ICOT Technical Report: TR-279

TR-279

# Analyzing Success Patterns of Logic Programs by Abstract Hybrid Iuterpretation

by
T. Kanamori and T. Kawamura (Mitsubishi)

June, 1987



ICOT

Mita Kokusai Bldg. 21F
4-28 Mita 1-Chome
Minato-ku Tokyo 108 Japan

(03) 456-3191～5
Telex ICOT J32964

Institute for New Generation Computer Technology

# Analyzing Success Patterns of Logic Programs by Abstract Hybrid Interpretation

Tadashi KANAMORI Tadashi KAWAMURA

Mitsubishi Electric Corporation
Central Research Laboratory
Tsukaguchi-Honmachi 8-1-1
Amagasaki, Hyogo, JAPAN 661

## Abstract

This paper presents a unified framework for analyzing success patterns of Prolog programs by abstract interpretation. The framework is based on OLDT resolution by Tamaki and Sato, a hybrid of the top-down and the bottom-up interpretations of Prolog programs. By directly abstracting the hybrid interpretation according to various abstracted domains, we can obtain various supersets of the goal patterns at calling time and exiting time without either diving into infinite looping or wasting time for irrelevant goal patterns. Depth-abstracted pattern enumeration, type inference and mode analysis are exemplified as analysis of success patterns when different abstracted domains are employed. A general framework for analysis of success patterns is shown as well with the necessary conditions of the approximation.

Keywords : Program Analysis, Abstract Interpretation, Prolog.

## Contents

## 1. Introduction

Automatic analysis of program properties is useful not only for human programmers to find program bugs but also for meta-processing systems to manipulate programs effectively. For example, the information of the form of goals appearing in the successfull execution can eliminate unnecessary backtracking from Prolog execution [12]. The information of data types sometimes plays an important role in verification of Prolog programs [6]. The information of modes provides the Prolog compiler with a chance to generate optimized codes [10]. Besides these properties, the functionality (or determinacy) and the termination are of special importance [7],[3],[8].

It has been known that many seemingly different methods for analyzing the properties of conventional programs can be accommodated into a single framework called *abstract interpretation* [1],[2]. The abstract interpretation is an interpretation of programs in an abstracted domain (rather than the actual standard domain) such that the results of the interpretation in the abstract domain give some useful information about the interpretation in the actual standard domain due to the correspondence between the standard and abstract domains. Recently, Mellish [11] proposed a framework for abstract interpretation of Prolog programs in order to give a theoretical foundation to his practical technieques for analyzing determinacy, modes and shared structures [6]. His approach derives simultaneous recurrence equations for the sets of goals at calling time and exiting time during the top-down execution of a given top-level goal, and obtains a superset of the least solution of the simultaneous recurrence equations using a bottom-up approximation. The reason of the separation of simulating the top-down execution and solving by the bottom-up approximation is two-fold. One is that, by simulating the top-down execution, we can focus on just the goals relevant to the top-level goal. Other is that, by solving by the bottom-up approximation, we can obtain solutions without diving into infinite looping. On the other hand, Tamaki and Sato [13] proposed a new method for interpreting Prolog programs, a hybrid of the top-down and the bottom-up interpretations. Their hybrid interpretation method can compute solutions of a given top-level goal without either diving into infinite looping (unlike the usual top-down interpretation) or wasting time for goals irrelevant to the top-level goal (unlike the usual bottom-up interpretation), so that it is suitable for a *standard interpretation* model underlying abstract interpretation more directly without the separation of simulating the top-level execution and solving by the bottom-up approximation.

This paper presents a unified framework for analyzing success patterns of Prolog programs by abstract interpretation. The framework is based on OLDT resolution by Tamaki and Sato [13]. By directly abstracting the hybrid interpretation according to various abstracted domains, we can obtain various supersets of the goal patterns at calling time and exiting time without either diving into infinite looping or wasting time for irrelevant goal patterns. Depth-abstracted pattern enumeration, type inference and mode analysis are exemplified as analysis of success patterns when different abstracted domains are employed. A general framework for analysis of success patterns is shown as well with the necessary conditions of the approximation.

First, we will show the basic hybrid interpretation method and its implementation in Section 2. Next, we will give a formulation of "analysis of success patterns" in Section 3. Then, after examining "depth-abstracted pattern enumeration" as an introductory example in Section 4, we will show how the approach is generalized for "type inference" and "mode

analysis" in Section 5 and Section 6. Last, we will present a general framework for the analysis of success patterns by abstract hybrid interpretation in Section 7.

