References

[1] M. Bruynooghe. A Practical Framework for the Abstract Interpretation of Logic Programs. *J. Logic Programming*, 10:91–124, 1991.

[2] M. Bruynooghe, M. Codish, and A. Mulkers. Abstract unification for a composite domain deriving sharing and freeness properties of program variables. In *ICLP'94 post-conference workshop on the verification and analysis of logic programs*, pages 213–230, Santa Margherita Ligure, Italy, 1994. June.

[3] M. Bruynooghe, G. Janssens, A. Callebaut, and B. Demoen. Abstract Interpretation: Towards the Global Optimization of Prolog Programs. In *SLP'87*, pages 192–204. MIT Press, 1987.

[4] F. Bueno, M. García de la Banda, and M. Hermenegildo. Effectiveness of Global Analysis in Strict Independence-Based Automatic Program Parallelization. In *International Symposium on Logic Programming*, pages 320–336. MIT Press, 1994.

[5] J.-H. Chang and A. M. Despain. Semi-intelligent backtracking of prolog based static data dependency analysis. In *JICSLP'85*. IEEE Computer Society, 1985.

[6] M. Codish, D. Dams, and E. Yardeni. Derivation and Safety of an Abstract Unification Algorithm for Groundness and Aliasing Analysis. In *ICLP'91*, pages 79–93, Paris, France, 1991. MIT Press.

[7] M. Codish, A. Mulkers, M. Bruynooghe, M. J. García de la Banda, and M. Hermenegildo. Improving abstract interpretation by combining domains. In *PEPM'93*. ACM Press, 1993.

[8] A. Cortesi and G. Filé. Abstract interpretation of logic programs: an abstract domain for groundness, sharing, freeness and compoundness analysis. In *PEPM'91*, pages 52–61. ACM Press, 1991.

[9] P. Cousot and R. Cousot. Abstract Interpretation and Application to Logic Programs. *J. of Logic Programming*, 13(2–3), 1992.

[10] D. Dams. Personal communication on linearity lemma 2.2. July, 1993.

[11] S. Debray and D. S. Warren. Automatic Mode Inference for Logic Programs. *J. of Logic Programming*, 5(3):207–230, 1988.

[12] S. K. Debray. Static inference of modes and data dependencies in logic programs. *ACM TOPLAS*, 11(3):418–450, July 1989.

[13] M. Hermenegildo and F. Rossi. Non-strict independent and-parallelism. In *ICLP'90*, pages 237–252, Jerusalem, 1990. MIT Press.

[14] D. Jacobs and A. Langen. Static Analysis of Logic Programs. *J. Logic Programming*, pages 154–314, 1992.

[15] G. Janssens and W. Simoens. On the Implementation of Abstract Interpretation Systems for (Constraint) Logic Programs. In *CC'94*, pages 172–198. Springer-Verlag, 1994.

[16] N. Jones and H. Søndergaard. *Abstract Interpretation of Declarative Languages*, chapter A Semantics-Based Framework for the Abstract Interpretation of Prolog, pages 123–142. Ellis Horwood, 1987.

[17] T. Karamori and T. Kawamura. Analyzing Success Patterns of Logic Programs by Abstract Hybrid Interpretation. Technical Report TR-279, ICOT, 1987.

[18] A. King. $Share \times Free$ Revisited. Technical report, Computing Laboratory, University of Kent at Canterbury, Canterbury, CT2 7NF, UK, 1994.

[19] A. King. A Synergistic Analysis for Sharing and Groundness which traces Linearity. In *ESOP'94*, pages 363–378, Edinburgh, UK, 1994. Springer-Verlag.

[20] A. King and P. Soper. Depth-k Sharing and Freeness. In *ICLP'94*, Santa Margherita Ligure, Italy, 1994. MIT Press.

[21] J. Lassez, M. J. Maher, and K. Marriott. *Foundations of Deductive Databases and Logic Programming*, chapter Unification Revisited. Morgan Kaufmann, 1987.

[22] B. Le Charlier and P. Van Hentenryck. Compositional bits. In *POPL'94*, 1994.

[23] B. Le Charlier and P. Van Hentenryck. Experimental Evaluation of a Generic Abstract Interpretation Algorithm for Prolog. *ACM Transactions on Programming Languages and Systems*, 1994.

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

[25] K. Marriott and H. Søndergaard. Analysis of constraint logic programs. In *NACLP'90*, pages 531–547. MIT Press, 1990.

