Aliasing and Groundness Analysis of Logic Programs through Abstract Interpretation and its Safety

Werner Hans and Stephan Winkler*
RWTH Aachen, Lehrstuhl für Informatik II
Ahornstraße 55, W-5100 Aachen, Germany

Aachener Informatikberichte
Nr. 92–27

Abstract

This paper describes a safe method, which derives aliasing and groundness properties of logic programs (with equality). The abstract domain bases on the one proposed in [Jacobs,Langen89], which has the advantage of great potential precision and supports the simple computation through abstract interpretation. The abstract execution follows the framework suggested in [Bruynooghe91] determining useful information for all points within the program. The abstract operations as well as the embedding in this framework are formulated in a precise manner and are proven correct; the latter is the crucial topic here. In addition, several refinements are examined.

*The authors are supported by the grant In 20/6-1 from the 'Deutsche Forschungsgemeinschaft'. e-mail: {hans,winkler}@zeus.informatik.rwth-aachen.de

1 Introduction

Aliasing and groundness analysis is an important global analysis method offering many opportunities to improve the execution of compiled logic programs[1]. For example, the knowledge of groundness is advantageous to improve the unification process, and the knowledge of absent aliasing allows the optimization of the backtracking behaviour. The derived results may also enable the rearrangement of the clauses and literals.

Groundness analysis of logic programs was first proposed in [Mellish81] to improve the compilation of programs. An ancestor paper [Mellish86] used abstract interpretation to infer useful global properties. Abstract interpretation intuitively means that the programs are executed (interpreted) using a simple abstract domain instead of the concrete domain. Nontermination is avoided by the usage of finite abstract domains and memo tables. The results of this analysis initiated further research: [Debray,Warren88] developed an abstract domain considering also unbound variables. The authors mentioned the risk of wrong results due to variable aliasing and solved this problem by coarse approximations. In [Debray89], this solution is improved by extending the domain with a second part, only used to store the aliasing of variables. At the same time [Jacobs,Langen89] suggested a different domain, which captures both groundness and aliasing derivation in an accurate manner. Our approach originates from this domain.

Beside the sequential optimization, the automatic parallelization of logic programs is supported, which seems very promising, due to the high degree of implicit parallelism. On the one hand, the several clauses can be executed in parallel (or-parallelism), on the other hand, the different literals within each clause can be executed in parallel (and-parallelism). In particular the *restricted and-parallelism* ([DeGroot84]) is well suited, because it retains the single assignment property of logic variables and preserves enough parallelism: Two literals can be safely executed in parallel, if they do not share any variables, due to binding conflicts. This enforces, that the arguments must be ground or independent.

Logic programs are usually translated in an extended intermediate language containing several tests (conditional dependence expressions) to fix the conditions, which support parallel execution ([Muthukumar,Hermenegildo90b, Jacobs,Langen88]). But implementations have shown that the execution of this tests — in particular the independence checks — is quite expensive.

Global program analysis helps to decrease the number of such tests; in the average, the decrease is enormous.

Example 1: Suppose the substitution $\{W/f_0(C),\ X/f_1(A,B),\ Y/f_2(A),\ Z/1\}$ and its abstract representation $\{\{A,X,Y\},\{B,X\},\{C,W\}\}$ expressing the dependencies between the listed variables. Given the goal

$$p(X),q(W,Z),r(Z,Y),$$

there are four possibilities for exploiting parallelism, but only two allow restricted and-parallelism (without any test).

- $p \| q$ is possible, because Z is ground, and X and W do not share variables. Groundness of Z is known, due to its absence from the abstract representation.

[1] We assume static, cut–free Prolog programs.

- p‖r is not valid, due to the potential binding conflict of variable A through X and Y. This aliasing is expressed by the set {A,X,Y}.
- q‖r is possible.
- p‖q‖r is not valid. □

It is important that expressive abstract domains are accompanied with precise operations. Two different approaches were presented independently in [Jacobs,Langen89] and [Muthukumar,Hermenegildo90b]. In [Muthukumar,Hermenegildo91] the core operation became more closely related to the former although their execution strategy remained different: In [Jacobs,Langen89] a pure bottom-up strategy is used, while the latter authors chose a top-down driven bottom-up strategy.

The core of abstract interpretation is the unification process in the abstract domain, and its precision determines the quality of the whole analysis. Unification is performed while checking the application of clauses.

Example 2: Assume the clause *append([X|T],Y,[X|T1]):-append(T,Y,T1)* and the goal *append(A,[B],L)*. It is intended, that the last element of the list L should be extracted. The initial abstraction is

$$\{\{A\},\{B\},\{L\}\} \cup \{\{X\},\{T\},\{Y\},\{T1\}\}.$$

The first part is inherited from the goal, and the second one reflects the fact that on procedure entry all new variables are independent in pairs. Abstract unification yields (after 3 steps):

$$\{\{A,T\},\{A,L,X\},\{A,L,T,X\},\{A,L,T1,X\},\{A,L,T,T1,X\},\{L,T1\},\{B,Y\}\}.$$

Performing the local computation, the dependencies of the global variables are removed:

$$\{\{T\},\{X\},\{T,X\},\{T1,X\},\{T,T1,X\},\{T1\},\{Y\}\}.$$

This abstraction expresses that Y is independent from X,T, and T1. □

Intuitively[2], one basic unification operation takes the followings steps (t_1 and t_2 are unified):

1. Extract from the abstraction κ all sets, which have at least one variable in common with the terms t_1, t_2, named S_1 and S_2, respectively,
2. create the 'closures' of S_1 and S_2,
3. join the elements of the closed sets in pairs,
4. remove S_1 and S_2 from κ and add the join.

[2]This intuitive description will be formally specified in the following sections.

The second step is the crucial part in computing the abstraction of the resulting substitution. Due to the lack of information at the abstract level, approximations are necessary to obtain correct results. Each closure operation introduces extra aliasing, possibly not occurring at the concrete level. This impedes groundness propagation, too.

Several improvements are possible by the management of additional information. Considering the concrete unification of two terms, special situations occur frequently in practice[3]:

- One term is ground: Then, the unifier maps variables to ground terms and yields that the second term becomes ground, too[4]. In the preceding example, groundness of L yields {{T},{Y}}.
- One term is an unbound variable: This appears usually, while performing the unification at procedure entry. Then, only *one binding* occurs and ensures that the abstract counterpart needs no closure operation.

  This refinement was suggested in [Muthukumar,Hermenegildo91]. If we know that A (in the example) is unbound, then unification leads to
  {{T},{T1},{T1,X},{X},{Y}},
  and shows the independence of T from any other variable, too.
- The work of [Bruynooghe91] introduces unifiers, actually expressing a match. This ensures that at least one closure operations is superfluous.
- If the terms contain each (unbound) variable at most once and the terms do not share variables, then, no sharing is introduced through unification. Even if only one term fulfills the first assumption, sharing occurs in a restricted manner. Both cases reduce the number of closure operations.

  This refinement was first proposed in [Codish et al.91]. But the authors used the (in some sense weaker) abstract domain of [Søndergaard86] and proved the soundness of their approach.

Note that the third refinement is some kind of generalization of the second one, due to the trivial possibility of binding orientation, although, the same improvement cannot be retained. In the same sense the last refinement generalizes the first one.

The incorporation of the second and fourth refinements enforces the extension of the abstract domain to become able to store 'unboundness' of variables and the absence of sharing inside the terms. The result is the cartesian product of two finite abstract domains, similar to [Debray89, Søndergaard86, Muthukumar,Hermenegildo91]. One component stores *definite instantiations* of variables and the second one stores *potential aliasing*. To keep the presentation clear, the extension is chosen small. But it seems very promising to manage additional properties, which can furthermore improve the compilation. However, the main topic of this paper is the presentation of the refined analysis in a precise fashion and the proof of its soundness.

[3]Successful unification is assumed. Failure behaviour is added in Section 6.

[4]Although this situation is also captured by the original approach, the special treatment is worthwhile because the remaining expensive closure operation can be avoided.

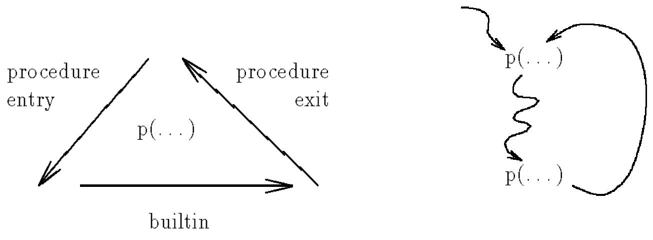

Figure 1: The Abstract Operations (a) and the Representation of Recursion (b)

One further improvement is *preunification*: In general, some (concrete) parts of terms are known, while the rest is managed with abstractions. Preunification simply means, to use the known part to detect failing unifications or to decompose structures (flattening).

The abstract unification operations are embedded in the framework of [Bruynooghe91]. He specified three local operations *procedure entry, procedure exit*, and *builtin* processing, which are related to the representation of the concrete computation as an AND-OR-graph.

Given a set of queries, each successful computation can be sketched by a (concrete) AND-tree, which visualizes the path chosen in the search tree. Each node corresponds to a (predicate/builtin) call and the edges show the flow of execution. These AND-trees are subsumed by *one* AND-OR-graph at the abstract level (cf. to Figure 5 below). Each node of the AND-tree is related to an AND-node of the graph. The alternatives of each call are given by a following OR-node. Therefore the different concrete AND-trees correspond to particular paths in the AND-OR-graph.

Each predicate call distinguishes two operations (Figure 1a): The *procedure entry* corresponds to the invocation and modifies the current abstraction. In order to work with finite representations, several 'global' information is removed and replaced some other information.

The counterpart of procedure entry is the *procedure exit* operation used to restore and update abstractions. Now, the abstract interpretation consists of repeated procedure entries/exits.

Predefined builtins are processed via special abstract operations specified for each builtin. Especially the equality operator (=) and the relational operator (<) are considered in this paper.

The nodes of the AND-OR-graph are annotated with abstractions subsuming the respective substitutions of the AND-trees. Related to the entry and the exit operations, each node has an entry and an exit abstraction[5].

This representation is well suited unless in the presence of recursive calls (Figure 1b). Then, there is the risk of infinite AND-OR-graphs. The graphical representation is solved by a backward edge; the computation is done with a fixpoint operation: If an ancestor procedure entry happens with exactly the same (finite) abstraction, then, procedure exit is immediately performed with an approximate default abstraction (initially the bottom

[5]The exit abstraction of some node and the entry abstraction of its 'successor' are equal.

element of the domain). Reaching the first call (from bottom), some intermediate exit abstraction is determined and the reevaluation of this predicate is initiated (updating the default value). This repeated reevaluation continues until the intermediate substitution does not change. Then the (top) exit is performed.

To store intermediate results and default abstractions memo tables are useful. They can further be used to prune other reevaluation. [Bruynooghe91] specified three local conditions and proved that their fulfillment yields the global soundness of the abstract interpreter following his strategy. We will therefore show that these conditions are satisfied by our specified operations.

The rest of this paper presents the analysis approach originated from [Jacobs,Langen89], together with the refinements in a mathematical setting, and proves their soundness. The next section introduces the basic notions. The soundness of the determination of aliasing, both for the general case, as well as for frequently occurring special settings are proved in the third section (at an abstract level). The following section describes the extensions needed to detect the special settings. Section 5 combines both abstract domains and operations, and introduces our finite representation, which is closely related to practical use. The three conditions of [Bruynooghe91] are verified in Section 6. The specification of a simplified[6] algorithm for aliasing and groundness analysis concludes this section. The remaining section discusses optimizing opportunities.

2 Preliminary

Execution of logic programs results in generation and modification of terms, which are managed through substitutions:

Definition 1: Term, Substitution
Let V and FS be the countable sets of *variables* and *functor symbols*, respectively.
a) By $T_{FS}(V)$ the set of all *terms* t constructed from FS and V are denoted. The variables in t are denoted by $var(t)$.
b) A substitution $\sigma \in Sub$ is an idempotent mapping σ:V$\longrightarrow T_{FS}(V)$ with a finite *domain* $dom(\sigma) := \{x \mid x \neq x\sigma\}$. Its *range* and *co-domain* are defined by $(rgv(\sigma) =)\ rgv_V(\sigma) := \bigcup_{x \in V} var(x\sigma)$ and $codom(\sigma) := rgv_{dom(\sigma)}(\sigma)$. □

We make in particular heavy use of the notion of *dom*, *codom*, and *rgv*. Note that the definition distinguishes between the co-domain of some substitution and its range. Several basic properties are mentioned in the following

Remark:

- $codom(\sigma) \subset rgv(\sigma)$,
- $dom(\sigma) \cup rgv(\sigma) = V$, and
- $dom(\sigma) \cap codom(\sigma) = dom(\sigma) \cap rgv(\sigma) = \emptyset$. □

[6]Fixpoint computation is done by full reevaluation of the goal, instead of some recursive predicates.

Given two terms t_1 and t_2, t_1 is called *ground*, if $var(t_1) = \emptyset$; t_1 and t_2 are *independent*, if $var(t_1) \cap var(t_2) = \emptyset$. The extension of the application of σ to terms is straight forward. The composition of two substitutions σ, θ is denoted by $\sigma\theta$ $(t\sigma\theta = \theta(\sigma(t)))$. A substitution σ_1 is *more general* than σ_2, if $\sigma_2 = \sigma_1\theta$ for some substitution θ.

Definition 2: Match, Unifier, mgu
Let $t_1, t_2 \in T_{FS}(V)$.
a) A substitution σ is called a *match* of t_1 to t_2, if $t_1\sigma = t_2$.
b) A substitution σ is called a *unifier* of t_1 and t_2, if $t_1\sigma = t_2\sigma$. It is called the *most general unifier* of t_1 and t_2 $(mgu(t_1, t_2))$, if it is more general than any other unifier. t_1 and t_2 are *unifyable*, if they have a common unifier. □

Remark: $dom(mgu(t_1, t_2)) \cup codom(mgu(t_1, t_2)) \subset var(t_1) \cup var(t_2)$. □

Sometimes an equivalent set of equations in solved form is considered, instead of the mgu. This set is implicit evaluated during the unification process ([Martelli,Montanari82]).