In the following, we assume familiarity with the basic terminologies of first order logic such as term, atom, definite clause, negative clause, substitution, most general unifier (m.g.u.) and so on. We follow the syntax of DEC-10 Prolog. Negative clauses and sequences of atoms are often used interchangeably. As syntactical variables, we use $X$, $Y$, $Z$ for variables, $s$, $t$ for terms and $A$, $B$ for atoms, possibly with primes and subscripts. In addition, we use $t[Z]$ for a term containing some occurence of variable $Z$, and $\theta$, $\sigma$, $\tau$, $\eta$ for substitutions.

## 2. Standard Hybrid Interpretation of Logic Programs

In this section, we will first present a basic hybrid interpretation method of Prolog programs [13], then a modified hybrid interpretation method suitable for the basis of the abstract interpretation presented later.

### 2.1 Basic Hybrid Interpretation of Logic Programs

#### (1) Search Tree

A *search tree* is a tree with its nodes labelled with negative or null clauses, and with its edges labelled with substitutions. A *search tree of* negative clause $G$ is a search tree whose root node is labelled with $G$. The relation between a node and its child nodes in a search tree is specified in various ways depending on various strategies of "resolution". In this paper, the class of "ordered linear" strategies is assumed. (See the explanations of OLDT resolution in the following subsection (4), and of OLD resolution in Section 3.)

A *refutation* of negative clause $G$ is a path in a search tree of $G$ from the root to a node labelled with the null clause $\square$. Let $\theta_1, \theta_2, \ldots, \theta_k$ be the labels of the edges on the path. Then, the *answer substitution of the refutation* is the composed substitution $\tau = \theta_1\theta_2 \cdots \theta_k$, and the *solution of the refutation* is $G\tau$.

Consider a path in a search tree from one node to another node. Intuitively, when the leftmost atom of the starting node's label is refuted just at the ending node, the path is called a unit subrefutation of the atom. More formally, let $G_0, G_1, \ldots, G_k$ be a sequence of labels of the nodes and $\theta_1, \theta_2, \ldots, \theta_k$ be the labels of the edges on the path. The path is called a *unit subrefutation* of atom $A$ when $G_0, G_1, G_2, \ldots, G_{k-1}, G_k$ are of the form

$\leftarrow A, G$",
$\leftarrow H_1, G\theta_1$",
$\leftarrow H_2, G\theta_1\theta_2$",
$\vdots$
$\leftarrow H_{k-1}, G\theta_1\theta_2 \cdots \theta_{k-1}$"
$\leftarrow G\theta_1\theta_2 \ldots \theta_k$",

respectively, where $G$, $H_1$, $H_2$, ..., $H_{k-1}$ are sequences of atoms. Then, the *answer substitution of the unit subrefutation* is the composed substitution $\tau = \theta_1\theta_2 \cdots \theta_k$, and the *solution of the unit subrefutaion* is $A\tau$.

#### (2) Solution Table

A *solution table* is a set of entries. Each entry is a pair of the *key* and the *solution list*. The key is an atom such that there is no other identical key (modulo renaming of variables)

in the solution table. The solution list is a list of atoms, called *solutions*, such that each solution in it is an instance of the corresponding key.

**(3) Association**

Let $Tr$ be a search tree whose nodes labelled with non-null clauses are classified into either *solution nodes* or *lookup nodes*, and let $Tb$ be a solution table. (The solution nodes and lookup nodes are explained later.) An *association* of $Tr$ and $Tb$ is a set of pointers pointing from each lookup node in $Tr$ *into* some solution list in $Tb$ such that the leftmost atom of the lookup node's label and the key of the solution list are variants of each other.

*Example 2.1.1* An association of a search tree of "$reach(a, Y_0)$" and a solution table is depicted in the figure below. The underline denotes the lookup node, and the dotted line denotes the association from the lookup node.

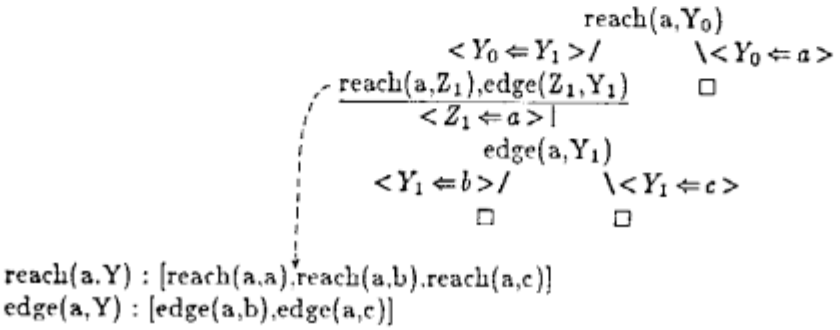



**Figure 2.1.1 Search Tree, Solution Table and Association**

**(4) OLDT Structure**

The hybrid Prolog interpreter is modeled by OLDT resolution. An *OLDT structure* of negative clause $G$ is a triple $(Tr, Tb, As)$ satisfying the following conditions:

(a) $Tr$ is a search tree of $G$. The relation between a node and its child nodes in a search tree is specified by the following *OLDT resolution*. Each node of the search tree labelled with non-null clause is classified into either a *solution node* or a *lookup node*.