[26] C. Mellish. *Abstract Interpretation of Declarative Languages*, chapter Abstract Interpretation of Prolog Programs, pages 181–198. Ellis Horwood, 1987.

[27] A. Mulkers, W. Simoens, G. Janssens, and M. Bruynooghe. On the Practicality of Abstract Equation Systems. Technical Report CW198, K. U. Leuven, Celestijnenlaan 200 A, 3001 Herverlee, Belgium, November 1994.

[28] K. Muthukumar and M. Hermenegildo. Combined determination of sharing and freeness of program variables through abstract interpretation. In *ICLP'91*, pages 49–63, Paris, France, 1991. MIT Press.

[29] K. Muthukumar and M. Hermenegildo. Compile-time Derivation of Variable Dependency through Abstract Interpretation. *J. of Logic Programming*, pages 315–437, 1992.

[30] U. Nilsson. *Abstract Interpretations and Abstract Machines: contributions to a methodology for the implementation of logic programs.* PhD thesis, Department of Computer and Information Science, 1992. Link"oping studies in science and technology dissertation no. 265.

[31] H. Søndergaard. An application of the abstract interpretation of logic programs: occur-check reduction. In *ESOP'86*, pages 327–338, New York, 1986. Springer-Verlag.

[32] R. Sundararajan and J. Conery. An abstract interpretation scheme for groundness, freeness, and sharing analysis of logic programs. In *12^{th} FST and TCS Conference*, New Delhi, India, December 1992. Springer-Verlag.

[33] A. Taylor. *High Performance Prolog Implementation.* PhD thesis, Basser Department of Computer Science, NSW 2006, Australia, July 1991.

[34] W. Winsborough. Multiple Specialisation Using Mimimal Function Graphs. *J. of Logic Programming*, 1990.

[35] W. Winsborough and A. Wærn. Transparent and-parallelism in the presence of shared free variables. In *ICLP'88*, pages 749–764. MIT Press, 1988.

[36] H. Xia. *Analyzing Data Dependencies, Detecting And-Parallelism and Optimizing Backtracking in Prolog Programs.* PhD thesis, University of Berlin, April 1989.

7 $Share_{Pvar} \times Free_{Pvar}$ Appendix

As with $Share_{Pvar} \times Free_{Pvar} \times Lin_{Pvar}$, the mapping $\alpha^{\mathcal{SF}}_{Pvar}$ and $\gamma^{\mathcal{SF}}_{Pvar}$ follow directly from sh_{Pvar} and fr_{Pvar}.

Definition 7.1 **($\alpha^{\mathcal{SF}}_{Pvar}$ and $\gamma^{\mathcal{SF}}_{Pvar}$)** *The abstraction and concretisation mappings* $\alpha^{\mathcal{SF}}_{Pvar} : \wp(Sub) \rightarrow Share_{Pvar} \times Free_{Pvar}$ *and* $\gamma^{\mathcal{SF}}_{Pvar} : Share_{Pvar} \times Free_{Pvar} \rightarrow \wp(Sub)$ *are defined by:*

$$\alpha^{\mathcal{SF}}_{Pvar}(\Phi) = \langle \alpha^{\mathcal{S}}_{Pvar}(\Phi), \alpha^{\mathcal{F}}_{Pvar}(\Phi)\rangle \quad \gamma^{\mathcal{SF}}_{Pvar}(\phi^{\mathcal{SF}}) = \gamma^{\mathcal{S}}_{Pvar}(\phi^{\mathcal{S}}) \cap \gamma^{\mathcal{F}}_{Pvar}(\phi^{\mathcal{F}})$$

Linearity can be tracked because, like before, if $\phi \in \gamma^{\mathcal{SF}}_{Pvar}(\phi^{\mathcal{SF}})$ then the variable multiplicity of $\phi(t)$ can be (partially) deduced from t and $\phi^{\mathcal{SF}}$. Lemma 4.1 explains how $\chi^{\mathcal{SF}}$ approximates χ.