The next definition relates the concrete set of substitutions and its abstract representation via concretization and abstraction functions in a general setting.

Definition 3: Abstraction
An *abstraction* of $(\mathcal{P}(Sub), \subset, \cup)$ is a 5-tuple $\bar{\mathcal{A}} = (ASub_\perp, \sqsubset, \sqcup, \alpha, \gamma)$ satisfying:

- $(ASub_\perp, \sqsubset, \sqcup)$ is a semilattice with some bottom element $\perp$,
- the *concretization* function $\gamma : ASub_\perp \longrightarrow \mathcal{P}(Sub)$ is monotonic with $\gamma(\perp) = \emptyset$,
- the *abstraction* function $\alpha : \mathcal{P}(Sub) \longrightarrow ASub_\perp$ is monotonic with:

$$\forall \Sigma \subset Sub, \kappa \in ASub_\perp : \Sigma \subset \gamma(\alpha(\Sigma)) \text{ and } \kappa = \alpha(\gamma(\kappa)),$$

- $\sqsubset$ is the partial ordering and $\sqcup$ is the least upper bound (lub) operation with
$\forall \kappa, \kappa' \in ASub_\perp : \kappa \sqsubset \kappa' \Leftrightarrow \gamma(\kappa) \subset \gamma(\kappa') \qquad \kappa \sqcup \kappa' = \alpha(\gamma(\kappa) \cup \gamma(\kappa'))$. □

Variable multiplicity extends the notion of groundness. Its definition captures both the non-occurrence of variables within some term (groundness) as well as the unique occurrence. The latter means that there are no variables inside this term locally shared. We call such terms unshared or *linear*. The application of the *max* operation at the abstract level stresses its approximate behaviour, e.g., value 2 reflects the case that the term *may* become shared under some substitution.

Definition 4: Variable Multiplicity
a) The variable multiplicity $\chi : T_{FS}(V) \longrightarrow \{0,1,2\}$ of some term t is defined by:

$$\chi(t) = \begin{cases} 0 & ;\text{if } var(t) = \emptyset, \\ 1 & ;\text{if every variable occurs only once,} \\ 2 & ;\text{otherwise.} \end{cases}$$

b) Its counterpart at the abstract level is the *abstract variable multiplicity* $\chi^{\mathcal{A}} : T_{FS}(V) \times ASub_\perp \longrightarrow \{0,1,2\}$: $\chi^{\mathcal{A}}(t,\kappa) = \max\{\chi(t\sigma) \mid \sigma \in \gamma(\kappa)\}$. □

Definition 5: Closure under Union, Reduction
a) For any set S, the closure under union of S, denoted by S^*, is the least set T with:
$S \subset T$ and with $S, S' \in T \Longrightarrow S \cup S' \in T$.
b) Let A be a set, $\mathcal{S} \subset \mathcal{P}(A)$. The reduction of $\mathcal{S}$ to T⊂A is: $\mathcal{S}_T := \{U \in \mathcal{S} \mid T \subset U\}$, which collects all supersets of T. □

3 Aliasing

This section deals with the representation of aliasing of variables and discusses the soundness of several unification operations performed during the abstract execution. For this moment, no finiteness of the abstract representation is required.

The discussion starts with the general case. Here, the abstract counterpart of unification is specified quite similar to [Jacobs,Langen89][7] and the missing proof is given.

The remaining subsections present three refinements: unbound variable occurrence, the presence of independent terms, and the matching case.

This section is concluded by the definition of an intermediate abstraction.

3.1 General Case

The representation of aliasing founded upon the notion of occurrence sets and sharing. Given a variable x, its occurrence set under the substitution σ contains all variables, through x occurs. The function sharing $\mathcal{A}$ simply collects all relevant $(\neq \emptyset)$ occurrence sets.

Definition 6: Occurrence, Sharing
The *occurrence* and the *sharing* are mappings $occ : Sub \times V \longrightarrow \mathcal{P}(V)$ and $\mathcal{A} : Sub \longrightarrow \mathcal{P}(\mathcal{P}(V))$ defined by:
$occ(\sigma, x) = \{y \in V \mid x \in var(y\sigma)\} \qquad \mathcal{A}(\sigma) = \{occ(\sigma,x) \neq \emptyset \mid x \in V\}$. □

Given the sharing $\mathcal{A}(\sigma)$, groundness of $y\sigma$ $(y \in V)$ is known, if y is contained in no set of $\mathcal{A}(\sigma)$. Otherwise, there is a variable $x \in var(y\sigma)$ yielding the contradiction $y \in occ(\sigma,x) \in \mathcal{A}(\sigma)$. Similarly, two terms $x\sigma$,$y\sigma$ $(x, y \in V)$ are independent, if there is no variable which occurs through both of them. Therefore, these two terms are independent, if there is no set in $\mathcal{A}(\sigma)$ containing both x and y $(\mathcal{A}(\sigma)_{\{x,y\}} = \emptyset)$.

Example 3: Assume $\sigma = \{X/f_1(A,B), Y/f_2(A), Z/1\}$[8]. Then $occ(\sigma, A) = \{A, X, Y\}$, $occ(\sigma, B) = \{B, X\}$, and $occ(\sigma, X) = occ(\sigma, Y) = occ(\sigma, Z) = \emptyset$.
Therefore $\mathcal{A}(\sigma) = \bigcup_{x \in \{A,B,X,Y,Z\}} occ(\sigma,x) \cup \{\{x\} \mid x \in V \backslash \{A,B,X,Y,Z\}\}$. □

Some basic properties of occurrence sets:

Lemma 1: Let $x \in V, t \in T_{FS}(V), \sigma, \theta \in Sub$.

[7] Changes become necessary, due to technical reasons.
[8] The examples are written in Prolog like notation differing with the theoretical part, especially, the notation of variables differs.

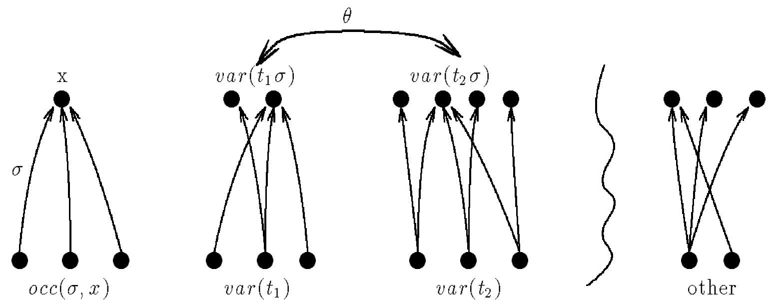

Figure 2: Influences of Concrete (Unification-)Bindings to the Occurrence Sets

a) $occ(\sigma\theta, x) = \bigcup_{y \in occ(\theta,x)} occ(\sigma, y)$,
b) $x \in var(t\sigma) \Leftrightarrow occ(\sigma, x) \cap var(t) \neq \emptyset$, and
c) $occ(\sigma, x) \neq \emptyset \Leftrightarrow x \in occ(\sigma, x) \Leftrightarrow x \in rgv(\sigma)$. □

The relationship of the concrete operations and the abstract ones, while unifying two terms t_1 and t_2, is sketched in Figure 2[9]. This figure shows at the lower row the variables of t_1 and t_2, the upper row sketches the variables of $t_1\sigma$ and $t_2\sigma$. Selecting a variable at the upper row, its occurrence set is determined by all adjacent variables of the incoming arrows.

At the concrete level, bindings are performed at the upper row. But, this row is in general unknown during abstract execution, only the lower row is known. Therefore, the effects of the unification on the occurrence sets must be carefully examined: If at the concrete level one variable is bound to some term, then, all participating occurrence sets need an update, i.e., if variable x is bound to a term containing the variable y, then the new occurrence set of y is the join of the old ones of x and y. More formally:

Theorem 1: Let $t_1, t_2 \in T_{FS}(V), \sigma, \theta \in Sub$, $t_1\sigma$ and $t_2\sigma$ unifyable with $\theta = mgu(t_1\sigma, t_2\sigma)$, $S_i = \{S \in \mathcal{A}(\sigma) \mid S \cap var(t_i) \neq \emptyset\}$ (i=1,2), and $T = \{S^1 \cup S^2 \mid S^1 \in S_1^*, S^2 \in S_2^*\}$.
Then $\mathcal{A}(\sigma\theta) \subset \mathcal{A}(\sigma) \backslash S_1 \backslash S_2 \cup T$. □

Proof: Let $x \in V, \emptyset \neq occ(\sigma, x) \in \mathcal{A}(\sigma)$. Note that the other case $\emptyset = occ(\sigma, x)$ implies $x \notin rgv(\sigma) \supset var(t_1\sigma) \cup var(t_2\sigma) \supset dom(\theta) \cup codom(\theta)$; therefore $occ(\theta, x) = \{x\}$ and $occ(\sigma\theta, x) \overset{La.1}{=} occ(\sigma, x) = \emptyset \notin \mathcal{A}(\sigma\theta)$.

It remains to show $occ(\sigma\theta, x) \in \mathcal{A}(\sigma) \backslash S_1 \backslash S_2 \cup T$.

Case 1: $x \notin var(t_1\sigma) \cup var(t_2\sigma)$.
Therefore $x \notin dom(\theta) \cup codom(\theta)$. This yields $occ(\theta, x) = \{x\}$ and $occ(\sigma\theta, x) = occ(\sigma, x) \in \mathcal{A}(\sigma)$. Clearly, $occ(\sigma, x) \cap var(t_i) = \emptyset$ (La.1). This gives $occ(\sigma\theta, x) \notin S_i$, and finally $occ(\sigma\theta, x) \in \mathcal{A}(\sigma) \backslash S_1 \backslash S_2$.

[9] The strict separation of the different variable sets is only for clarity.

Case 2: $x \in var(t_1\sigma)$.
a) From $x \in dom(\theta) \Rightarrow x \notin rgv(\theta) \Rightarrow occ(\theta, x) = \emptyset \Rightarrow occ(\sigma\theta, x) = \emptyset \notin \mathcal{A}(\sigma\theta)$.
b) Otherwise: $x \in rgv(\theta) \Rightarrow x \notin dom(\theta) \Rightarrow x \in occ(\theta, x)$. Note that $x \in var(t_1\sigma)$ implies $occ(\theta, x) \subset var(t_1\sigma) \cup var(t_2\sigma)$. Then

$$occ(\sigma\theta, x) \overset{La.1}{=} \bigcup_{y \in occ(\theta,x)} occ(\sigma, y) = \underbrace{\bigcup_{y_1 \in Y_1} \underbrace{occ(\sigma, y_1)}_{\in S_1}}_{\in S_1^*} \cup \underbrace{\bigcup_{y_2 \in Y_2} \underbrace{occ(\sigma, y_2)}_{\in S_2}}_{\in S_2^*} \in T$$

($Y_i := occ(\theta, x) \cap var(t_i\sigma)$) because of

- $x \in occ(\theta, x) \cap var(t_1\sigma) \Rightarrow Y_1 \neq \emptyset$,
- $x \in rgv(\theta) \cap var(t_1\sigma) \Rightarrow \exists y \in var(t_2\sigma) : x \in var(y\theta) \Rightarrow Y_2 \neq \emptyset$,
- $y_i \in var(t_i\sigma) \Rightarrow occ(\sigma, y_i) \cap var(t_i) \neq \emptyset \Rightarrow \emptyset \neq occ(\sigma, y_i) \in S_i$ (i=1,2).

Case 3 : $x \in var(t_2\sigma)$ is symmetric to case 2. □

Example 4: (refer to Example 2) Consider again the clause *append([X|T],Y,[X|T1]):- append(T,Y,T1)*, the goal *append(A,[B],L)*, and the abstraction $\mathcal{A}(\sigma) = \{\{A\}, \{B\}, \{L\}, \{X\}, \{T\}, \{Y\}, \{T1\}\}$. The overall unification leads to three basic unifications ([X|T],A), (Y,[B]), and ([X|T1],L). The first pair yields $S_1 = \{\{X\}, \{T\}\}$, $S_1^* = \{\{X\}, \{X, T\}, \{T\}\}$, $S_2 = S_2^* = \{\{A\}\}$, $T = \{\{A, X\}, \{A, X, T\}, \{A, T\}\}$, and $\mathcal{A}(\sigma\theta_1) \subset \{\{B\}, \{L\}, \{Y\}, \{T1\}\} \cup T$.
Subsequent unifications of the remaining pairs derive:
$\mathcal{A}(\sigma\theta_1\theta_2 \quad) \subset \{\{L\}, \{T1\}, \{A, X\}, \{A, X, T\}, \{A, T\}, \{B, Y\}\}$, and
$\mathcal{A}(\sigma\theta_1\theta_2\theta_3) \subset \{\{A, T\}, \{A, L, X\}, \{A, L, T, X\}, \{A, L, T1, X\}, \{A, L, T, T1, X\}, \{L, T1\}, \{B, Y\}\}$. □

The next statement offers the first opportunity of optimization, due to the removal of some operations (implementation issue), although it is captured by the general theorem. Furthermore, it stresses how groundness propagation is done.

Corollary 1: Let t_i, σ, θ, S_i as in Theorem 1. $\chi(t_1\sigma) = 0$ implies $\mathcal{A}(\sigma\theta) \subset \mathcal{A}(\sigma) \backslash S_1 \backslash S_2$. □

Proof: $\chi(t_1\sigma) = 0 \overset{La.1}{\Rightarrow} \forall x \in V : occ(\sigma, x) \cap var(t_1) = \emptyset \Rightarrow S_1 = \emptyset \Rightarrow T = \emptyset$. □

3.2 Unbound Variable Occurrence

This refinement was mentioned in [Muthukumar,Hermenegildo91]. It deals with variables participating in the concrete unification. Then, the resulting unifier is very restrictive (one binding occurs) and enables the removal of both (expensive) closure operations. This simplifies the computation at the abstract level, and — more important — it increases the precision because less potential aliasing is noted and, therefore, the better approximation is done.