(b) $Tb$ is a solution table.

(c) $As$ is an association of $Tr$ and $Tb$. The tail of the solution list pointed from a lookup node is called the *associated solution list* of the lookup node.

Let $G$ be a negative clause of the form "$A_1, A_2, \ldots, A_n$" ($n \geq 1$). A node of OLDT structure $(Tr, Tb, As)$ labelled with negative clause $G$ is said to be *OLDT resolvable* when it satisfies either of the following conditions:

(a) The node is a terminal solution node of $Tr$, and there is some definite clause "$B_0$ :- $B_1, B_2, \ldots, B_m$" ($m \geq 0$) in program $P$ such that $A_1$ and $B_0$ are unifiable, say by an m.g.u. $\theta$. (Without loss of generality, we assume that the m.g.u. $\theta$ substitutes a term consisting of fresh variables for every variable in $A_1$ and the definite clause.) The negative clause (or possibly null clause) "$B_1\theta, B_2\theta, \ldots, B_m\theta, A_2\theta, \ldots, A_n\theta$" is called the *OLDT resolvent*.

(b) The node is a lookup node of $Tr$, and there is some solution $B\tau$ in the associated solution list of the lookup node such that $B\tau$ is an instance of $A_1$, say by an instantiation $\theta$. (Again, we assume that the instantiation $\theta$ substitutes a term consisting of fresh variables for every variable in $A_1$, that is, a fresh variant of $B\tau$ is an instance of $A_1$ by

$\theta$.) The negative clause (or possibly null clause) "$A_2\theta, \ldots, A_n\theta$" is called the *OLDT resolvent*.

The restriction of the substitution $\theta$ to the variables of $A_1$ is called the *substitution of the OLDT resolution*.

The *initial OLDT structure* of negative clause $G$ is the triple $(Tr_0, Tb_0, As_0)$, where $Tr_0$ is a search tree consisting of just the root solution node labelled with $G$, $Tb_0$ is the solution table consisting of just one entry whose key is the leftmost atom of $G$ and solution list is the empty list, and $As_0$ is the empty set of pointers.

An *immediate extension* of OLDT structure $(Tr, Tb, As)$ in program $P$ is the result of the following operations, when a node $v$ of OLDT structure $(Tr, Tb, As)$ is OLDT resolvable.

(a) When $v$ is a terminal solution node, let $C_1, C_2, \ldots, C_k$ $(k \geq 0)$ be all the clauses with which the node $v$ is OLDT resolvable, and $G_1, G_2, \ldots, G_k$ be the respective OLDT resolvents. Then add $k$ child nodes of $v$ labelled with $G_1, G_2, \ldots, G_k$, to $v$. The edge from $v$ to the node labelled with $G_i$ is labelled with $\theta_i$, where $\theta_i$ is the substitution of the OLDT resolution with $C_i$. When $v$ is a lookup node, let $B_1\tau_1, B_2\tau_2, \ldots, B_k\tau_k$ $(k \geq 0)$ be all the solutions with which the node $v$ is OLDT resolvable, and $G_1, G_2, \ldots, G_k$ be the respective OLDT resolvents. Then add $k$ child nodes of $v$ labelled with $G_1, G_2, \ldots, G_k$, to $v$. The edge from $v$ to the node labelled with $G_i$ is labelled with $\theta_i$, where $\theta_i$ is the substitution of the OLDT resolution with $B_i\tau_i$. A new node labelled with a non-null clause is a lookup node when the leftmost atom of the new negative clause is a variant of some key in $Tb$, and is a solution node otherwise.

(b) Replace the pointer from the OLDT resolved lookup node with the one pointing to the last of the associated solution list. Add a pointer from the new lookup node to the head of the solution list of the corresponding key.

(c) When a new node is a solution node, add a new entry whose key is the leftmost atom of the label of the new node and whose solution list is the empty list. When a new node is a lookup node, add no new entry. For each unit subrefutation of atom $A$ (if any) starting from a solution node and ending with some of the new nodes, add its solution $A\tau$ to the last of the solution list of $A$ in $Tb$, if $A\tau$ is not in the solution list.

An OLDT structure $(Tr', Tb', As')$ is an extension of OLDT structure $(Tr, Tb, As)$ if $(Tr', Tb', As')$ is obtained from $(Tr, Tb, As)$ through successive application of immediate extensions.