Definition 7.2 **($\chi^{\mathcal{SF}}$)** *The abstract variable multiplicity operator* $\chi^{\mathcal{SF}} : Term \times (Share_{Pvar} \times Free_{Pvar} \times Lin_{Par}) \rightarrow \{0, 1, 2\}$ *is defined by:*

$$\chi^{\mathcal{SF}}(t, \phi^{\mathcal{SF}}) = \begin{cases} 0 \text{ if } var(t) \cap var(\phi^{\mathcal{S}}) = \emptyset \\ 1 \text{ else if } \forall u \in var(\phi^{\mathcal{S}}).\chi_u(t) \leq 1 \wedge \\ \qquad var(t) \cap \phi^{\mathcal{S}} = \phi^{\mathcal{F}} \qquad \wedge \\ \qquad \forall u, v \in var(t).rel(u, \phi^{\mathcal{S}}) \cap rel(v, \phi^{\mathcal{S}}) = \emptyset \\ 2 \text{ otherwise} \end{cases}$$

Lemma 7.1

$$var(t) \subseteq Pvar \wedge \phi \in \gamma^{\mathcal{SF}}_{Pvar}(\phi^{\mathcal{SF}}) \Rightarrow \chi(\phi(t)) \leq \chi^{\mathcal{SF}}(t, \phi^{\mathcal{SF}})$$

The corresponding proof is numbered 9.7.

Definition 7.3 **($mgu^{\mathcal{SF}}$)**

$$mgu^{\mathcal{SF}}(t, t', \phi^{\mathcal{SF}}) = \mu^{\mathcal{SF}} \text{ where}$$

$$\mu^{\mathcal{S}} = \phi^{\mathcal{S}} \setminus (rel(t, \phi^{\mathcal{S}}) \cup rel(t', \phi^{\mathcal{S}})) \cup \\ \begin{cases} rel(t, \phi^{\mathcal{S}}) \ \square\, rel(t', \phi^{\mathcal{S}}) \text{if } t \in \phi^{\mathcal{F}} \vee \chi^{\mathcal{SF}}(t', \phi^{\mathcal{SF}}) \leq 1 \\ rel(t, \phi^{\mathcal{S}})^{\star} \,\square\, rel(t', \phi^{\mathcal{S}}) \text{otherwise} \end{cases}$$

$$\mu^{\mathcal{F}} = \begin{cases} \phi^{\mathcal{F}} \cup \{t\} & \text{if } t' \in \phi^{\mathcal{F}} \\ \phi^{\mathcal{F}} \setminus var(rel(t, \phi^{\mathcal{S}})) & \text{otherwise} \end{cases}$$

The mapping $mgu^{\mathcal{SFL}}$ divides into two cases, like before, applying different analysis strategies according to whether $\phi(t)$ is free or $\phi(t')$ is free or linear. Because of the lack of precise linearity information, however, the second, less precise case is likely to be selected more frequently in $mgu^{\mathcal{SF}}$ than in $mgu^{\mathcal{SFL}}$. Lemma 9.8 is established in proof 9.8.

Lemma 7.2

$$\phi \in \gamma^{\mathcal{SF}}_{Pvar}(\phi^{\mathcal{SF}}) \wedge var(\phi(t)) \cap var(\phi(t')) = \emptyset \wedge var(t) \cup var(t') \subseteq Pvar \wedge \\ \varphi \in mgu(\{\phi(t) = \phi(t')\}) \wedge \varphi(\phi(t)) = \phi(t') \wedge mgu^{\mathcal{SF}}(t, t', \phi^{\mathcal{SF}}) = \mu^{\mathcal{SF}} \Rightarrow \varphi \circ \phi \in \gamma^{\mathcal{SF}}_{Pvar}(\mu^{\mathcal{SF}})$$

The $mgu^{\mathcal{SF}}$ relation, $restriction^{\mathcal{SF}}$, $entry^{\mathcal{SF}}$ and $exit^{\mathcal{SF}}$ are defined in a similar way to before.

8 $Share_{Pvar}$ Appendix

For $Share_{Pvar}$, without freeness or linearity information, a closure calculation cannot be avoided. However, this compares favourably with the two closure that are required in the standard abstract unification algorithm [14]. The proof for lemma 8.1 is numbered 9.9.

Definition 8.1 ($mgu^{\mathcal{S}}$)

$$\begin{aligned}&mgu^{\mathcal{S}}(t,t',\phi^{\mathcal{S}}) = \\ &\quad \phi^{\mathcal{S}} \setminus (rel(t,\phi^{\mathcal{S}}) \cup rel(t',\phi^{\mathcal{S}})) \ \cup \ rel(t,\phi^{\mathcal{S}})^{\star}\Box rel(t',\phi^{\mathcal{S}})\end{aligned}$$