Theorem 2: Let t_i, σ, θ, S_i as in Theorem 1, $t_1\sigma \in V$, and $T = \{S^1 \cup S^2 \mid S^1 \in S_1, S^2 \in S_2\}$.
Then $\mathcal{A}(\sigma\theta) \subset \mathcal{A}(\sigma)\backslash S_1 \backslash S_2 \cup T$. $\square$

Proof: Let x∈V, $\emptyset \neq occ(\sigma, x) \in \mathcal{A}(\sigma)$.

Case 1: $x \notin var(t_1\sigma) \cup var(t_2\sigma)$ equals the proof of Theorem 1.

Case 2: $x = t_1\sigma$: $x \in dom(\theta)$ yields $occ(\sigma\theta, x) = \emptyset$. Otherwise $x \in rgv(\theta) \Rightarrow t_2\sigma \in V$. ($t_2\sigma \notin V$ implies $x \in dom(\theta)$). Therefore

$$occ(\sigma\theta, x) = \underbrace{occ(\sigma, x)}_{\in S_1} \cup \underbrace{occ(\sigma, t_2\sigma)}_{\in S_2} \in T.$$

Case 3: $x \in var(t_2\sigma)$: Consider only $x \in rgv(\theta)$.

$$occ(\sigma\theta, x) = \underbrace{occ(\sigma, x)}_{\in S_2} \cup \underbrace{occ(\sigma, t_1\sigma)}_{\in S_1} \in T.$$ $\square$

Example 5: (cf. Examples 2 and 4) We assume the same situation except the additional knowledge of the freeness of the variables A and B, i.e., $A\sigma, B\sigma \in V$. Now we compute:

$$\begin{aligned}
&\mathcal{A}(\sigma\theta_1) \subset \{\{B\}, \{L\}, \{Y\}, \{T1\}, \{A, X\}, \{A, T\}\},\\
&\mathcal{A}(\sigma\theta_1\theta_2) \subset \{\{L\}, \{T1\}, \{A, X\}, \{A, T\}, \{B, Y\}\},\\
&\mathcal{A}(\sigma\theta_1\theta_2\theta_3) \subset \{\{A, T\}, \{A, L, X\}, \{A, L, T1, X\}, \{L, T1\}, \{B, Y\}\}.
\end{aligned}$$ $\square$

3.3 Independent Terms

The following refinement is well suited to perform procedure entry because the selected clause introduces fresh variables being independent from the ones of the goal. As long as the variables of the clause participates the first time[10] in the unification process (see Section 6), this optimization can be safely applied. Nevertheless, the general application is possible, due to the next lemma, which states a condition to determine the independence of two terms.

Theorem 3: Let t_i, σ, θ, S_i as in Theorem 1. Let $var(t_1\sigma) \cap var(t_2\sigma) = \emptyset$ and

$$T = \begin{cases}
\emptyset & ; \chi(t_1\sigma) = 0 \vee \chi(t_2\sigma) = 0,\\
\{S^1 \cup S^2 \mid S^1 \in S_1^*, S^2 \in S_2^*\} & ; \chi(t_1\sigma) = \chi(t_2\sigma) = 2,\\
\{S^1 \cup S^2 \mid S^1 \in S_1, S^2 \in S_2\} & ; \chi(t_1\sigma) = \chi(t_2\sigma) = 1,\\
\{S^1 \cup S^2 \mid S^1 \in S_1^*, S^2 \in S_2\} & ; \chi(t_1\sigma) = 1 \wedge \chi(t_2\sigma) = 2,\\
\{S^1 \cup S^2 \mid S^1 \in S_1, S^2 \in S_2^*\} & ; \chi(t_1\sigma) = 2 \wedge \chi(t_2\sigma) = 1.
\end{cases}$$

Then $\mathcal{A}(\sigma\theta) \subset \mathcal{A}(\sigma)\backslash S_1 \backslash S_2 \cup T$. $\square$

Proof: Note the symmetry of the last two cases. The first two cases are already proven. Let $x \in V, \emptyset \neq occ(\sigma, x)$ as usual.

[10] This is only a syntactical property.

We start with $\chi(t_1\sigma) = \chi(t_2\sigma) = 1$, i.e., $t_1\sigma$ and $t_2\sigma$ have only single variable occurrences. Let us assume $x \in var(t_1\sigma)$ and $x \in rgv(\theta)$ ($\Rightarrow x \in codom(\theta)$). The other cases are similar to the preceding proofs. Consider the set of equations E_θ in solved form generated during the unification process. Let $e = e', \bar{e} = \bar{e}' \in E_\theta$ different equations. Then all variables occur only once and imply:

- $var(e) \subset var(t_i\sigma), var(e') \subset var(t_{3-i}\sigma) \Rightarrow var(e) \cap var(e') = \emptyset$, and
- $\chi(t_1\sigma) = \chi(t_2\sigma) = 1 \Rightarrow var(e) \cap (var(\bar{e}) \cup var(\bar{e}')) = \emptyset$.

Thus $\forall \bar{x} \in codom(\theta)\exists_1 y \in dom(\theta) : \bar{x} \in var(y\theta) \overset{x\equiv\bar{x}}{\Rightarrow} occ(\theta, x) = \{x, y\}, y \in var(t_2\sigma)$.

$$\Rightarrow occ(\sigma\theta, x) = \underbrace{occ(\sigma, x)}_{\in S_1} \cup \underbrace{occ(\sigma, y)}_{\in S_2} \in T.$$

Now consider $\chi(t_1\sigma) = 1, \chi(t_2\sigma) = 2$. $x \notin var(t_1\sigma) \cup var(t_2\sigma)$ or $x \in dom(\theta)$ are as usual. Therefore assume $x \in rgv(\theta)$ ($\Rightarrow x \in codom(\theta)$). Consider again E_θ. Let $e = e', \bar{e} = \bar{e}' \in E_\theta$ different equations, w.l.o.g., $var(e), var(\bar{e}) \subset var(t_1\sigma)$. Then

- $var(e) \cap var(e') = \emptyset$, and
- $\chi(t_1\sigma) = 1 \Rightarrow var(e) \cap var(\bar{e}) = \emptyset$.

Case 1: $x \in var(t_1\sigma)$:

Thus $\forall \bar{x} \in var(t_1\sigma) \cap codom(\theta)\exists_1 y \in var(t_2\sigma) \cap dom(\theta) : \bar{x} \in var(y\theta)$
$\overset{x\equiv\bar{x}}{\Rightarrow} \{x, y\} \subset occ(\theta, x) \subset var(t_1\sigma) \cup \{y\}$

$$\Rightarrow occ(\sigma\theta, x) = \underbrace{\bigcup_{z \in occ(\theta,x)\cap var(t_1\sigma)} \underbrace{occ(\sigma, z)}_{\in S_1}}_{\in S_1^*} \cup \underbrace{occ(\sigma, y)}_{\in S_2} \in T.$$

Case 2: $x \in var(t_2\sigma)$:

Thus $\forall \bar{x} \in var(t_2\sigma) \cap codom(\theta)\ \nexists(\bar{x} \neq)y \in var(t_2\sigma) \cap dom(\theta) : \bar{x} \in var(y\theta)$.
$\overset{x\equiv\bar{x}}{\Rightarrow} \{x\} \subsetneq occ(\theta, x) \subset var(t_1\sigma) \cup \{x\}$

$$\Rightarrow occ(\sigma\theta, x) = \underbrace{occ(\sigma, x)}_{\in S_2} \cup \underbrace{\bigcup_{z \in occ(\theta,x)\cap var(t_1\sigma)} \underbrace{occ(\sigma, z)}_{\in S_1}}_{\in S_1^*} \in T.$$ $\square$

Remark: Note that in the theorem the assumption $var(t_1\sigma) \cap var(t_2\sigma) = \emptyset$ is needed, due to $t_1 = A, t_2 = B, \sigma = \{A/f(X, Z), B/f(Y, X)\}$. $\theta = \{Y \to X, Z \to X\}$ is a valid unifier and a counter example. We would get $\mathcal{A}(\sigma) = \{\{A, X\}, \{A, Z\}, \{B, X\}, \{B, Y\}\}$ and $\mathcal{A}(\sigma\theta) \subset \{\{A, B, X, Y\}, \{A, B, X\}, \{A, B, Y, Z\}, \{A, B, X, Z\}\}$. But $occ(\sigma\theta, X) = \{A, B, X, Y, Z\} \in \mathcal{A}(\sigma\theta)$. $\square$

Lemma 2: Let t_i, σ, S_i as in Theorem 1. A sufficient condition for the independence of the terms $t_1\sigma$ and $t_2\sigma$ is: $S_1 \cap S_2 = \emptyset$. $\square$

Proof: $x \in var(t_1\sigma) \cap var(t_2\sigma) \overset{La.1}{\Leftrightarrow} occ(\sigma, x) \cap var(t_1) \neq \emptyset \neq occ(\sigma, x) \cap var(t_2) \Leftrightarrow S_1 \cap S_2 \neq \emptyset$. $\square$

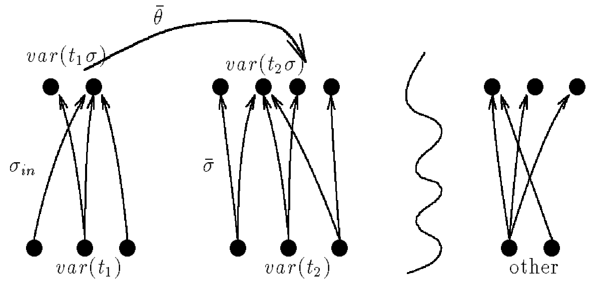

Figure 3: Influences of Concrete (Match-)Bindings to the Occurrence Sets

3.4 Matching

The abstract execution bases on three operations. In particular, procedure exit is optimized in this subsection. Procedure exit is useless, as long as infinite abstractions are allowed. But below, the domain of abstractions is restricted and enforces this operation to extend such abstractions to further variables. This refinement bases on the observation that the unification at this point is in fact a match (Figure 3) and enables the saving of closures.

Let us start with an explanation of the theorem, whose assumptions are adjusted to the operation procedure exit. Because a procedure entry precedes each procedure exit, there exists an in-substitution σ_{in}, just before the predicate call happens. The successful procedure entry yields a unifier θ unifying the head of the clause and the respective literal of the goal. Now, the locality of the computation within the clause implies that the modifications (σ) of the call-substitution $\sigma_{in}\theta$ affect only variables of $rgv_{W_2}(\sigma_{in}\theta)$; W_2 corresponds to the variables of the clause. The evaluated exit-substitution is denoted by $\bar{\sigma}$.

The theorem shows that the out-substitution $\sigma_{in}\bar{\theta}$ can be approximated, even if only the in-substitution σ_{in} accesses the bindings of the goal variables.

Theorem 4: Let $W_1, W_2 \subset V$, $t_i \in T_{FS}(W_i)$, $\sigma_{in}, \sigma, \bar{\sigma}, \bar{\theta} \in Sub$, $\theta := mgu(t_1\sigma_{in}, t_2\sigma_{in})$, $dom(\sigma) \subset rgv_{W_2}(\sigma_{in}\theta)$, $\bar{\sigma} := \sigma_{in}\theta\sigma$, $\bar{\theta} \in Sub$ with $t_1\sigma_{in}\bar{\theta} = t_2\bar{\sigma}$, $S_1 = \{S \in \mathcal{A}(\sigma_{in}) \mid S \cap var(t_1) \neq \emptyset\}$, $S_2 = \{S \in \mathcal{A}(\bar{\sigma}) \mid S \cap var(t_2) \neq \emptyset\}$, and

$$T = \begin{cases} \emptyset & ; \chi(t_2\bar{\sigma}) = 0, \\ \{S^1 \cup S^2 \mid S^1 \in S_1, S^2 \in S_2\} & ; \chi(t_2\bar{\sigma}) = 1, \\ \{S^1 \cup S^2 \mid S^1 \in S_1^*, S^2 \in S_2\} & ; \chi(t_2\bar{\sigma}) = 2. \end{cases}$$

Then $\mathcal{A}(\sigma_{in}\bar{\theta}) \subset \mathcal{A}(\sigma_{in}) \backslash S_1 \cup T$. □

Proof: Let $x \in V$, $occ(\sigma_{in}, x) \neq \emptyset$:

$\chi(t_2\bar{\sigma}) = 0$ is similar to the general case.

$\chi(t_2\bar{\sigma}) = 1$: Consider three cases:

1. $x \notin var(t_1\sigma_{in}) \cup var(t_2\bar{\sigma})$.
   $\Rightarrow \begin{array}{l} occ(\sigma_{in}\bar{\theta}, x) = occ(\sigma_{in}, x) \\ occ(\sigma_{in}, x) \cap var(t_1) = \emptyset \end{array} \Rightarrow occ(\sigma_{in}\bar{\theta}, x) \in \mathcal{A}(\sigma_{in}) \backslash S_1.$
2. $x \in var(t_2\bar{\sigma}) \Rightarrow occ(\bar{\sigma}, x) \cap var(t_2) \neq \emptyset$.
   $\chi(t_2\bar{\sigma}) = 1 \Rightarrow occ(\bar{\theta}, x) = \{x, y\}, y \in var(t_1\sigma_{in})$ ($\Rightarrow occ(\sigma_{in}, y) \cap var(t_1) \neq \emptyset$)
   $\Rightarrow occ(\sigma_{in}\bar{\theta}, x) = \underbrace{occ(\sigma_{in}, y)}_{\in S_1} \cup \underbrace{occ(\bar{\sigma}, x)}_{\in S_2} \in T$.
3. $x \in var(t_1\sigma_{in}) \backslash var(t_2\bar{\sigma}) \Rightarrow occ(\sigma_{in}, x) \cap var(t_1) \neq \emptyset \Rightarrow x \in dom(\bar{\theta}) \Rightarrow occ(\sigma_{in}\bar{\theta}, x) = \emptyset$.