*Example 2.1.2* Consider the following "graph reachability" program by Tamaki and Sato [13].

```
reach(X,Y) :- reach(X,Z), edge(Z,Y).
reach(X,X).
edge(a,b).
edge(a,c).
edge(b,a).
edge(b,d).
```

Then, the hybrid interpretation generates the following OLDT structures of "$reach(a, Y_0)$".

First, the initial OLDT structure below is generated. The root node of the search tree is a solution node. The solution table contains only one entry with its key $reach(a, Y)$ and its solution list [ ].

reach(a,$Y_0$)

reach(a,Y) : [ ]

**Figure 2.1.2 Basic Hybrid Interpretation at Step 1**

Secondly, the root node "$reach(a, Y_0)$" is OLDT resolved using the program to generate two child nodes. The generated left child node is a lookup node, because its leftmost atom is a variant of the key in the solution table. The association associates the lookup node to the head of the solution list of $reach(a, Y)$. The generated right child node is the end of a unit subrefutation of $reach(a, Y_0)$. Its solution $reach(a, a)$ is added to the solution list of $reach(a, Y)$.

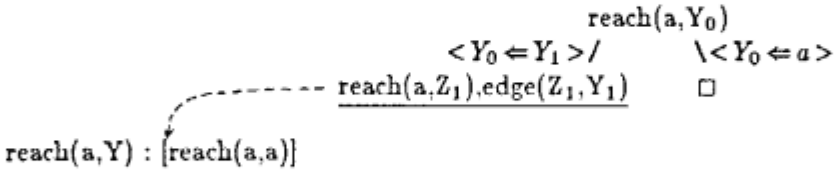



**Figure 2.1.3 Basic Hybrid Interpretation at Step 2**

Thirdly, the lookup node is OLDT resolved using the solution table to generate one child solution node. The association associates the lookup node to the last of the solution list.

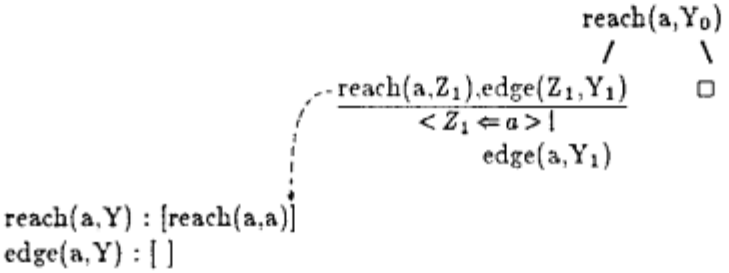



**Figure 2.1.4 Basic Hybrid Interpretation at Step 3**

Fourthly, the generated solution node is OLDT resolved further using the program to generate two new nodes labelled with the null clauses. These two nodes add two solutions $reach(a, b)$ and $reach(a, c)$ to the last of the solution list of $reach(a, Y)$, and two solutions $edge(a, b)$ and $edge(a, c)$ to the last of the solution list of $edge(a, Y)$.

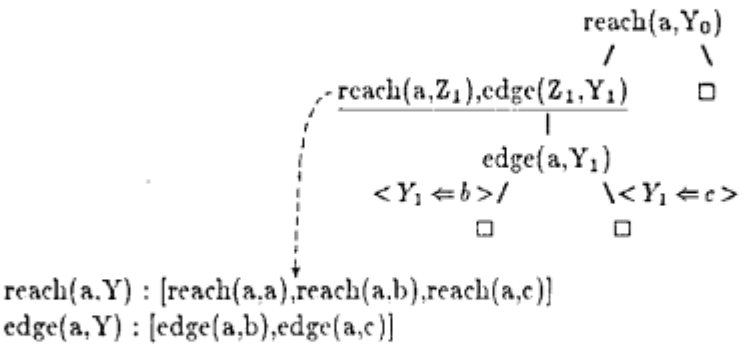
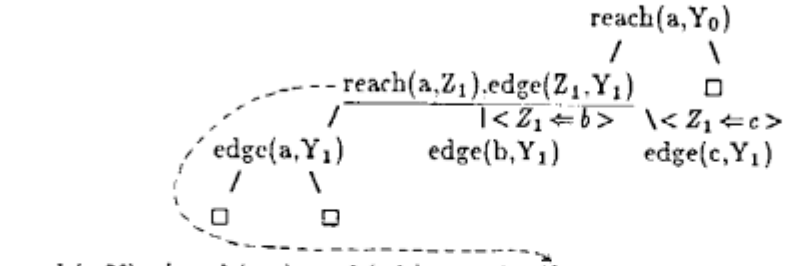