Lemma 8.1

$$\begin{aligned}&\phi \in \gamma^{\mathcal{S}}_{Pvar}(\phi^{\mathcal{S}}) \wedge var(\phi(t)) \cap var(\phi(t')) = \emptyset \wedge var(t) \cup var(t') \subseteq Pvar \ \wedge \\ &\quad \varphi \in mgu(\{\phi(t) = \phi(t')\}) \wedge \varphi(\phi(t)) = \phi(t') \wedge mgu^{\mathcal{S}}(t,t',\phi^{\mathcal{S}}) = \mu^{\mathcal{S}} \Rightarrow \varphi \circ \phi \in \gamma^{\mathcal{S}}_{Pvar}(\mu^{\mathcal{S}})\end{aligned}$$

9 Proof Appendix

Proof 9.1 (for lemma 4.1) *Let $var(t) \subseteq Pvar$ and $\phi \in \gamma^{\mathcal{SFL}}_{Pvar}(\phi^{\mathcal{SFL}})$.*

1. *Suppose $\chi(\phi(t)) = 0$. Immediate.*
2. *Suppose $\chi(\phi(t)) = 1$. Thus there exists $v \in var(t)$ such that $u \in var(\phi(v))$. Since $v \in Pvar$, $v \in var(occ_{Pvar}(u,\phi))$ and thus $v \in var(\phi^{\mathcal{S}})$. Hence $\chi^{\mathcal{SFL}}(t,\phi^{\mathcal{SFL}}) \neq 0$.*
3. *Suppose $\chi(\phi(t)) = 2$.*
   (a) *Suppose $u \in var(t)$ such that $\chi_u(t) = 2$ and $v \in var(\phi(u))$. Thus, since $u \in Pvar$, $u \in var(occ_{Pvar}(v,\phi))$ and thus $u \in var(\phi^{\mathcal{S}})$. Hence $\chi^{\mathcal{SFL}}(t,\phi^{\mathcal{SFL}}) = 2$.*
   (b) *Suppose $u, v \in var(t)$ such that $w \in var(\phi(u)) \cap var(\phi(v))$ and $u \neq v$. Thus, since $u, v \in Pvar$, $u, v \in var(occ_{Pvar}(w,\phi))$ and therefore $rel(u,\phi^{\mathcal{S}}) \cap rel(v,\phi^{\mathcal{S}}) \neq \emptyset$. Hence $\chi^{\mathcal{SFL}}(t,\phi^{\mathcal{SFL}}) = 2$.*
   (c) *Suppose $v \in var(t)$ such that $\chi_u(\phi(v)) = 2$. Thus, since $v \in Pvar$, $v \notin \phi^{\mathcal{L}}$. Hence $\chi^{\mathcal{SFL}}(t,\phi^{\mathcal{SFL}}) = 2$.*

Proof 9.2 (for lemma 4.2) *Let $\phi \in \gamma^{\mathcal{SFL}}_{Pvar}(\phi^{\mathcal{SFL}})$, $var(\phi(t)) \cap var(\phi(t')) = \emptyset$, $var(t) \cup var(t') \subseteq Pvar$, $\varphi \in mgu(\{\phi(t) = \phi(t')\})$, $\varphi(\phi(t)) = \phi(t')$ and $mgu^{\mathcal{SFL}}(t,t',\phi^{\mathcal{SFL}}) = \mu^{\mathcal{SFL}}$.*