$\chi(t_2\bar{\sigma}) = 2$: is similar to $\chi(t_2\bar{\sigma}) = 1$, except, that case 2) yields $dom(\bar{\theta}, x) = \{x\} \cup Y, \emptyset \neq Y \subset var(t_1\sigma_{in})$ and $occ(\sigma_{in}\bar{\theta}, x) = \underbrace{\bigcup_{y \in Y} \underbrace{occ(\sigma_{in}, y)}_{\in S_1}}_{\in S_1^*} \cup \underbrace{occ(\bar{\sigma}, x)}_{\in S_2} \in T$. □

Now, we define the domain to store aliasing. Note that this definition demands that abstractions are built over an arbitrary subset W of the set V of all variables. In Section 5 we will see how the theorems are adjusted to the actual abstraction. We introduce the **Notation:** Let $W \subset V$: $\mathcal{A}(\sigma) \mid_W := \{S \cap W \neq \emptyset \mid S \in \mathcal{A}(\sigma)\}$

Definition 7: Aliasing Abstraction
Let $W \subset V$. The aliasing abstraction is given by the 5-tuple $\bar{\mathcal{A}} = (ASub^A_{W,\perp}, \sqsubset, \sqcup, \alpha^A_W, \gamma^A_W)$ with[11]

- $\mathrm{ASub}^A_{W,\perp} = \mathcal{P}(\mathcal{P}(W)) \cup \perp \subset \mathcal{P}(\mathcal{P}(V)) \cup \perp$,
- for $\kappa \in ASub^A_{W,\perp}$: $\gamma^A_W(\kappa) = \{\sigma \in Sub \mid \mathcal{A}(\sigma) \mid_W \subset \kappa\}$,
- for $\Sigma \subset Sub$: $\alpha^A_W(\Sigma) = \bigcup_{\sigma \in \Sigma} \mathcal{A}(\sigma) \mid_W$,
- $\sqsubset, \sqcup$ as in the general definition. □

Remark: The proof of Lemma 2 shows an equivalence. But dealing with abstractions, only one implication — fortunately the important one — remains ($S_i = \{S \in \kappa \mid S \cap var(t_i) \neq \emptyset\}$):

$$\kappa \in ASub^A_{W,\perp}, \sigma \in \gamma^A_W(\kappa) : S_1 \cap S_2 = \emptyset \Rightarrow var(t_1\sigma) \cap var(t_2\sigma) = \emptyset. \quad □$$

4 Instantiation

This section introduces the augmentation of the abstract domain to take advantage of the refinements. To retain clarity, the minimal extension is chosen. This 'orthogonal' domain

[11] W may be finite.

stores the degree of instantiation of the variables via pattern. According to the aliasing part, three instantiations are distinguished. First, there exist unbound variables denoted by *Var*. The next class of terms — including variables — contains terms with at most single variable occurrences (*Linear*). This class denotes ground terms as well. Due to the possibility of inferring groundness from the aliasing part no further distinction is needed. For worst case approximations the pattern *Any*, denoting any term, is present.

Definition 8: i-pattern, i-state, Instantiation
a) *i-pattern* are elements of Pat={Var,Linear,Any} being ordered: Var$\sqsubset$Linear$\sqsubset$Any. The *least upper bound* of the i-pattern i_1,i_2 is the least element i — according to $\sqsubset$ — with $i_1 \sqsubset i$, $i_2 \sqsubset i$, denoted by $i_1 \sqcup i_2$.
b) An *i-state* $\iota \in Inst$ is any mapping $\iota : V \longrightarrow$ Pat with *domain* $dom(\iota)$.
c) The *instantiation* $\zeta : T_{FS}(V) \longrightarrow$Pat is defined

$$\zeta(t) = \begin{cases} Var & ; t \in V, \\ Linear & ; \chi(t) \leq 1, t \notin V, \\ Any & ; \text{otherwise.} \end{cases}$$ □

Definition 9: Instantiation Abstraction
Let $W \subset V$. The instantiation abstraction is given by the 5-tuple $\bar{\mathcal{A}} = (ASub^I_{W,\perp}, \sqsubset, \sqcup, \alpha^I_W, \gamma^I_W)$ with

- $\text{ASub}^I_{W,\perp} = (W \longrightarrow Pat) \cup \perp \subset Inst \cup \perp$,
- for $\iota \in ASub^I_{W,\perp} : \gamma^I_W(\iota) = \{\sigma \in Sub \mid \forall x \in W : \zeta(x\sigma) \sqsubset x\iota\}$,
- for $\Sigma \subset Sub : \alpha^I_W(\Sigma) = \iota$ with $x\iota = \bigsqcup_{\sigma \in \Sigma} \zeta(x\sigma)$ $(x \in W)$,
- $\sqsubset, \sqcup$ as in the general definition. □

5 Final Abstraction

After this preparation, our abstract domain is presented which captures all the mentioned refinements. From now on, we assume one valid abstraction of the query — specified by the user — and consider the computation of succeeding abstractions. This section contains three parts. The first subsection describes the general case of unification performed both during procedure entry as well as during explicit unification. The two lemmata focus on the behaviour of the pattern during unification. The following theorem summarizes the soundness of the operations in terms of our final abstraction. Note that the finite representation is dealt with implicitly by the 'arbitrary' set W. The next part discusses the matching case.

To conclude this step towards the execution model of the abstract interpreter, the approximate computability of the abstract variable multiplicity is demonstrated.

Definition 10: Aliasing Abstraction with Instantiation
Let $W \subset V$. The aliasing abstraction with instantiation is given by the 5-tuple $\bar{\mathcal{A}} = (ASub^W_\perp, \sqsubset, \sqcup, \alpha_W, \gamma_W)$ with

- $\text{ASub}^W_\perp = ((W \longrightarrow Pat) \times \mathcal{P}(\mathcal{P}(W))) \cup \perp \subset (Inst \times \mathcal{P}(\mathcal{P}(V))) \cup \perp$,
- for $\beta \equiv (\iota, \kappa) \in ASub^W_\perp$: $\gamma_W(\beta) = \{\sigma \in Sub \mid \forall x \in W : \zeta(x\sigma) \sqsubset x\iota \wedge \mathcal{A}(\sigma)\mid_W \subset \kappa\}$,
- for $\Sigma \subset Sub$: $\alpha_W(\Sigma) = (\iota, \bigcup_{\sigma \in \Sigma} \mathcal{A}(\sigma)\mid_W)$ with $x\iota = \bigsqcup_{\sigma \in \Sigma} \zeta(x\sigma)$ $(x \in W)$,
- $\sqsubset, \sqcup$ as in the general definition. □

We list several statements in the following
Remark:

- $\sqsubset$ and $\sqcup$ are used, both to express the order of instantiations, as well as to express the order of the three different abstractions. But there should be no confusion, due to the context.
- Note the strong coupling of W and $dom(\iota)$ $(= dom(\beta))$. Therefore, the script W is unnecessary. The previous definitions need only slight modifications to remove the dependence from W, i.e., only one abstraction and concretization function remain. But for clarity, we use the additional identifier W.
- $\beta, \beta' \in ASub^W_\perp : \beta \sqcup \beta' = (\bar{\iota}, \kappa \cup \kappa')$ with $x\bar{\iota} = x\iota \sqcup x\iota'$ $(dom(\bar{\iota}) = W)$
- $\gamma_W(\beta) = \gamma^I_W(\iota) \cap \gamma^A_W(\kappa)$, $\alpha_W(\Sigma) = (\alpha^I_W(\Sigma), \alpha^A_W(\Sigma))$
- We identify the several bottom elements. Therefore $ASub_\perp = \bigcup_{W \subset V} ASub^W_\perp = (Inst \times \mathcal{P}(\mathcal{P}(V))) \cup \perp$.
- $\perp \sqcup \beta = \beta \sqcup \perp = \beta$ $(\beta \in ASub_\perp)$
- The application of $\beta \equiv (\iota, \kappa) \in ASub^W_\perp$ to some term $t \in T_{FS}(W)$ is defined by:

$$t\beta = \begin{cases} t\iota & ; t \in V, \\ Linear \sqcup \bigsqcup_{y \in var(t)} y\iota & ; t \notin V \ \wedge (\chi(t) \leq 1 \vee (\chi(t) = 2 \wedge \forall x \in V_2 : \kappa_{\{x\}} = \emptyset)) \\ & \quad \wedge \forall x \neq y \in var(t) : \kappa_{\{x,y\}} = \emptyset, \\ Any & ; \text{otherwise.} \end{cases}$$

  Refer to the second case: The absence of sharing is known if 1) no sharing is introduced by the known part of the term t ($\chi(t) \leq 1$), 2) if the contained variables are independent ($\kappa_{\{x,y\}} = \emptyset$), and 3) if the variables do not possess shared parts ($y\iota$). But if the known part shares some variables ($\chi(t) = 2$), then these variables (V_2) must be ground ($\kappa_{\{x\}} = \emptyset$). Using the abstract variable multiplicity, this condition can be rewritten: $t \notin V \wedge \chi^{\mathcal{A}}((t, \beta) < 2$ ($\bigsqcup_{y \in var(t)} y\iota$ becomes superfluous). □

5.1 Unification

We start with the determination of the pattern, which approximates the result of unifying two terms. Of course, this result would be expected in Section 4, but, due to the hidden reference to the aliasing part of the abstraction via the abstract variable multiplicity, this result is moved to this place.

Lemma 3: Let $W \subset V, t_1, t_2 \in T_{FS}(W), \beta \in ASub_{\perp}^{W}, \sigma \in \gamma_W(\beta), \theta = mgu(t_1\sigma, t_2\sigma)$. Define ρ

$$\rho := \begin{cases} t_1\beta \sqcup t_2\beta & ; t_1\beta = Var \vee t_2\beta = Var, \\ \text{Linear} & ; t_1\beta \neq Var \neq t_2\beta \wedge (\chi^{\mathcal{A}}(t_1,\beta) = 0 \vee \chi^{\mathcal{A}}(t_2,\beta) = 0 \\ & \quad \vee(\chi^{\mathcal{A}}(t_1,\beta) = \chi^{\mathcal{A}}(t_2,\beta) = 1 \wedge var(t_1\sigma) \cap var(t_2\sigma) = \emptyset)), \\ Any & ; \text{ otherwise.} \end{cases}$$

Then a) $\forall x \in var(t_1) \cup var(t_2) : \zeta(x\sigma\theta) \sqsubset \rho$, □

b) $t_i\beta = Var : \forall x \in var(t_{3-i}) : \zeta(x\sigma\theta) \sqsubset x\beta$.

Proof: a) $\sigma \in \gamma_W(\beta) \Rightarrow \forall x \in W : \zeta(x\sigma) \sqsubset x\beta \Rightarrow \forall t \in T_{FS}(W) : \zeta(t\sigma) \sqsubset t\beta$. $t_1\sigma\theta = t_2\sigma\theta \Rightarrow \forall x \in var(t_1) \cup var(t_2) : \zeta(t_1\sigma\theta) = \zeta(t_2\sigma\theta) \sqsupset \zeta(x\sigma\theta)$. It remains to show $\zeta(t_1\sigma\theta) \sqsubset \rho$.

Case 1: $t_1\beta = Var \vee t_2\beta = Var$. In this case only one binding appears. This clear situation yields: $\zeta(t_1\sigma\theta) \sqsubset \zeta(t_1\sigma) \sqcup \zeta(t_2\sigma) \sqsubset t_1\beta \sqcup t_2\beta = \rho$.

Case 2: $t_1\beta \neq Var \neq t_2\beta$.

1. $\chi^{\mathcal{A}}(t_1,\beta) = 0 \vee \chi^{\mathcal{A}}(t_2,\beta) = 0$. The groundness of one term yields the groundness of the other term. Therefore $\zeta(t_1\sigma\theta) \sqsubset \text{Linear} = \rho$.
2. $\chi^{\mathcal{A}}(t_1,\beta) = \chi^{\mathcal{A}}(t_2,\beta) = 1 \wedge var(t_1\sigma) \cap var(t_2\sigma) = \emptyset$. This particular case prevents the introduction of any sharing[12]. Therefore $\zeta(t_1\sigma\theta) \sqsubset \text{Linear} = \rho$.

Case 3: Any is always safe.

b) $t_i\beta = Var \Rightarrow \forall x \in var(t_{3-i}) : \zeta(x\sigma\theta) = \zeta(x\sigma) \sqsubset x\beta$. □

According to the presence of aliasing, variables outside of $var(t_1) \cup var(t_2)$ can be affected. To manage these side effects the approximate propagation of the derived pattern becomes necessary.

Example 6: a) Assume X, Y and $Y, f(Z)$ to be unified with the initial abstraction $(\{X/Var, Y/Var, Z/Var\}, \{\{X\}, \{Y\}, \{Z\}\})$. After processing the first unification we have: $(\{X/Var, Y/Var, Z/Var\}, \{\{X, Y\}, \{Z\}\})$. Now, the previous lemma would lead to the wrong abstraction: $(\{X/Var, Y/Linear, Z/Var\}, \{\{X, Y, Z\}\})$, because X is bound to the term $f(Z)$. Weakening of the pattern of X make the abstraction safe.
b) In order to see a worse case suppose that $X, f(Y, Z))$ and Y, Z are unified with the same initial abstraction. While processing the unification we would get

$$(\{X/Linear, Y/Var, Z/Var\}, \{\{X, Y\}, \{X, Z\}\}),$$

being unchanged by the second step. But X is bound to $f(Z, Z)$ enforcing the pattern *Any*; the multiple presence of X in the sharing gives some hint about this weakening. □

Lemma 4: (Pattern Propagation) Let $t_i, W, V, \beta, \sigma, \theta, \rho$ as in the previous lemma. Let

[12] The soundness considerations are quite similar to the respective case in the proof of Theorem 3.