**Figure 2.1.5 Basic Hybrid Interpretation at Step 4**

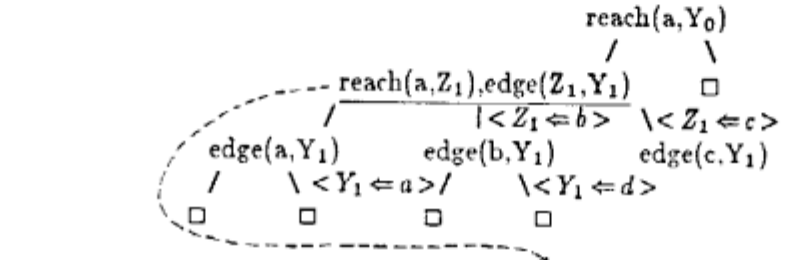
Fifthly, the lookup node is OLDT resolved using the solution table, since new solutions were added to the solution list of $reach(a, Y)$.




reach(a,Y) : [reach(a,a),reach(a,b),reach(a,c)]
edge(a,Y) : [edge(a,b),edge(a,c)]
edge(b,Y) : [ ]
edge(c,Y) : [ ]

**Figure 2.1.6 Basic Hybrid Interpretation at Step 5**

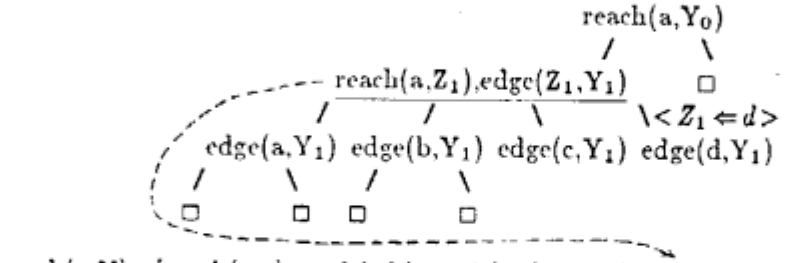
Sixthly, the left new solution node "$edge(b, Y_1)$" is OLDT resolved, and one new solution $reach(a, d)$ is added to the solution list of $reach(a, Y)$.




reach(a,Y) : [reach(a,a),reach(a,b),reach(a,c),reach(a,d)]
edge(a,Y) : [edge(a,b),edge(a,c)]
edge(b,Y) : [edge(b,a),edge(b,d)]
edge(c,Y) : [ ]

**Figure 2.1.7 Basic Hybrid Interpretation at Step 6**

Lastly, the lookup node is OLDT resolved once more using the solution table, and the extension process stops, because the solution nodes labelled with $edge(c, Y_1)$ and $edge(d, Y_1)$ are not OLDT resolvable.




reach(a,Y) : [reach(a,a),reach(a,b),reach(a,c),reach(a,d)]
edge(a,Y) : [edge(a,b),edge(a,c)]
edge(b,Y) : [edge(b,a),edge(b,d)]
edge(c,Y) : [ ]
edge(d,Y) : [ ]

**Figure 2.1.8 Basic Hybrid Interpretation at Step 7**

Though all solutions were found under the depth-first from-left-to-right extension strategy in this example, the strategy is not complete in general. The reason of the incompleteness is two-fold. One is that there might be generated infinitely many different solution nodes. Another is that some lookup node might generate infinitely many child nodes so that extensions at other nodes right to the lookup node might be inhibited forever.

### (5) Soundness and Completeness of OLDT Resolution

Let $G$ be a negative clause. An *OLDT refutation* of $G$ in program $P$ is a refutation in the search tree of some extension of OLDT structure of $G$. The *answer substitution of the OLDT refutation* and the *solution of the OLDT refutation* are defined in the same way as before. It is a basis of the abstract interpretation in this paper that OLDT resolution is sound and complete. (Do not confuse the completeness of the general OLDT resolution with the incompleteness of the one under a specific extension strategy, e.g., the depth-first from-left-to-right strayegy.)

**Theorem 2.1 (Soundness and Completeness of OLDT Resolution)**
If $G\tau$ is a solution of an OLDT refutation of $G$ in $P$, its universal closure $\forall X_1 X_2 \cdots X_n$ $G\tau$ is a logical consequence of $P$.

If a universal closure $\forall Y_1 Y_2 \cdots Y_m$ $G\sigma$ is a logical consequence of $P$, there is $G\tau$ which is a solution of an OLDT refutation of $G$ in $P$ and $G\sigma$ is an instance of $G\tau$.

*Proof.* Though our hybrid interpretation is different from the original OLDT resolution by Tamaki and Sato [13] in two respects (see Section 8), these differences do not affect the proof of the soundness and the completeness. See Tamaki and Sato [13] pp.93-94.