1. *Let $v \in Uvar$. To show $occ_{Pvar}(v,\varphi \circ \phi) \in \mu^{\mathcal{S}}$.*
   (a) *Suppose $v \notin cod(\varphi \circ \phi)$. Thus $v \notin var(\varphi \circ \phi(w))$ for all $w \in dom(\varphi \circ \phi)$.*
      i. *Suppose $v \notin dom(\varphi \circ \phi)$, that is, $\varphi \circ \phi(v) = v$. Thus $\phi(v) = v'$ and $\varphi(v') = v$. Suppose $v \neq v'$. Hence $v \in var(\phi(t)) \cup var(\phi(t'))$. Thus there exists $var(t) \cup var(t')$ such that $v \in var(\phi(w))$. But since $\phi(v) = v'$, $v \neq w$ and because $dom(\varphi) \cap cod(\varphi) = \emptyset$, $\varphi(v) = v$ and therefore $v \in var(\varphi \circ \phi(w))$. Hence $v \in cod(\varphi \circ \phi)$ which is a contradiction. Thus $v = v'$.*
         A. *Suppose $v \notin var(\phi(t))$ and $v \notin var(\phi(t'))$. Hence $v \notin cod(\varphi)$ and therefore $occ_{Pvar}(v,\varphi \circ \phi) = occ_{Pvar}(v,\phi)$. But $var(t) \cap var(occ_{Pvar}(v,\phi)) = \emptyset$ and $var(t') \cap var(occ_{Pvar}(v,\phi)) = \emptyset$. Hence $occ_{Pvar}(v,\varphi \circ \phi) \in \mu^{\mathcal{S}}$.*
         B. *Suppose $v \in var(\phi(t))$ and $v \notin var(\phi(t'))$. Since $\varphi \in mgu(\{\phi(t) = \phi(t')\})$, $v \in dom(\varphi)$ or $v \in cod(\varphi)$. Since $\varphi(v) = v$, $v \notin dom(\varphi)$ and thus $v \in cod(\varphi)$. Thus $v \in var(\varphi \circ \phi(t))$ and therefore $v \in var(\varphi \circ \phi(t'))$. Since $v \notin var(\phi(t')$, there exists $w \in var(\phi(t'))$ such that $v \in var(\varphi(w))$. Thus $v \in var(\varphi \circ \phi(t'))$ and since $v \notin cod(\varphi \circ \phi)$, $v = t'$ so that $\phi(t') = v$ which is a contradiction.*
         C. *Suppose $v \notin var(\phi(t))$ and $v \in var(\phi(t'))$. Like case 1(a)iB.*

D. *Suppose* $v \in var(\phi(t))$ *and* $v \in var(\phi(t'))$. *Since* $\varphi(v) = v$ *and* $v \notin cod(\varphi \circ \phi)$, $v \notin cod(\phi)$. *Thus* $v \in var(t)$ *and* $v \in var(t')$ *so that* $var(t) \cap var(t') \neq \emptyset$ *which is a contradiction.*

ii. *Suppose* $v \in dom(\varphi \circ \phi)$. *Since* $v \notin cod(\varphi \circ \phi)$, $occ_{Pvar}(v, \varphi \circ \phi) = \emptyset \in \mu^{\mathcal{S}}$.

(b) *Suppose* $v \in cod(\varphi \circ \phi) \setminus var(\varphi \circ \phi(t))$. *Suppose* $v \in cod(\varphi)$. *Thus* $v \in var(\varphi \circ \phi(t))$ *which is a contradiction. Suppose* $v \in dom(\varphi)$. *Thus* $v \notin cod(\varphi)$ *and hence* $v \notin cod(\varphi \circ \phi)$ *which is a contradiction. Hence* $occ_{Pvar}(v, \varphi \circ \phi) = occ_{Pvar}(v, \phi) \in \phi^{\mathcal{S}}$. *Suppose* $v \in var(\phi(t)) \cup var(\phi(t'))$. *Since* $v \notin var(\varphi \circ \phi(t))$, $v \in dom(\varphi)$ *and therefore* $v \notin cod(\varphi)$. *Hence* $v \notin cod(\varphi \circ \phi)$ *which is a contradiction. Thus* $var(t) \cap var(occ_{Pvar}(v, \phi))$ *and similarly* $var(t') \cap var(occ_{Pvar}(v, \phi)) = \emptyset$ *and therefore* $occ_{Pvar}(v, \varphi \circ \phi) \in \mu^{\mathcal{S}}$.

(c) *Suppose* $v \in cod(\varphi \circ \phi) \cap var(\varphi \circ \phi(t))$. *Note that* $occ_{Pvar}(v, \varphi \circ \phi) = \cup_{v \in var(\varphi(w))}$ $occ_{Pvar}(w, \phi)$.

i. *Suppose* $t \in \phi^{\mathcal{F}}$ *with* $\phi(t) = v_t$. *Thus* $\varphi = \{v_t \mapsto \phi(t')\}$. *Since* $v \in var(\varphi \circ \phi(t))$, $v \in var(\phi(t'))$. *Thus* $\{w \mid v \in var(\varphi(w)\} = \{v_t, v\}$. *Hence* $occ_{Pvar}(v, \varphi \circ \phi) \in rel(t, \phi^{\mathcal{S}}) \square rel(t', \phi^{\mathcal{S}}) \subseteq \mu^{\mathcal{S}}$.