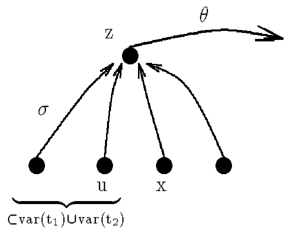

Figure 4: Pattern Propagation

$\bar{S} = \bigcup_{S \in \kappa, S \cap (var(t_1) \cup var(t_2)) \neq \emptyset} S$. Define $\hat{\imath} : W \longrightarrow Pat$ by

$$x\hat{\imath} := \begin{cases} x\beta & ; x \notin \bar{S} \vee (x \in var(t_i) \wedge t_{3-i}\beta = Var), \\ \rho \sqcup x\beta & ; x \in \bar{S} \wedge t_1\beta \neq Var \neq t_2\beta \wedge (x \in var(t_1) \cup var(t_2) \vee x\beta = Var \\ & \qquad \vee (x\beta = \text{Linear} \wedge \mid \kappa_{\{x\}} \mid = 1)), \\ \text{Any} & ; \text{otherwise.} \end{cases}$$

Then $\forall x \in W : \zeta(x\sigma\theta) \sqsubset x\hat{\imath}$. □

$\mid \kappa_{\{x\}} \mid = 1$ refers to the notion of reduction and expresses that the abstraction κ must contain exactly one occurrence set with the variable x.

Proof: The soundness of the first and the last cases is obvious. The first case affects variables, which can not be changed during unification or are correct by the previous lemma, while the worst case approximation holds always. Consider $x \in \bar{S}$ in the second case. $x \in var(t_1) \cup var(t_2)$ is again correct by the previous lemma. Let $x \notin var(t_1) \cup var(t_2)$ (Figure 4):

a) $x\beta = Var \quad \Rightarrow x\iota = Var \Rightarrow x\sigma \in V$.
$x \in \bar{S} \quad \Rightarrow \exists (x\sigma =) z \in V, u \in var(t_1) \cup var(t_2) : \{u, x\} \subset occ(\sigma, z)$
$\Rightarrow \zeta(x\sigma\theta) = \zeta(z\theta) \sqsubset \zeta(u\sigma\theta) \sqsubset \rho \sqsubset \rho \sqcup x\beta$.

b) $x\beta = \text{Linear} \wedge \mid \kappa_{\{x\}} \mid = 1$. Consider two cases:

1. $\chi(x\sigma) = 0 :\Rightarrow \chi(x\sigma\theta) = 0 \Rightarrow \zeta(x\sigma\theta) = \text{Linear} \sqsubset \rho \sqcup x\beta$.
2. $var(x\sigma) = \{z_1, \ldots, z_n\} \Rightarrow x \in occ(\sigma, z_i)$.

$$\begin{aligned} \mid \kappa_{\{x\}} \mid = 1 &\Rightarrow occ(\sigma, z_i) \cap W = occ(\sigma, z_1) \cap W = \{x_1, \ldots, x_m\} \ni x \\ &\Rightarrow occ(\sigma, z_i) \supset \{x_1, \ldots, x_m\} \Rightarrow \{z_1, \ldots, z_n\} \subset var(x_i\sigma). \end{aligned}$$

Therefore, each operation performed during unification considers all variables within $x\sigma$.

$$\chi(x\sigma) = 1 \Rightarrow \zeta(x\sigma\theta) \sqsubset \zeta(x\sigma) \sqcup \bigsqcup_{\bar{x} \in \{x_1 \ldots x_m\} \cap (var(t_1) \cup var(t_2))} \zeta(\bar{x}\sigma\theta) \sqsubset x\beta \sqcup \rho.$$ □

The next statement summarizes all the previous results and extends them to the final single step unification operation. Particularly, the domain restriction W is added. The theorem consists of three parts. First the instantiation is considered followed by the description of the aliasing computation. The third part does groundness propagation and gives the final abstraction.

Theorem 5: Abstract Unification
Let $t_i, W, \beta, \sigma, \theta, \hat{\imath}$ as in the previous lemma.
a) Define the abstract unification i-unify$_W : T^2_{FS}(W) \times ASub^W_\perp \backslash \perp \longrightarrow Inst$ dealing with instantiations:

$$\text{i-unify}_W(t_1, t_2, \beta) = \hat{\imath}.$$

Then $\sigma\theta \in \gamma^I_W(\text{i-unify}_W(t_1, t_2, \beta))$.
b) The abstract unification a-unify$_W : T^2_{FS}(W) \times ASub^W_\perp \backslash \perp \longrightarrow \mathcal{P}(\mathcal{P}(W))$ dealing with aliasing is defined by:

$$\text{a-unify}_W(t_1, t_2, \beta) = \kappa \backslash S_1 \backslash S_2 \cup T.$$

with $S_i = \{S \in \kappa \mid S \cap var(t_i) \neq \emptyset\}$ $(i = 1, 2)$ and T given by:

- if $S_1 \cap S_2 = \emptyset$ $(\Rightarrow var(t_1\sigma) \cap var(t_2\sigma) = \emptyset)$

$$T = \begin{cases} \emptyset & ; \chi^{\mathcal{A}}(t_1, \beta) = 0 \vee \chi^{\mathcal{A}}(t_2, \beta) = 0, \\ \{S^1 \cup S^2 \mid S^1 \in S_1, S^2 \in S_2\} & ; t_1\beta = Var \vee t_2\beta = Var \vee \\ & ; \chi^{\mathcal{A}}(t_1, \beta) = \chi^{\mathcal{A}}(t_2, \beta) = 1, \\ \{S^1 \cup S^2 \mid S^1 \in S_1^*, S^2 \in S_2\} & ; \chi^{\mathcal{A}}(t_1, \beta) = 1 \wedge \chi^{\mathcal{A}}(t_2, \beta) = 2, \\ \{S^1 \cup S^2 \mid S^1 \in S_1, S^2 \in S_2^*\} & ; \chi^{\mathcal{A}}(t_1, \beta) = 2 \wedge \chi^{\mathcal{A}}(t_2, \beta) = 1, \\ \{S^1 \cup S^2 \mid S^1 \in S_1^*, S^2 \in S_2^*\} & ; \chi^{\mathcal{A}}(t_1, \beta) = \chi^{\mathcal{A}}(t_2, \beta) = 2. \end{cases}$$

- if $S_1 \cap S_2 \neq \emptyset$ $(\Leftarrow var(t_1\sigma) \cap var(t_2\sigma) \neq \emptyset)$

$$T = \begin{cases} \emptyset & ; \chi^{\mathcal{A}}(t_1, \beta) = 0 \vee \chi^{\mathcal{A}}(t_2, \beta) = 0, \\ \{S^1 \cup S^2 \mid S^1 \in S_1, S^2 \in S_2\} & ; t_1\beta = Var \vee t_2\beta = Var, \\ \{S^1 \cup S^2 \mid S^1 \in S_1^*, S^2 \in S_2^*\} & ; \text{otherwise.} \end{cases}$$

Then $\sigma\theta \in \gamma^{\mathcal{A}}_W(\text{a-unify}_W(t_1, t_2, \beta))$.
c) (Groundness Propagation) The abstract unification unify$^{\mathcal{A}}_W : T^2_{FS}(W) \times ASub^W_\perp \backslash \perp \longrightarrow ASub^W_\perp \backslash \perp$ is defined by:

$$\text{unify}^{\mathcal{A}}_W(t_1, t_2, \beta) = (\bar{\imath}, \bar{\kappa})$$

with $\hat{\imath} = \text{i-unify}_W(t_1, t_2, \beta)$, $\bar{\kappa} = \text{a-unify}_W(t_1, t_2, \beta)$, and $x\bar{\imath} = \begin{cases} \text{Linear} & ; \bar{\kappa}_{\{x\}} = \emptyset, \\ x\hat{\imath} & ; \text{otherwise.} \end{cases}$

Then $\sigma\theta \in \gamma_W(\text{unify}^{\mathcal{A}}_W(t_1, t_2, \beta))$. □

Proof: a) $\sigma \in \gamma_W(\beta) \overset{(La.4)}{\Rightarrow} \forall x \in W : \zeta(x\sigma\theta) \sqsubset \text{i-unify}_W(t_1, t_2, \beta)(x) \Rightarrow \sigma\theta \in \gamma^I_W(\text{i-unify}_W(t_1, t_2, \beta))$.
b) $\{S^1 \cup S^2 \mid S^1 \in S_1, S^2 \in S_2\} \subset \begin{matrix} \{S^1 \cup S^2 \mid S^1 \in S_1^*, S^2 \in S_2\} \\ \{S^1 \cup S^2 \mid S^1 \in S_1, S^2 \in S_2^*\} \end{matrix} \subset \{S^1 \cup S^2 \mid S^1 \in S_1^*, S^2 \in S_2^*\}$. Therefore, it suffices to consider $S_1 \cap S_2 = \emptyset$. $\sigma \in \gamma_W(\beta) \Rightarrow \mathcal{A}(\sigma) \mid_W \subset \kappa$.

Set $S_i' = \{S \in \mathcal{A}(\sigma) \mid_W \mid S \cap var(t_i) \neq \emptyset\}$, T': replace in T S_i by S_i',
$S_i'' = \{S \in \mathcal{A}(\sigma) \mid S \cap var(t_i) \neq \emptyset\}$, T'': replace in T S_i by S_i''.
Due to the above mentioned inclusion property of the different T, which also holds for T'', and $\chi(t\sigma) \leq \chi^{\mathcal{A}}(t, \beta)$, Theorems 2 and 3 yield: $\mathcal{A}(\sigma\theta) \subset \mathcal{A}(\sigma) \backslash S_1'' \backslash S_2'' \cup T''$. Therefore

$$\begin{aligned} \mathcal{A}(\sigma\theta) \mid_W & \subset (\mathcal{A}(\sigma) \backslash S_1'' \backslash S_2'' \cup T'') \mid_W & = \mathcal{A}(\sigma) \mid_W \backslash S_1'' \mid_W \backslash S_2'' \mid_W \cup T'' \mid_W \\ & = \mathcal{A}(\sigma) \mid_W \backslash S_1' \backslash S_2' \cup T' & = (\mathcal{A}(\sigma) \mid_W \backslash S_1') \cap (\mathcal{A}(\sigma) \mid_W \backslash S_2') \cup T' \\ & \subset \kappa \backslash S_1 \backslash S_2 \cup T & = \text{a-unify}_W(t_1, t_2, \beta) \end{aligned}$$

because of

$$\mathcal{A}(\sigma) \mid_W \backslash S_i' = \{S \in \mathcal{A}(\sigma) \mid_W \mid S \cap var(t_i) = \emptyset\} \subset \{S \in \kappa \mid S \cap var(t_i) = \emptyset\} = \kappa \backslash S_i.$$

Finally $\sigma\theta \in \gamma^{\mathcal{A}}_W(\text{a-unify}_W(t_1, t_2, \beta))$.
c) $\sigma\theta \in \gamma_W((\text{i-unify}_W(t_1, t_2, \beta), \text{a-unify}_W(t_1, t_2, \beta)))$.
$\bar{\kappa}_{\{x\}} = \emptyset \Rightarrow var(x\sigma\theta) = \emptyset \Rightarrow \zeta(x\sigma\theta) = \text{Linear} \Rightarrow \sigma\theta \in \gamma_W(\text{unify}^{\mathcal{A}}_W(t_1, t_2, \beta))$. □

5.2 Matching

The matching case refers to the procedure exit operation and, therefore, the assumptions are adjusted to this case. The proofs proceed similar to the previous subsection: The next lemma prepares the instantiation determination and the theorem summarizes the results.

Lemma 5: Let $W_1, W_2 \subset V, W_1 \cap W_2 = \emptyset, t_i \in T_{FS}(W_i), \beta_i \in ASub^{W_i}_\perp, \sigma_i \in \gamma_{W_i}(\beta_i), \theta := mgu(t_1\sigma_1, t_2\sigma_1), \sigma \in Sub$ with $dom(\sigma) \subset rgv_{W_2}(\sigma_1\theta)$. σ_2 should also satisfy $\sigma_2 := \sigma_1\theta\sigma$. Finally, let $\bar{\theta} \in Sub$ with $t_1\sigma_1\bar{\theta} = t_2\sigma_2$. Set $\rho := t_2\beta_2$.
Then $\forall x \in var(t_1) : \zeta(x\sigma_1\bar{\theta}) \sqsubset \rho$.
b) Let $\bar{S} = \bigcup_{S \in \kappa_1, S \cap var(t_1) \neq \emptyset} S$. Define $\hat{\imath} : W_1 \longrightarrow Pat$ by

$$x\hat{\imath} := \begin{cases} x\beta_1 & ; x \notin \bar{S}, \\ \rho \sqcup x\beta_1 & ; x \in \bar{S} \wedge (x \in var(t_1) \vee x\beta_1 = Var \vee (x\beta_1 = \text{Linear} \wedge \mid \kappa_{1\,\{x\}} \mid = 1)), \\ \text{Any} & ; \text{otherwise.} \end{cases}$$

Then $\forall x \in W_1 : \zeta(x\sigma_1\bar{\theta}) \sqsubset x\hat{\imath}$. □

Proof: a) $\forall x \in var(t_1) : \zeta(x\sigma_1\bar{\theta}) \sqsubset \zeta(t_1\sigma_1\bar{\theta}) = \zeta(t_2\sigma_2) \sqsubset t_2\beta_2 = \rho$.
b) This part proceeds similar to the proof of Lemma 4. We focus on the non-trivial second case. i) $x \in var(t_1)$ is correct by a).
ii)

$$\begin{aligned} x\beta_1 = Var & \Rightarrow x\sigma_1 \in V. \\ x \in \bar{S} & \Rightarrow \exists (x\sigma_1 =) z \in V, u \in var(t_1) : \{u, x\} \subset occ(\sigma_1, z) \\ & \Rightarrow \zeta(x\sigma_1\bar{\theta}) = \zeta(z\bar{\theta}) \sqsubset \zeta(u\sigma_1\bar{\theta}) \sqsubset \rho \sqsubset \rho \sqcup x\beta_1. \end{aligned}$$

iii) $x\beta_1 = \text{Linear} \wedge \mid \kappa_{1\,\{x\}} \mid = 1$. Consider two cases:

1. $\chi(x\sigma_1) = 0 :\Rightarrow \zeta(x\sigma_1\bar{\theta}) = \text{Linear} \sqsubset \rho \sqcup x\beta_1$.