### 2.2 Modified Hybrid Interpretation of Logic Programs

In order to make the conceptual presentation of the hybrid interpretation simpler, we have not considered the details of how it is implemented. In particular, it is not obvious in the "immediate extension of OLDT structure"

(a) how we can know whether a new node is the end of a unit subrefutation starting from some solution node, and

(b) how we can obtain the solution of the unit subrefutation efficiently if any.

It is an easy solution to insert a special call-exit marker $[\![A_1, \theta]\!]$ between $B_1\theta$, $B_2\theta$, ..., $B_m\theta$ and $A_2\theta$, ..., $A_n\theta$ when a solution node is OLDT resolved using an m.g.u. $\theta$, and obtain the unit subrefutation of $A_1$ and its solution $A_1\tau$ when the leftmost of a new OLDT resolvent is the special call-exit marker $[\![A_1, \tau]\!]$. But, we will use the following modified framework. (Though such redefinition might be confusing, it is a little difficult to grasp the intuitive meaning of the modified framework without the explanation in Section 2.1.)

A *search tree* of OLDT structure in the modified framework is a tree with its nodes labelled with a pair of a (generalized) negative clause and a substitution. (We have said "generalized", because it might contain non-atoms, i.e., call-exit markers. The edges are not labelled with substitutions any more.) A *search tree of* $(G, \sigma)$ is a serach tree whose root node is labelled with $(G, \sigma)$. The clause part of each label is a sequence "$\alpha_1$, $\alpha_2$, ..., $\alpha_n$" consisting of either atoms in the body of the clauses in $P \cup \{G\}$ or *call-exit markers* of the form $[\![A, \sigma']\!]$. A *refutation of* $(G, \sigma)$ is a path in a search tree of $(G, \sigma)$ from the root to a node labelled with $(\square, \tau)$. The *answer substitution of the refutation* is the substitution $\tau$, and the *solution of the refutation* is $G\tau$. A *solution table* and an *association* are defined in the same way as before.

An *OLDT structure* is a triple of a search tree, a solution table and an association. The relation between a node and its child nodes in search trees of OLDT structures is specified by the following modified OLDT resolution.

A node of OLDT structure $(Tr, Tb, As)$ labelled with ("$\alpha_1,\alpha_2,\ldots,\alpha_n$", $\sigma$) is said to be *OLDT resolvable* when it satisfies either of the following conditions:

(a) The node is a terminal solution node of $Tr$, and there is some definite clause "$B_0$ :- $B_1, B_2, \ldots, B_m$" ($m \geq 0$) in program $P$ such that $\alpha_1\sigma$ and $B_0$ are unifiable, say by an m.g.u. $\theta$.

(b) The node is a lookup node of $Tr$, and there is some solution $B\tau$ in the associated solution list of the lookup node such that (a fresh variant of) $B\tau$ is an instance of $\alpha_1\sigma$, say by an instantiation $\theta$.

The OLDT resolvent is obtained through the following two phases, called *calling phase* and *exiting phase* since they correspond to a "Call" (or "Redo") line and an "Exit" line in the messages of the conventional DEC10 Prolog tracer. A call-exit marker is inserted in the calling phase when a node is OLDT resolved using the program, while no call-exit marker is generated when a node is OLDT resolved using the solution table. When there is a call-exit marker at the leftmost of the clause part in the exiting phase, it means that some unit subrefutation is obtained.

(a) (Calling Phase) When a node labelled with ("$\alpha_1, \alpha_2, \ldots, \alpha_n$", $\sigma$) is OLDT resolved, the intermediate label is generated as follows:

  a-1. When the node is OLDT resolved using a definite clause "$B_0$ :- $B_1, B_2, \ldots, B_m$" in program $P$ and an m.g.u. $\theta$, the intermediate clause part is "$B_1,B_2,\ldots,B_m$, $[\![\alpha_1.\sigma]\!]$, $\alpha_2,\ldots,\alpha_n$", and the intermediate substitution part $\tau_0$ is $\theta$.

  a-2. When the node is OLDT resolved using a solution $B\tau$ in the solution table and an instantiation $\theta$, the intermediate clause part is "$\alpha_2, \ldots, \alpha_n$", and the intermediate substitution part $\tau_0$ is $\sigma\theta$.