ii. *Suppose* $\chi^{\mathcal{SFL}}(t', \phi^{\mathcal{SFL}}) \leq 1$. *There exists* $W_t \subseteq var(\phi(t))$ *and* $W_{t'} \subseteq var(\phi(t'))$ *such that* $occ_{Pvar}(v, \varphi \circ \phi) = \cup_{w \in W_t \cup W_{t'}} occ_{Pvar}(w, \phi)$. *Since* $v \in var(\varphi \circ \phi(t))$, $W_t \neq \emptyset$ *and thus* $W_{t'} \neq \emptyset$. *Suppose* $w, w' \in W_t$ *and* $w \neq w'$. *Since* $\chi^{\mathcal{SFL}}(t', \phi^{\mathcal{SFL}}) \leq 1$, *by lemma 4.1,* $\chi(\phi(t')) \leq 1$ *and because* $var(\phi(t)) \cap var(\phi(t')) = \emptyset$, *by lemma 2.1,* $w \notin var(\phi(t))$ *or* $w' \notin var(\phi(t))$ *which is a contradiction. Now suppose* $w, w' \in W_{t'}$ *and* $w \neq w'$. *Since* $dom(\varphi) \cap W_{t'} = \emptyset$, $var(\varphi(w)) \cap var(\varphi(w')) = \emptyset$ *which is a contradiction. Thus* $occ_{Pvar}(t, \varphi \circ \phi) \in rel(t, \phi^{\mathcal{S}}) \square rel(t', \phi^{\mathcal{S}}) \subseteq \mu^{\mathcal{S}}$.

iii. *Suppose* $t \notin \phi^{\mathcal{F}}$ *and* $\chi^{\mathcal{SFL}}(t', \phi^{\mathcal{SFL}}) = 2$. *There exists* $W_t \subseteq var(\phi(t))$ *and* $W_{t'} \subseteq var(\phi(t'))$ *such that* $occ_{Pvar}(v, \varphi \circ \phi) = \cup_{w \in W_t \cup W_{t'}} occ_{Pvar}(w, \phi)$. *Like in case 1(c)ii,* $W_{t'} = \{w'\}$ *so that* $occ_{Pvar}(t, \varphi \circ \phi) \in rel(t, \phi^{\mathcal{S}})^\star \square rel(t', \phi^{\mathcal{S}}) \subseteq \mu^{\mathcal{S}}$.

2. *Let* $v \in \mu^{\mathcal{F}}$.

(a) *Suppose* $t' \in \phi^{\mathcal{F}}$ *where* $\phi(t') = v_{t'}$.

i. *If* $v = t$ *then* $v \in fr_{Pvar}(\varphi \circ \phi)$ *since* $\varphi \circ \phi(t) = v_{t'}$.

ii. *If* $v \in \phi^{\mathcal{F}}$ *then* $v \in fr_{Pvar}(\varphi \circ \phi)$ *since* $fr_{Pvar}(\varphi \circ \phi) = fr_{Pvar}(\phi) \subseteq \phi^{\mathcal{F}}$.

(b) *Suppose* $t' \notin \phi^{\mathcal{F}}$. *Since* $v \notin var(rel(t, \phi^{\mathcal{S}}))$, $var(\phi(v)) \cap var(\phi(t)) = \emptyset$ *and because* $dom(\varphi) \subseteq var(\phi(t))$, $\varphi \circ \phi(v) = \phi(v) \in fr_{Pvar}(\phi)$.

3. *Let* $v \in \mu^{\mathcal{L}}$.

(a) *Suppose* $\chi^{\mathcal{SFL}}(t', \phi^{\mathcal{SFL}}) \leq 1$, *by lemma 4.1,* $\chi(\phi(t')) \leq 1$ *and thus* $\chi(\varphi \circ \phi(t)) \leq 1$.

i. *If* $v \in var(t)$ *then* $v \in ln_{Pvar}(\varphi \circ \phi)$ *since* $\chi(\varphi \circ \phi(t)) \leq 1$.

ii. *If* $v \in \phi^{\mathcal{L}}$ *then* $v \in fr_{Pvar}(\varphi \circ \phi)$ *since* $fr_{Pvar}(\varphi \circ \phi) = fr_{Pvar}(\phi) \subseteq \phi^{\mathcal{L}}$.

(b) *Suppose* $\chi^{\mathcal{SFL}}(t', \phi^{\mathcal{SFL}}) = 2$. *Since* $v \notin var(rel(t, \phi^{\mathcal{S}}))$, $var(\phi(v)) \cap var(\phi(t)) = \emptyset$ *and because* $dom(\varphi) \subseteq var(\phi(t))$, $\varphi \circ \phi(v) = \phi(v) \in ln_{Pvar}(\phi)$.