2. $var(x\sigma_1) = \{z_1, \ldots, z_n\} \Rightarrow x \in occ(\sigma_1, z_i)$.

$$\begin{aligned} \mid \kappa_{1\{x\}} \mid = 1 & \Rightarrow occ(\sigma_1, z_i) \cap W_1 = occ(\sigma_1, z_1) \cap W_1 = \{x_1, \ldots, x_m\} \ni x \\ & \Rightarrow occ(\sigma_1, z_i) \supset \{x_1, \ldots, x_m\} \Rightarrow \{z_1, \ldots, z_n\} \subset var(x_i\sigma_1). \end{aligned}$$

Therefore, each operation performed during unification considers all variables within $x\sigma_1$.
$\chi(x\sigma_1) = 1 \Rightarrow \zeta(x\sigma_1\bar{\theta}) \sqsubset \zeta(x\sigma_1) \sqcup \bigsqcup_{\bar{x}\in\{x_1\ldots x_m\}\cap(var(t_1)\cup var(t_2))} \zeta(\bar{x}\sigma_1\bar{\theta}) \sqsubset x\beta_1 \sqcup \rho$. □

The next theorem has the same outline as Theorem 5. It consists of three parts dealing with the instantiations and aliasing computation concluded by the description of the final operation. Note the simplification of T in case b) compared with the theorem 4, which is caused by the assumption $W_1 \cap W_2 = \emptyset$.

Theorem 6: Let $W_i, t_i, \beta_i, \sigma_i, \bar{\theta}, \hat{\imath}$ as in the previous lemma.
a) Define the abstract matching i-match$_{W_1,W_2}$: $T_{FS}(W_1) \times T_{FS}(W_2) \times (ASub_\perp^{W_1} \backslash \perp) \times (ASub_\perp^{W_2} \backslash \perp) \to Inst$ dealing with instantiations:

$$\text{i-match}_{W_1,W_2}(t_1, t_2, \beta_1, \beta_2) := \hat{\imath}.$$

Then $\sigma_1\bar{\theta} \in \gamma_{W_1}^I(\text{i-match}_{W_1,W_2}(t_1, t_2, \beta_1, \beta_2))$.
b) The abstract matching a-match$_{W_1,W_2}$: $T_{FS}(W_1)\times T_{FS}(W_2)\times(ASub_\perp^{W_1}\backslash \perp)\times(ASub_\perp^{W_2}\backslash \perp) \longrightarrow \mathcal{P}(\mathcal{P}(W_1))$ dealing with aliasing is defined by:

$$\text{a-match}_{W_1,W_2}(t_1, t_2, \beta_1, \beta_2) = \kappa_1 \backslash S_1 \cup T.$$

with $S_i = \{S \in \kappa_i \mid S \cap var(t_i) \neq \emptyset\}$ $(i = 1, 2)$ and T given by:

$$T = \begin{cases} \emptyset & ; \chi^{\mathcal{A}}(t_2, \beta_2) = 0, \\ S_1 & ; \chi^{\mathcal{A}}(t_2, \beta_2) = 1, \\ S_1^* & ; \chi^{\mathcal{A}}(t_2, \beta_2) = 2. \end{cases}$$

Then $\sigma_1\bar{\theta} \in \gamma_{W_1}^{\mathcal{A}}(\text{a-match}_{W_1,W_2}(t_1, t_2, \beta_1, \beta_2))$.
c) The abstract matching match$^{\mathcal{A}}_{W_1,W_2}$: $T_{FS}(W_1)\times T_{FS}(W_2)\times(ASub_\perp^{W_1}\backslash \perp)\times(ASub_\perp^{W_2}\backslash \perp) \longrightarrow ASub_\perp^{W_1} \backslash \perp$ is defined by:

$$\text{match}^{\mathcal{A}}_{W_1,W_2}(t_1, t_2, \beta_1, \beta_2) = (\bar{\imath}, \bar{\kappa}).$$

with $\hat{\imath} = \text{i-match}_{W_1,W_2}(t_1, t_2, \beta_1, \beta_2)$, $\bar{\kappa} = \text{a-match}_{W_1,W_2}(t_1, t_2, \beta_1, \beta_2)$, and
$x\bar{\imath} = \begin{cases} \text{Linear} & ; \bar{\kappa}_{\{x\}} = \emptyset, \\ x\hat{\imath} & ; \text{otherwise.} \end{cases}$
Then $\sigma_1\bar{\theta} \in \gamma_{W_1}(\text{match}^{\mathcal{A}}_{W_1,W_2}(t_1, t_2, \beta_1, \beta_2))$. □

Proof: a) $\sigma_i \in \gamma_{W_i}(\beta_i) \overset{(La.5)}{\Rightarrow} \forall x \in W_1 : \zeta(x\sigma_1\bar{\theta}) \sqsubset \text{i-match}_{W_1,W_2}(t_1, t_2, \beta_1, \beta_2)(x) \Rightarrow \sigma_1\bar{\theta} \in \gamma_{W_1}^I(\text{i-match}_{W_1,W_2}(t_1, t_2, \beta_1, \beta_2))$.
b)$\sigma_i \in \gamma_{W_i}(\beta_i) \Rightarrow \mathcal{A}(\sigma_i) \mid_{W_i} \subset \kappa_i$. Set

$$S_i' = \{S \in \mathcal{A}(\sigma_i) \mid_{W_i} \mid S \cap var(t_i) \neq \emptyset\}, T' = \begin{cases} \emptyset & ; \chi^{\mathcal{A}}(t_2, \beta_2) = 0, \\ \{S^1 \cup S^2 \mid S^1 \in S_1', S^2 \in S_2'\} & ; \chi^{\mathcal{A}}(t_2, \beta_2) = 1, \\ \{S^1 \cup S^2 \mid S^1 \in S'^*_1, S^2 \in S_2'\} & ; \chi^{\mathcal{A}}(t_2, \beta_2) = 2, \end{cases}$$

$S_i'' = \{S \in \mathcal{A}(\sigma_i) \mid S \cap var(t_i) \neq \emptyset\}$, T'': replace in T' S_i' by S_i''.
Due to $\emptyset \subset \{S^1 \cup S^2 \mid S^1 \in S_1'', S^2 \in S_2''\} \subset \{S^1 \cup S^2 \mid S^1 \in S''^*_1, S^2 \in S_2''\}$ and $\chi(t_2\sigma_2) \leq \chi^{\mathcal{A}}(t_2, \beta_2)$, Theorem 4 yields: $\mathcal{A}(\sigma_1\bar{\theta}) \subset \mathcal{A}(\sigma_1)\backslash S_1'' \cup T''$. Therefore

$$\begin{aligned} \mathcal{A}(\sigma_1\bar{\theta}) \mid_{W_1} &\subset (\mathcal{A}(\sigma_1)\backslash S_1'' \cup T'') \mid_{W_1} && = \mathcal{A}(\sigma_1) \mid_{W_1} \backslash S_1'' \mid_{W_1} \cup T'' \mid_{W_1} \\ &= \mathcal{A}(\sigma_1) \mid_{W_1} \backslash S_1' \cup T' \mid_{W_1} && \subset \kappa_1 \backslash S_1 \cup T = \text{a-match}_{W_1,W_2}(t_1, t_2, \beta_1, \beta_2) \end{aligned}$$

and $\sigma_1\bar{\theta} \in \gamma_{W_1}^{\mathcal{A}}(\text{a-match}_{W_1,W_2}(t_1, t_2, \beta_1, \beta_2))$.
c) $\sigma_1\bar{\theta} \in \gamma_W((\text{i-match}_{W_1,W_2}(t_1, t_2, \beta_1, \beta_2), \text{a-match}_{W_1,W_2}(t_1, t_2, \beta_1, \beta_2)))$.
$\bar{\kappa}_{\{x\}} = \emptyset \Rightarrow var(x\sigma_1\bar{\theta}) = \emptyset \Rightarrow \zeta(x\sigma_1\bar{\theta}) = \text{Linear} \Rightarrow \sigma_1\bar{\theta} \in \gamma_{W_1}(\text{match}^{\mathcal{A}}_{W_1,W_2}(t_1, t_2, \beta_1, \beta_2))$. □

5.3 Evaluation of $\chi^{\mathcal{A}}$

In order to complete the previous theorems, the computation of the abstract variable multiplicity is shown. Of course, this has again an approximate behaviour. The negation of the second condition of the next lemma in combination with the negation of the first case gives a sufficient condition for $\chi^{\mathcal{A}}(t, \beta) = 1$.

Note that this suffices for implementation purposes because worse results are imprecise, but remain correct, i.e., one has to check the first case: If it is satisfied, return the value 0. Otherwise check the second one, and if this holds, return 2, else 1.

Lemma 6: Let $W \subset V, t \in T_{FS}(W), \beta \in ASub_\perp^W$. Let V_1 be the variables of t with single occurrences, and $V_2 := var(t)\backslash V_1$.

1. $\chi^{\mathcal{A}}(t, \beta) = 0 \Leftrightarrow \forall S \in \kappa : S \cap var(t) = \emptyset$,

2. $\chi^{\mathcal{A}}(t, \beta) = 2 \Rightarrow \exists S \in \kappa : S \cap V_2 \neq \emptyset \vee \exists x \in V_1 : x\beta = \text{Any} \vee \exists (x \neq) y \in V_1 : \kappa_{\{x,y\}} \neq \emptyset$.

□

Proof: Note $occ(\sigma, x) \cap var(t) = occ(\sigma, x) \cap W \cap var(t)$.

1. $$\begin{aligned} &\chi^{\mathcal{A}}(t, \beta) = 0 \Leftrightarrow \forall \sigma \in \gamma_W(\beta) : \chi(t\sigma) = 0 \Leftrightarrow \forall \sigma \in \gamma_W(\beta) : var(t\sigma) = \emptyset \\ \Leftrightarrow\ &\forall \sigma \in \gamma_W(\beta), x \in V : occ(\sigma, x) \cap var(t) = \emptyset \Leftrightarrow \forall S \in \kappa : S \cap var(t) = \emptyset. \end{aligned}$$

2. $\chi^{\mathcal{A}}(t, \beta) = 2. \Leftrightarrow \exists \sigma \in \gamma_W(\beta) : \chi(t\sigma) = 2$

Case 1: $\chi(t) \leq 1$.

$$\begin{aligned} &\chi^{\mathcal{A}}(t, \beta) = 2 \\ \Leftrightarrow\ &\exists \sigma \in \gamma_W(\beta), x \in var(t) : \chi(x\sigma) = 2 \vee \exists (x \neq) y \in var(t), z \in V : \\ &\qquad \{x, y\} \subset occ(\sigma, z) \\ \Rightarrow\ &\exists \sigma \in \gamma_W(\beta), x \in var(t) : x\iota = Any \vee \exists (x \neq) y \in var(t) : \mathcal{A}(\sigma)_{\{x,y\}} \neq \emptyset \\ \Rightarrow\ &\exists \sigma \in \gamma_W(\beta), x \in var(t) : x\iota = Any \vee \exists (x \neq) y \in var(t) : (\mathcal{A}(\sigma) \mid_W)_{\{x,y\}} \neq \emptyset \\ \Leftrightarrow\ &\exists x \in var(t) : x\beta = Any \vee \exists (x \neq) y \in var(t) : \kappa_{\{x,y\}} \neq \emptyset \end{aligned}$$

Case 2: $\chi(t) = 2$. Define two auxiliary substitutions θ_1, θ_2:

$$\theta_i(x) := \begin{cases} x & ; x \in V_i, \\ c & ; \text{otherwise,} \end{cases}$$

for some arbitrary constant c. Note $\chi(t\theta_1) \leq 1$ and $\chi(t\theta_2) = 2$.

$$
\begin{array}{ll}
 & \chi^{\mathcal{A}}(t,\beta) = 2 \\
\Leftrightarrow & \exists \sigma \in \gamma_W(\beta) : \chi(t\theta_2\sigma) = 2 \vee \chi(t\theta_1\sigma) = 2 \\
\Rightarrow & \chi^{\mathcal{A}}(t\theta_2,\beta) \neq 0 \vee \chi^{\mathcal{A}}(t\theta_1,\beta) = 2 \\
\Rightarrow & \exists S \in \kappa : S \cap \underbrace{var(t\theta_2)}_{=V_2} \neq \emptyset \vee \exists x \in \overbrace{var(t\theta_1)}^{=V_1} : x\beta = Any \vee \\
 & \qquad \exists (x \neq) y \in V_1 : \kappa_{\{x,y\}} \neq \emptyset.
\end{array}
$$

□

6 Abstract Execution

The previous results are combined to specify the execution strategy of our abstract interpreter for logic programs[13]. The signature of logic programs consists of *functor symbols* $f \in FS$ and *predicate symbols* $p \in PS$. *Literals* $l \in Lit$ are constructed similar to terms; their top symbol is always a predicate symbol, while the nested parts are normal terms. A *clause*$\in Clause$ consists of the *head*$\in Lit$ and the *body*$\in Lit^*$, the former is one literal, and the latter a (possible empty) sequence of literals. Clauses are denoted via head:–body. The clauses form the program. They are grouped to *predicates* according to the common predicate symbol of the head. Therefore, we consider the logic program as the mapping P:$PS \longrightarrow Clause^*$. The *goal*$\in Lit^*$ of the program is treated as the clause with an empty head.

The domain of *var* is extended to literals and clauses. Similar the domain of the concretization function γ_W is extended to terms via:

$$\gamma_W(t,\beta) = \{t\sigma \mid \sigma \in \gamma_W(\beta)\}.$$

The first part of the next definition describes the unification operation (Unify$^{\mathcal{A}}$) in its final version. The specification considers failing unification and performs *preunification*. Particularly, preunification is useful to increase the precision of the derivation. The decomposition of the known parts of structures (constants) leads finally to $t_1 \in V$ or $t_2 \in V$. The unification of $t_1, t_2 \in T_{FS}(V)\backslash FS\backslash V$, i.e., $t_i = f_i(t_{i,1} \ldots t_{i,n_i})$ is successful, only if $f_1 = f_2$ and $n_1 = n_2$. Then decomposition applies, which reduces the complexity of the terms. If constants (f()=) f are decomposed, no unification at the abstract level is done.