(b) (Exiting Phase) When there are $k$ call-exit markers $[\![A_1,\sigma_1]\!]$, $[\![A_2,\sigma_2]\!]$, ..., $[\![A_k,\sigma_k]\!]$ at the leftmost of the intermediate clause part, the label of the new node is generated as follows:

  b-1. The clause part is obtained by eliminating all these call-exit markers. The substitution part is $\sigma_k \cdots \sigma_2\sigma_1\tau_0$.

  b-2. Add $A_1\sigma_1\tau_0$, $A_2\sigma_2\sigma_1\tau_0$, ..., $A_k\sigma_k \cdots \sigma_1\tau_0$ to the last of the solution lists of $A_1\sigma_1$, $A_2\sigma_2$, ..., $A_k\sigma_k$, respectively, if they are not in the solution lists.

The precise algorithm is shown in Figure 2.2.1. The processing at the calling phase is performed in the first **case** statement, while that of the exiting phase is performed in the second **while** statement successively.

Note that, when a node is labelled with $(G, \sigma)$, the substitution part $\sigma$ always shows the instantiation of atoms to the left of the leftmost call-exit marker in $G$. When there is a call-exit marker $[\![A_j,\sigma_j]\!]$ at the leftmost of clause part in the exiting phase, we need to update the substitution part by composing $\sigma_j$ in order that the property above still holds after eliminating the call-exit marker. The sequence $\tau_1, \tau_2, \ldots, \tau_i$ denotes the sequence of updated substitutions. In addition, when we pass a call-exit marker $[\![A_j,\sigma_j]\!]$ in the **while** loop above with substitution $\tau_j$, the atom $A_j\tau_j$ denotes the solution of the unit subrefutation of $A_j\sigma_j$. The solution $A_j\tau_j$ is added to the solution list of $A_j\sigma_j$.

A node labelled with ("$\alpha_1, \alpha_2, \ldots, \alpha_n$", $\sigma$) is a lookup node when a variant of atom $\alpha_1\sigma$ already exists as a key in the solution table, and is a solution node otherwise ($n \geq 1$).

OLDT-resolve(("$\alpha_1, \alpha_2, \ldots, \alpha_n$", $\sigma$) : label) : label ;
  $i := 0$;
  **case**
    **when** a solution node is OLDT resolved with "$B_0$ :- $B_1, B_2, \ldots, B_m$" in $P$
      let $\theta$ be the m.g.u. of $\alpha_1\sigma$ and $B_0$ ;
      let $G_0$ be a negative clause "$B_1, B_2, \ldots, B_m, [\![\alpha_1, \sigma]\!], \alpha_2, \ldots, \alpha_n$" ;
      let $\tau_0$ be the substitution $\theta$ ; — (A)
    **when** a lookup node is OLDT resolved with "$B\tau$" in $Tb$
      let $\theta$ be the instantiation of $\alpha_1\sigma$ to (a fresh variant of) $B\tau$ ;
      let $G_0$ be a negative caluse "$\alpha_2, \ldots, \alpha_n$" ;
      let $\tau_0$ be the composed substitution $\sigma\theta$ ; — (B)
  **endcase**
  **while** the leftmost of $G_i$ is a call-exit marker $[\![A_{i+1}, \sigma_{i+1}]\!]$ **do**
      let $G_{i+1}$ be $G_i$ other than the leftmost call-exit marker ;
      let $\tau_{i+1}$ be $\sigma_{i+1}\tau_i$ ; — (C)
      add $A_{i+1}\tau_{i+1}$ to the last of $A_{i+1}\sigma_{i+1}$'s solution list if it is not in it ;
      $i := i + 1$ ;
  **endwhile**
  $(G_{new}, \sigma_{new}) := (G_i, \tau_i)$ ;
  **return** $(G_{new}, \sigma_{new})$.

**Figure 2.2.1 Modified Hybrid Interpretation**

The *initial OLDT structure* of $(G, \sigma)$ is a triple $(Tr_0, Tb_0, As_0)$, where $Tr_0$ is a search tree of $G$ consisting of just the root solution node labelled with $(G, \sigma)$, $Tb_0$ is a solution table consisting of just one entry whose key is the leftmost atom of $G$ and solution list is [ ], and $As_0$ is the empty set of pointers. The *immediate extension of OLDT structure*, *extension of OLDT structure*, *answer substitution of OLDT refutation* and *solution of OLDT refutation* are defined in the same way as before.

*Example 2.2* Consider the example in Section 2.1 again. The modified hybrid interpretation generates the following OLDT structures of $reach(a, Y_0)$.

First, the initial OLDT structure below is generated. Now, the root node is labelled with ("$reach(a, Y_0)$",<>).

reach(a,$Y_0$)
<>

reach(a,Y) : [ ]

**Figure 2.2.2 Modified Hybrid Interpretation at Step 1**

Secondly, the root node ("$reach(a, Y_0)$",<>) is OLDT resolved using the program to generate two child nodes. The intermediate label of the left child node is

("$reach(X_1, Z_1), edge(Z_1, Y_1), [\![reach(a, Y_0), <> ]\!]$", $<Y_0 \Leftarrow Y_1, X_1 \Leftarrow a>$).