Proof 9.3 (for theorem 4.1) *Let* $\phi \in \gamma^{\mathcal{SFL}}_{Pvar}(\phi^{\mathcal{SFL}})$, $E = \{t_i = t_i'\}_{i=1}^n$ $var(\phi(t_i)) \cap var(\phi(t_j)) = \emptyset$, $var(E) \subseteq Pvar$, $\varphi \in mgu(\phi(E))$ $\varphi(\phi(t_i)) = \phi(t_i')$ *and* $mgu^{\mathcal{SFL}}(E, \phi^{\mathcal{SFL}}, \mu^{\mathcal{SFL}})$. *By induction on the steps of* $mgu^{\mathcal{SFL}}$ *and by lemma 4.2 there exists* $\theta \in mgu(\phi(E))$ *such that* $\theta \circ \phi \in \gamma^{\mathcal{SFL}}_{Pvar}(\psi^{\mathcal{SFL}})$. *But* $\theta \approx \varphi$ *[21] and thus* $\theta \circ \phi \approx \varphi \circ \phi$. *Hence* $\varphi \circ \phi \in \gamma^{\mathcal{SFL}}_{Pvar}(\psi^{\mathcal{SFL}})$.

Proof 9.4 (for lemma 4.3) *Let* $\phi \in \gamma^{\mathcal{SFL}}_{Pvar}(\phi^{\mathcal{SFL}})$ *and* $u \in Uvar$. *Now* $occ_{Pvar \cap Pvar'}(u, \phi) = occ_{Pvar}(u, \phi) \cap Pvar' \in \phi^{\mathcal{S}} \upharpoonright^{\mathcal{S}} Pvar'$. *Thus* $\phi \in \gamma^{\mathcal{S}}_{Pvar \cap Pvar'}(\phi^{\mathcal{S}} \upharpoonright^{\mathcal{S}} Pvar')$. *Moreover,* $\phi \in \gamma^{\mathcal{F}}_{Pvar \cap Pvar'}(\phi^{\mathcal{F}} \upharpoonright^{\mathcal{F}} Pvar')$ *since* $\phi^{\mathcal{F}} \cap Pvar' \subseteq fr_{Pvar \cap Pvar'}(\phi)$ *because* $\phi^{\mathcal{F}} \subseteq fr_{Pvar}(\phi)$. *Similarly,* $\phi \in \gamma^{\mathcal{L}}_{Pvar \cap Pvar'}(\phi^{\mathcal{L}} \upharpoonright^{\mathcal{L}})$.

Proof 9.5 (for theorem 4.2) *Let* $\phi_{in} \in \gamma^{\mathcal{SFL}}_{Pvar}(\phi^{\mathcal{SFL}}_{in})$, $\varphi \in mgu(\{\Upsilon(\phi_{in}(a_{call})) = a_{head}\})$, $\Upsilon(\phi_{in}(a_{call})) = \varphi(a_{head})$ *and* $\theta = unify(a_{head}, \epsilon, a_{call}, \phi_{in})$.