The second part describes the final match operation (Match$^{\mathcal{A}}$).

Definition 11: Unify$^{\mathcal{A}}$, Match$^{\mathcal{A}}$
Let $W, W_1, W_2 \subset V, \beta_i \in ASub_\perp, dom(\iota_1) \cap dom(\iota_2) = \emptyset, p(t_1^i \ldots t_n^i) \in Lit, var(p(t_1^i \ldots t_n^i)) \subset dom(\iota_i)$.
a) Unification:
i) We define the auxiliary function[14] unify-set$_W^{\mathcal{A}}$: $\mathcal{P}(T_{FS}^2(W)) \times ASub_\perp^W \longrightarrow ASub_\perp^W$

13Prolog's notion is assumed.

14In order to prevent the annotation of the rules with the different conditions, the application in textual order is assumed.

$$
\begin{array}{ll}
\text{unify-set}_W^{\mathcal{A}}(\emptyset,\beta) & = \beta \\
\text{unify-set}_W^{\mathcal{A}}(\mathrm{T},\perp) & = \perp \\
\text{unify-set}_W^{\mathcal{A}}(\{(f(t_1 \ldots t_n), f(t'_1 \ldots t'_n))\} \cup T,\beta) & = \text{unify-set}_W^{\mathcal{A}}(\{(t_i,t'_i) \mid 1 \leq i \leq n\} \cup T,\beta) \\
\text{unify-set}_W^{\mathcal{A}}(\{(f(t_1 \ldots t_n), g(t'_1 \ldots t'_m))\} \cup T,\beta) & = \perp \ (f \neq g \vee n \neq m) \\
\text{unify-set}_W^{\mathcal{A}}(\{(t,t')\} \cup T,\beta) & = \text{unify-set}_W^{\mathcal{A}}(\mathrm{T},\text{unify}_W^{\mathcal{A}}(t,t',\beta))
\end{array}
$$

ii) Unify$^{\mathcal{A}}$: $Lit^2 \times ASub_\perp^2 \longrightarrow ASub_\perp$

Unify$^{\mathcal{A}}(p(t_1^1 \ldots t_n^1), p(t_1^2 \ldots t_n^2), \beta_1, \beta_2)$ =unify-set$_R^{\mathcal{A}}(\{(t_i^1, t_i^2) \mid 1 \leq i \leq n\}, \bar{\beta}) \mid_{dom(\iota_1)}$

with $R = dom(\iota_1) \cup dom(\iota_2), \bar{\beta} = (\bar{\iota}, \kappa_1 \cup \kappa_2)$ and $x\bar{\iota} = \begin{cases} x\iota_1 & , x \in dom(\iota_1) \\ x\iota_2 & , x \in dom(\iota_2). \end{cases}$

b) Matching:
i) We define the auxiliary function match-set$_{W_1,W_2}^{\mathcal{A}}$: $\mathcal{P}(T_{FS}(W_1) \times T_{FS}(W_2)) \times ASub_\perp^{W_1} \times ASub_\perp^{W_2} \longrightarrow ASub_\perp^{W_1}$

$$
\begin{array}{ll}
\text{match-set}_{W_1,W_2}^{\mathcal{A}}(\emptyset,\beta_1,\beta_2) & = \beta_1 \\
\text{match-set}_{W_1,W_2}^{\mathcal{A}}(\mathrm{T},\perp,\beta_2) & = \perp \\
\text{match-set}_{W_1,W_2}^{\mathcal{A}}(\mathrm{T},\beta_1,\perp) & = \perp \\
\text{match-set}_{W_1,W_2}^{\mathcal{A}}(\{(f(t_1 \ldots t_n), f(t'_1 \ldots t'_n))\} \cup T,\beta_1,\beta_2) & = \\
\qquad \text{match-set}_{W_1,W_2}^{\mathcal{A}}(\{(t_i,t'_i) \mid 1 \leq i \leq n\} \cup T,\beta_1,\beta_2) & \\
\text{match-set}_{W_1,W_2}^{\mathcal{A}}(\{(f(t_1 \ldots t_n), g(t'_1 \ldots t'_m))\} \cup T,\beta_1,\beta_2) & = \perp \ (f \neq g \vee n \neq m) \\
\text{match-set}_{W_1,W_2}^{\mathcal{A}}(\{(t,t')\} \cup T,\beta_1,\beta_2) & = \\
\qquad \text{match-set}_{W_1,W_2}^{\mathcal{A}}(\mathrm{T},\text{match}_{W_1,W_2}^{\mathcal{A}}(t,t',\beta_1,\beta_2),\beta_2) &
\end{array}
$$

ii) $Match^{\mathcal{A}}$: $Lit^2 \times ASub_\perp^2 \longrightarrow ASub_\perp$

$Match^{\mathcal{A}}(p(t_1^1 \ldots t_n^1), p(t_1^2 \ldots t_n^2), \beta_1, \beta_2) =$
match-set$_{dom(\iota_1),dom(\iota_2)}^{\mathcal{A}}(\{(t_i^1, t_i^2) \mid 1 \leq i \leq n\}, \beta_1, \beta_2)$.□

Remark: The soundness of unify-set$_W^{\mathcal{A}}$, match-set$_{W_1,W_2}^{\mathcal{A}}$, and Match$^{\mathcal{A}}$ follows immediately by induction. The correctness of Unify$^{\mathcal{A}}$ is shown in the next theorem. □

[Bruynooghe91] named three operations, which have to be specified to apply his framework to a particular analysis. Further, he mentioned three sufficient local conditions for the global soundness. The next definition formalizes this three operations followed by the theorem that considers the satisfaction of the conditions. The notation within the definition and the theorem are related to Figure 5, which depicts the active part of the whole AND-OR-graph. A predicate call is shown: β_{in} denotes the abstraction before procedure entry, β_{out} after procedure exit. The OR-node shows the alternatives and the second level sketches the local computation of one alternative at the abstract level[15].

Definition 12: Procedure Entry, Procedure Exit, Explicit Unification
a) procedure_entry:$Clause \times Lit \times ASub_\perp \longrightarrow ASub_\perp$:

procedure_entry(Cl,B$_i$,β_{in}):=Unify$^{\mathcal{A}}$(head(Cl),B$_i$,β_{init},β_{in}) ($= \beta_{in}^j$)

with $\beta_{init} = (\iota_{init}, \kappa_{init}), x\iota_{init} = Var \ (x \in var(Cl))$, and $\kappa_{init} = \{\{x\} \mid x \in var(Cl)\}$ (head(Cl)=H$_j$).

b) procedure_exit:Lit$\times Lit^* \times ASub_\perp \times ASub_\perp^* \longrightarrow ASub_\perp$:

procedure_exit(B$_i$,$\mathcal{H}$,β_{in},$\boldsymbol{\mathcal{B}}$):=$\bigsqcup_{1 \leq j \leq m} \beta_j$ ($= \beta_{out}$)

15The subgraphs below D_i are pruned.

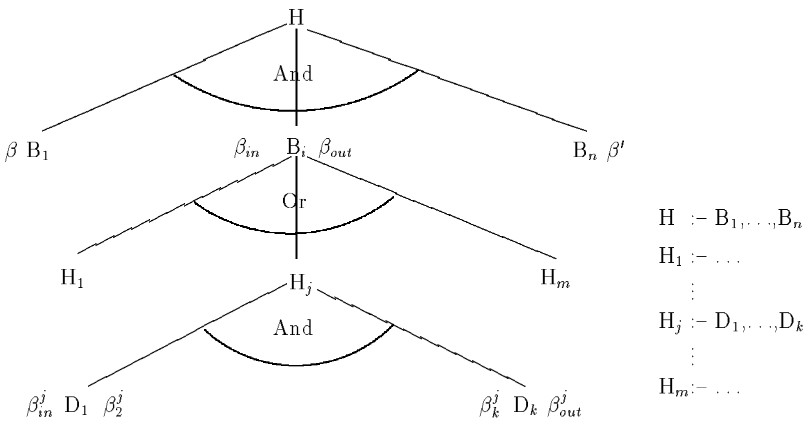

Figure 5: Part of an Abstract AND-OR-Graph

with $\mathcal{H} = (H_1 \ldots H_m)$, $\mathcal{B} = (\beta^1_{out} \ldots \beta^m_{out})$, and $\beta_j = \text{Match}^{\mathcal{A}}(B_i, H_j, \beta_{in}, \beta^j_{out})$.
c) explicit_unify:$T^2_{FS}(V) \times ASub_\perp \longrightarrow ASub_\perp$:

$$\text{explicit\_unify}(t, t', \beta_{in}) := \text{unify-set}^{\mathcal{A}}_{dom(\iota_{in})}(\{t, t'\}, \beta_{in}) \; (= \beta_{out}). \qquad \square$$

Theorem 7: Let $H : -B_1, \ldots, B_n.$ and $H_j : -D_1, \ldots, D_k.$ renamed clauses, $W = var(H : -B_1, \ldots, B_n.)$, $U = var(H_j : -D_1, \ldots, D_k.)$, $W \cap U = \emptyset$, $\beta_{in}, \beta_{out} \in ASub^W_\perp$, β^j_{in}, $\beta^j_{out} \in ASub^U_\perp$.
a) Procedure Entry: Let $\sigma_{in} \in \gamma_W(\beta_{in})$, $B_i\sigma_{in}$ and H$_j$ be unifyable with $\theta = mgu(B_i\sigma_{in}, H_j)$.
Then $(H_j : -D_1, \ldots, D_k.)\theta \in \gamma_U((H_j : -D_1, \ldots, D_k.)\beta^j_{in})$.
b) Let $1 \leq j \leq m$, $\sigma_j \in \gamma_U(\beta^j_{out})$, $\sigma_{in} \in \gamma_W(\beta_{in})$, $\theta_j \in Sub$ with

- $B_i\sigma_{in}\theta_j = H_j\sigma_j$,
- $(dom(\theta_j) \cup codom(\theta_j)) \cap var((H : -B_1, \ldots, B_n.)\sigma_{in}) \subset var(B_i\sigma_{in})$

Then $(H : -B_1, \ldots, B_n.)\sigma_{in}\theta_j \in \gamma_W((H : -B_1, \ldots, B_n.)\beta_{out})$.
c)Builtin: Let $\sigma_{in} \in \gamma_W(\beta_{in})$ and the execution of $B_i\sigma_{in}$ yields $\theta \in Sub$.
Then $(H : -B_1, \ldots, B_n.)\sigma_{in}\theta \in \gamma_W((H : -B_1, \ldots, B_n.)\beta_{out})$. $\square$

Proof: a) H$_j$ introduces new variables. Therefore $\forall x \in U : \zeta(x\sigma_{in}) = Var$, $occ(\sigma_{in}, x) = \{x\}$ and $\sigma_{in} \in \gamma_U(\beta_{init})$. Because of $\forall x \in V \backslash U : occ(\sigma_{in}, x) \cap U = \emptyset$

$$\begin{array}{ll}
\Rightarrow & \sigma_{in} \in \gamma_{U \cup W}(\bar\beta) \; (\bar\beta \text{ in Unify}^{\mathcal{A}}) \\
\Rightarrow & \sigma_{in}\theta \in \gamma_{U \cup W}(\text{unify-set}^{\mathcal{A}}_{U \cup W}(\ldots, \bar\beta)) \\
\Rightarrow & \sigma_{in}\theta \in \gamma_U(\text{unify-set}^{\mathcal{A}}_{U \cup W}(\ldots, \bar\beta) \mid_U) = \gamma_U(\beta^j_{in}) \\
\Rightarrow & (H_j : -D_1, \ldots, D_k.)\sigma_{in}\theta \in \gamma_U((H_j : -D_1, \ldots, D_k.)\beta^j_{in}) \\
\overset{dom(\sigma_{in}) \cap U = \emptyset}{\Rightarrow} & (H_j : -D_1, \ldots, D_k.)\theta \in \gamma_U((H_j : -D_1, \ldots, D_k.)\beta^j_{in})
\end{array}$$

b) Let j fixed, but arbitrary, $\sigma_{in} \in \gamma_W(\beta_{in})$, $\sigma_j \in \gamma_U(\beta^j_{out})$.

$$\begin{array}{ll}
\Rightarrow & \sigma_{in}\theta_j \in \gamma_W(\text{match-set}^{\mathcal{A}}_{W,U}(\ldots, \beta_{in}, \beta^j_{out})) = \gamma_W(\beta_j) \subset \gamma_W(\beta_{out}) \\
\Rightarrow & (H : -B_1, \ldots, B_n.)\sigma_{in}\theta_j \in \gamma_W((H : -B_1, \ldots, B_n.)\beta_{out}).
\end{array}$$

c) Examples: Explicit Unification and Test $<$.
i) Explicit Unification: $\sigma_{in} \in \gamma_W(\beta_{in})$, $\theta = mgu(t\sigma_{in}, t'\sigma_{in})$
$\Rightarrow \sigma_{in}\theta \in \gamma_W(\text{unify-set}^{\mathcal{A}}_W(\{(t, t')\}, \beta_{in})) = \gamma_W(\beta_{out}) \Rightarrow (H : -B_1, \ldots, B_n.)\sigma_{in}\theta \in \gamma_W((H : -B_1, \ldots, B_n.)\beta_{out})$.
ii) Test $t < t'$: $\sigma_{in} \in \gamma_W(\beta_{in})$, $\beta_{in} = (\iota, \kappa)$, $\chi(t\sigma_{in}) = \chi(t'\sigma_{in}) = 0$, $\bar S = \{S \in \kappa \mid S \cap (var(t) \cup var(t')) \neq \emptyset\}$.

$$\text{Set } \bar\beta = \begin{cases} \perp & ; \exists x \in var(t_1) \cup var(t_2) : x\iota = Var \\ (\bar\iota, \kappa \backslash \bar S) & ; \text{otherwise with } x\bar\iota = \begin{cases} \text{Linear} & ; (\kappa \backslash \bar S)_{\{x\}} = \emptyset \\ x\iota & ; \text{otherwise} \end{cases} \end{cases}$$

$\Rightarrow \sigma_{in} \in \gamma_W(\bar\beta)$. Because of $\theta = id, \bar\beta = \beta_{out} :\Rightarrow (H : -B_1, \ldots, B_n.)\sigma_{in}\theta \in \gamma_W((H : -B_1, \ldots, B_n.)\beta_{out})$.
Note that the predefined test operator is only applicable, if all arguments are ground. Therefore, we can safely adjust the abstraction to this restriction. $\square$

We conclude this section with the specification of the algorithm in a simplified form. It differs from the approach in [Bruynooghe91] only in the fixpoint computation. Our one is less optimized because the computation of the fixpoint is done by repeated execution of the whole program. In [Bruynooghe91] only the computation of the approximate predicates is repeated. Approximation is needed when at an ancestor node the same call (same predicate, same (finite) abstraction) appears. In this case, the exit abstraction is estimated by a valid default abstraction, i.e., the bottom element $\perp$. Returning to the first call the delivered abstraction is compared with the stored one (initially $\perp$). If these abstractions differ, recomputation with updated default substitution is initiated[16]. The default abstractions are stored in *memo tables*, which are mappings memo$\in Mem = Clause \times ASub_\perp \longrightarrow ASub_\perp$. They map each pair of clause and (in-)abstraction to the current (out-)abstraction. Furthermore, the memo table can be used to prune recomputation of the predicate at 'independent' positions. In this point our algorithm is the better one. The pruning mechanism uses a set *processed*$\in Pro = \mathcal{P}(Clause \times ASub_\perp)$, which stores all clauses with their abstractions, already computed within the current iteration.