Thus $\theta = (\varphi \circ \epsilon) \upharpoonright Pvar$ *where* $\varphi \in mgu(\{\epsilon(a_{head}) = \Upsilon(\phi_{in}(a_{call}))\})$. *Observe that* $\varphi \in mgu(\{\epsilon(a_{head}) = \Upsilon(\phi_{in}(\Upsilon^{-1}(\Upsilon(a_{call}))))\})$ *and thus putting* $\sigma = \epsilon \cup (\Upsilon \circ \phi_{in} \circ \Upsilon^{-1})$, $\varphi \in mgu(\sigma(\{a_{head} = \Upsilon(a_{call})\}))$. *Note that* $\epsilon \in \gamma^{\mathcal{SFL}}_{Pvar}(\epsilon^{\mathcal{SFL}})$ *and* $\phi_{in} \circ \Upsilon^{-1} \in \gamma^{\mathcal{SFL}}_{\Upsilon(Pvar)}(\Upsilon(\phi^{\mathcal{SFL}}_{in}))$ *and hence* $\Upsilon \circ \phi_{in} \circ \Upsilon^{-1} \in \gamma^{\mathcal{SFL}}_{\Upsilon(Pvar)}(\Upsilon(\phi^{\mathcal{SFL}}_{in}))$. *Since* $var(\epsilon) \cap var(\Upsilon \circ \phi_{in} \circ \Upsilon^{-1}) = \emptyset$, $\sigma \in \gamma_{Pvar \cup \Upsilon(Pvar)}(\epsilon^{\mathcal{SFL}} \cup \Upsilon(\phi^{\mathcal{SFL}}_{in}))$. *Thus, by theorem 4.1, since* $var(\sigma(a_{head})) \cap var(\sigma(\Upsilon(a_{call}))) = \emptyset$, $var(a_{head}) \cup var(\Upsilon(a_{call})) \subseteq Pvar \cup \Upsilon(Pvar)$, $\varphi \in mgu(\sigma(\{a_{head} = \Upsilon(a_{call})\}))$ *and* $\varphi(\sigma(a_{head})) = \sigma(\Upsilon(a_{call}))$ *it follows that* $\varphi \circ \sigma \in \gamma^{\mathcal{SFL}}_{Pvar \cup \Upsilon(Pvar)}(\mu^{calSFL})$ *where* $\mu^{calSFL} = mgu^{\mathcal{SFL}}(\{a_{head} = \Upsilon(a_{call})\}, \epsilon^{\mathcal{SFL}} \cup \Upsilon(\phi^{\mathcal{SFL}}_{in}))$. *Thus, by lemma 4.3,* $\varphi \circ \sigma \in \gamma^{\mathcal{SFL}}_{Pvar}(\mu^{calSFL} \upharpoonright^{\mathcal{SFL}} Pvar)$ *and because* $(\varphi \circ \sigma) \upharpoonright Pvar = (\varphi \circ [\sigma \upharpoonright Pvar]) \upharpoonright Pvar = (\varphi \circ \epsilon) \upharpoonright Pvar$, $(\varphi \circ \epsilon) \upharpoonright Pvar \in \gamma^{\mathcal{SFL}}_{Pvar}(\mu^{calSFL} \upharpoonright^{\mathcal{SFL}} Pvar)$ *and therefore* $\theta \in \gamma^{\mathcal{SFL}}_{Pvar}(entry^{\mathcal{SFL}}(a_{head}, \epsilon^{\mathcal{SFL}}, a_{call}, \phi^{\mathcal{SFL}}_{in}))$.

Proof 9.6 (for theorem 4.3) *Like proof 9.5.*

Proof 9.7 (for lemma 7.1) *Let* $var(t) \subseteq Pvar$ *and* $\phi \in \gamma^{\mathcal{SF}}_{Pvar}(\phi^{\mathcal{SF}})$.

1. *Suppose* $\chi^{\mathcal{SF}}(t, \phi^{\mathcal{SF}}) = 0$. *Put* $\phi^{\mathcal{L}} = \emptyset$. *Thus* $\phi^{\mathcal{L}} \subseteq ln_{Pvar}(\phi)$ *and* $\phi \in \gamma^{\mathcal{SFL}}_{Pvar}(\phi^{\mathcal{SFL}})$. *But* $\chi^{\mathcal{SFL}}(t, \phi^{\mathcal{SFL}}) = 0$ *so that, by lemma 4.1,* $\chi(\phi(t)) = 0$.

2. *Suppose* $\chi^{\mathcal{SF}}(t, \phi^{\mathcal{SF}}) = 1$. *Let* $v \in var(t)$.

   (a) *Suppose* $v \in var(\phi^{\mathcal{S}})$. *Then* $v \in \phi^{\mathcal{F}}$ *and hence* $\chi(\phi(v)) = 1$ *so that* $v \in ln_{Pvar}(\phi)$.

   (b) *Suppose* $v \notin var(\phi^{\mathcal{S}})$. *Then* $\chi(\phi(v)) = 0$ *so that* $v \in ln_{Pvar}(\phi)$.

   Put $\phi^{\mathcal{L}} = var(t)$. *Thus* $\phi^{\mathcal{L}} \subseteq ln_{Pvar}(\phi)$ *and hence* $\phi \in \gamma^{\mathcal{SFL}}_{Pvar}(\phi^{\mathcal{SFL}})$. *But* $\chi^{\mathcal{SFL}}(t, \phi^{\mathcal{SFL}}) = 1$ *so that, by lemma 4.1,* $\chi(\phi(t)) \leq 1$.

3. *Suppose* $\chi^{\mathcal{SF}}(t, \phi^{\mathcal{SF}}) = 2$. *Immediate.*

Proof 9.8 (for lemma 7.2) *Like cases 1a, 1b, 1c and 2 of lemma 4.2.*

Proof 9.9 (for lemma 8.1) *Like cases 1a, 1b, 1(c)iii of lemma 4.2.*