Let the program P fixed. *fixpoint* iterates simply until the fixpoint is reached. *analyze* checks, whether the clause is already computed. If this test succeeds, pruning is done. Otherwise, the sequence of literals is processed. For each literal, according to the type, *process_predicate* or *process_builtin* is called. The former performs the procedure entry operation, processes the literals of each alternative by recursive calls of analyze, and does finally the procedure exit operation. The latter distinguishes between the different builtins. Due to the previous theorem, only the explicit unification and the test operator $<$ are sketched.

[16] In our approach this necessary recomputation occurs in the next iteration.

init-call$_P$:$Clause \longrightarrow ASub_\perp \times Mem$
init-call$_P$(G):=
  let memo(cl,β)=$\perp$ $\forall cl \in Clause, \beta \in ASub_\perp$ **in**
  fixpoint$_P$(G,memo)

fixpoint$_P$:$Clause \times Mem \longrightarrow Mem$
fixpoint$_P$(G,memo):=
  let $\beta = (\iota, \{\{x\} \mid x \in var(G)\})$ **and** $x\iota = Var$ $(x \in var(G))$ **and**
    (β',processed,memo$'$)=analyze$_P$(G,β,$\emptyset$,memo) **in**
  if memo=memo$'$ **then** memo
  else fixpoint$_P$(G,memo$'$)

analyze$_P$:$Clause \times ASub_\perp \times Pro \times Mem \longrightarrow ASub_\perp \times Pro \times Mem$
analyze$_P$(C,β_0,processed,memo$_0$):=
  if (C,β_0) $\in$processed **then** (memo$_0$(C,β_0),processed,memo$_0$)
  else let C$\equiv$H:-B$_1, \ldots,$B$_n$ **and** processed$_0$=processed$\cup\{(C, \beta_0)\}$ **and**
      (β_i,processed$_i$,memo$_i$)=
        if builtin(B$_i$) **then** (process_builtin(B$_i$,β_{i-1}),processed$_{i-1}$,memo$_{i-1}$)
        else process_predicate$_P$(B$_i$,β_{i-1},processed$_{i-1}$,memo$_{i-1}$) **in**
    (β_n,processed$_n$,memo$_n$[(C,β_0)/memo$_n$(C,β_0)$\sqcup\beta_n$])

process_predicate$_P$:$Lit \times ASub_\perp \times Pro \times Mem \longrightarrow ASub_\perp \times Pro \times Mem$
process_predicate$_P$(q($t_1, \ldots, t_n$),β_{in},processed$_0$,memo$_0$):=
  let $\{cl_1, \ldots, cl_m\}$ $\equiv$P(q) **and** β^i_{in}=procedure_entry(cl$_i$,q($t_1, \ldots, t_n$),β_{in}) **and**
    (β^i_{out},processed$_i$,memo$_i$)=
      if β^i_{in} =$\perp$ **then** ($\perp$,processed$_{i-1}$,memo$_{i-1}$)
      else analyze$_P$(cl$_i$,β^i_{in},processed$_{i-1}$,memo$_{i-1}$) **in**
  (procedure_exit(q($t_1 \ldots t_n$),(head(cl$_1$)...head(cl$_m$)),β_{in},($\beta^1_{out} \ldots \beta^m_{out}$)),processed$_m$,
memo$_m$)

process_builtin:$Lit \times ASub_\perp \longrightarrow ASub_\perp$
process_builtin(L,β_{in}):=
  if L$\equiv$ $t = t'$ **then** Unify$^{\mathcal{A}}(t, t', \beta_{in})$
  else if L$\equiv$ $t < t'$ **then**
    let $\bar{S} = \{S \in \kappa_{in} \mid S \cap var(L) \neq \emptyset$ **and**

$$\beta_{out} = \begin{cases} \perp & ; \exists x \in var(L) : x\beta_{in} = Var \\ (\bar{\iota}, \kappa_{in} \backslash \bar{S}) & ; \text{otherwise with } x\bar{\iota} = \begin{cases} \text{Linear} & ; (\kappa_{in} \backslash \bar{S})_{\{x\}} = \emptyset \\ x\beta_{in} & ; \text{otherwise} \end{cases} \end{cases} \quad \textbf{in}$$

    β_{out}
  else ...

Figure 6: Outline of the Analyzer

7 Optimization Opportunities

To get clarity about our abstract analysis approach, we have not incorporated too much optimization. Although it is not necessary, one could extend the instantiation pattern with a ground pattern. This can at least speed up the abstract execution. In the same manner other properties could be stored here. But we must be aware of the correct pattern propagation.

Additionally, the explicit unification could be processed as in the concrete computation. Because the bindings are local, they do not influence any global values. This allows often a better decomposition.

As mentioned before, the simple iteration over the whole program is optimizable. But retaining the memo table for pruning advantages, a mechanism to determine, whether some clause must be reevaluated, is needed. Dynamic dependency graphs solve this problem ([Charlier et al.91]).

8 Conclusion

In this paper we have developted an accurate approach for aliasing and groundness analysis based on abstract interpretation. The aliasing domain was first proposed in [Jacobs,Langen89]. The extensions are enabled by the introduction of i-states, which captures the refinements suggested in [Muthukumar,Hermenegildo91] and [Codish et al.91] for similar analysis purposes.

The main goal of this paper was the *safe* derivation of the necessary operations and its *safe* embedding in the framework of [Bruynooghe91]. Our correctness considerations were done in four steps. Starting with infinite abstract representations of sets of substitutions, we proved the soundness of the unification rules, both for the general case as well as for the refinements, to our knowledge normally skipped. The second step considered the necessary extension of the aliasing domain in order to capture these refinements. The next steps added the finite representation and, finally, the safe embedding.

Quite recently, [Jacobs,Langen92] supplied the proof of the aliasing part of the general case, similar to our one, and showed furthermore the satisfaction of the algebraic properties of idempotence and commutativity. Due to these results, they suggested the pure (query independent) bottom-up strategy, a process, they called *condensing*, and stated that this will not lead to a loss of accuracy compared with the (query dependent) top-down driven bottom-up strategy. Particularly, the commutativity justifies this behaviour, because groundness propagation can be delayed performing first the bottom-up analysis (only once) and, afterwards, the (cheaper) groundness propagation using the query(queries).

However, they briefly mentioned similar refinements within their implementation. But, these extensions may not satisfy the algebraic properties. Particularly, the lack of commutativity[17] — shown in the final example — would led to worse results.

Example 7: a) Suppose we would like to unify $(A, f(X,Y))$ and (C, B) with the abstraction $(\{A/Linear, B/Var, C/Var, X/Var, Y/Var\}, \{\{A,B\},\{C\},\{X\},\{Y\}\})$. This

[17] if one follows our proposal

abstraction may result from a preceding abstract unification of $(A, f(B,2))$ and the initial abstraction $(\{A/Var, B/Var, C/Var, X/Var, Y/Var\}, \{\{A\},\{B\},\{C\},\{X\},\{Y\}\})$. Now, we observe the different pattern of A.

1. unify from left to right:
   (a) $\beta_1^l = (\{A/Linear, B/Linear, C/Var, X/Var, Y/Var\}, \{\{A,B,X\},\{A,B,Y\},\{C\}\})$
   (b) $\beta_2^l = (\{A/\mathbf{Any}, \ldots\}, \{\{A,B,C,X\},\{A,B,C,Y\}\})$
2. unify from right to left:
   (a) $\beta_1^r = (\{A/Linear, B/Var, C/Var, X/Var, Y/Var\}, \{\{A,B,C\},\{X\},\{Y\}\})$
   (b) $\beta_2^r = (\{A/\mathbf{Linear}, \ldots\}, \{\{A,B,C,X\},\{A,B,C,Y\}\})$

Although the aliasing part remains equal, the varying i-states will influence the following unifications and, therefore, the following aliasing parts.
b) Now, we contradict idempotence. Consider (X,Y) with the abstraction $(\{A/Var, B/Var, X/Var, Y/Any\}, \{\{A,Y\},\{B,Y\},\{Y\}\})$. A previous unification $(Y, f(A,A,B))$ would yield this abstraction from the initial one. The first unification leads to

$$\beta_1 = (\{\ldots, X/Any, Y/Any\}, \{\{A,X,Y\},\{B,X,Y\}\}).$$

But the second unification gives
$\beta_2 = (\{\ldots, X/Any, Y/Any\}, \{\{A,X,Y\},\{\mathbf{A},\mathbf{B},\mathbf{X},\mathbf{Y}\},\{B,X,Y\}\}) \neq \beta_1$. □
We have implemented our proposal — using the language platform C — and have compared its run times with earlier versions doing a more conservative static analysis. We recognized that the novel management of aliasing did not introduced notable overhead. Although the closure operation and set manipulation by bitmaps is not cheap in general, usually these operations are done with small sets (with small entries).

References

[Bruynooghe91] M. Bruynooghe, A Practical Framework for the Abstract Interpretation of Logic Programs, *Journal of Logic Programming*, Vol. 10, 1991

[Charlier et al.91] B. Le Charlier, K. Musumbu, and P. Van Hentenryck, A Generic Abstract Interpretation Algorithm and its Complexity Analysis, *Logic Programming: Proceedings of the Eighth International Conference*, MIT Press, June 1991

[Codish et al.91] M. Codish, D. Dams, and Eyal Yardeni, Derivation and Safety of an Abstract Unification Algorithm for Groundness and Aliasing Analysis, *Logic Programming: Proceedings of the Eighth International Conference*, MIT Press, June 1991

[Debray,Warren88] S. K. Debray and D. S. Warren, Automatic Mode Inference for Logic Programs, *Journal of Logic Programming*, Vol. 5, 1988

[Debray89] S. K. Debray, Static Inference of Modes and Data Dependencies in Logic Programs, *North American Conference on Logic Programming*, MIT Press, 1989

[DeGroot84] D. DeGroot, Restricted AND-Parallelism, *International Conference on Fifth Generation Computer Systems*, North Holland and OHMSHA, Ltd, 1984

[Englebert et al.] V. Englebert, B. Le Charlier, D. Roland, and P. Van Hentenryck, Generic Abstract Interpretation Algorithms for Prolog: Two Optimization Techniques and Their Experimental Evaluation, PLILP 92, *Lecture Notes in Computer Science*, 631, Springer Verlag, 1992

[Jacobs,Langen88] D. Jacobs and A. Langen, Compilation of Logic Programs for Restricted AND-Parallelism, *European Symposium on Programming*, 1988

[Jacobs,Langen89] D. Jacobs and A. Langen, Accurate and Efficient Approximation of Variable Aliasing in Logic Programs, *North American Conference on Logic Programming*, MIT Press, 1989

[Jacobs,Langen92] D. Jacobs and A. Langen, Static Analysis of Logic Programs for Independent And Parallelism, *Journal of Logic Programming*, Vol. 13, 1992

[Martelli,Montanari82] A. Martelli and U. Montanari, An Efficient Unification Algorithm, *ACM Trans. on Programming Languages and Systems*, Vol 4(2),1982

[Mellish81] C. S. Mellish, Automatic Generation of Mode Declarations for Prolog Programs, Technical Report 163, University of Edinburgh, Department of AI, 1981

[Mellish86] C. S. Mellish, Abstract Interpretation of Prolog Programs, 3^{rd} ICLP, *Lecture Notes in Computer Science*, 225, Springer Verlag, 1986

[Muthukumar,Hermenegildo90a] K. Muthukumar and M. Hermenegildo, The CDG, UDG, and MEL Methods for Automatic Compile-time Parallelization of Logic Programs for Independent AND-parallelism, *Logic Programming: Proceedings of the Seventh International Conference*, MIT Press, June 1990

[Muthukumar,Hermenegildo90b] K. Muthukumar and M. Hermenegildo, Compile-time Derivation of Variable Dependency Using Abstract Interpretation. To appear in Journal of Logic Programming (also published as Technical Report FIM 59.1/IA/90, Computer Science Dept, Universidad Politecnica de Madrid, Spain, Aug 1990

[Muthukumar,Hermenegildo91] K. Muthukumar and M. Hermenegildo, Combined Determination of Sharing and Freeness of Program Variables Through Abstract Interpretation, *Logic Programming: Proceedings of the Eighth International Conference*, MIT Press, June 1991

[Søndergaard86] H. Søndergaard, An Application of Abstract Interpretation of Logic Programs: Occur-Check Reduction, ESOP86, *Lecture Notes in Computer Science*, 213, Springer Verlag, 1986