It is the new label immediately, since its leftmost is not a call-exit marker. The intermediate label of the right child node is

("$[reach(a, Y_0), <> ]$", $< Y_0 \Leftarrow a, X_1 \Leftarrow a >$).
By eliminating the leftmost call-exit marker and composing the substitution, the new label is ($\square$,$< Y_0 \Leftarrow a, X_1 \Leftarrow a >$). (When the clause part of the label is $\square$, we will omit the assignments irrelevant to the top-level goal in the following figures, e.g., $< X_1 \Leftarrow a >$.) During the elimination of the call-exit marker, $reach(a, a)$ is added to the solution table.

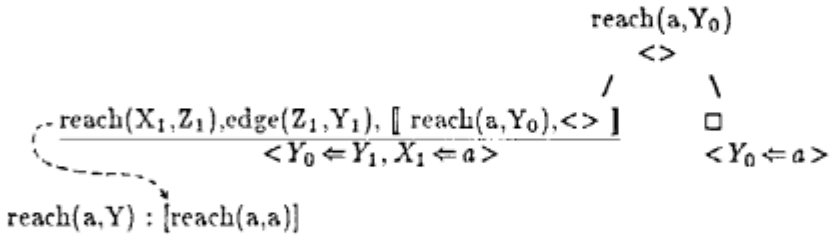



**Figure 2.2.3 Modified Hybrid Interpretation at Step 2**

Thirdly, the left lookup node is OLDT resolved using the solution table to generate one child solution node.

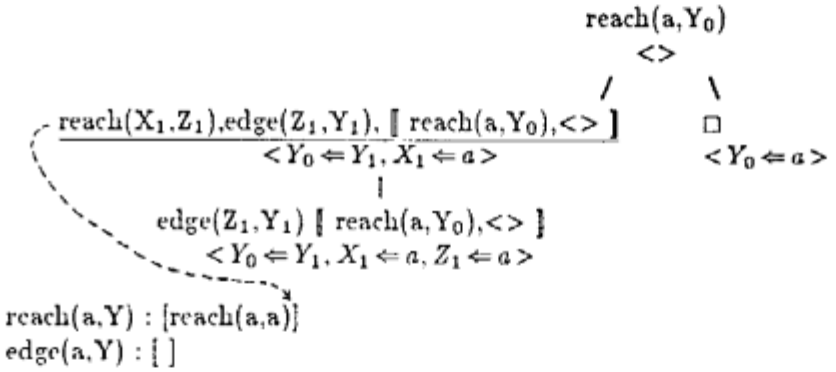



**Figure 2.2.4 Modified Hybrid Interpretation at Step 3**

Fourthly, the generated solution node is OLDT resolved using a unit clause "$edge(a, b)$" in program $P$ to generate the intermediate label

("$[edge(Z_1, Y_1), < Y_0 \Leftarrow Y_1, X_1 \Leftarrow a, Z_1 \Leftarrow a > ]$, $[reach(a, Y_0), <> ]$", $< Y_1 \Leftarrow b >$).

By eliminating the leftmost call-exit markers and composing substitutions, the new label is ($\square$, $< Y_0 \Leftarrow b, X_1 \Leftarrow a, Z_1 \Leftarrow a, Y_1 \Leftarrow b >$). During the elimination of the call-exit markers, $edge(a, b)$ and $reach(a, b)$ are added to the solution table.

Similarly, the node is OLDT resolved using a unit clause "$edge(a, c)$" in program $P$ to generate the intermediate label

("$[edge(Z_1, Y_1), < Y_0 \Leftarrow Y_1, X_1 \Leftarrow a, Z_1 \Leftarrow a > ]$, $[reach(a, Y_0), <> ]$", $< Y_1 \Leftarrow c >$)

By eliminating the leftmost call-exit markers and composing substitutions similarly, the new label is ($\square$, $< Y_0 \Leftarrow c, X_1 \Leftarrow a, Z_1 \Leftarrow a, Y_1 \Leftarrow c >$). This time, $edge(a, b)$ and $reach(a, b)$ are added to the solution table during the elimination of the call-exit markers.
The process of extension proceeds similarly to obtain all the solutions as in Example 2.1.2.

*Remark.* Note that we no longer need to keep the edges and the non-terminal solution nodes of search trees. In addition, we can throw away assignments in $\theta$ for the variables in $B\tau$ at step (B), and those in $\tau_i$ for variables not in $A_{i+1}\sigma_{i+1}$ at step (C) in Figure 2.2.1.

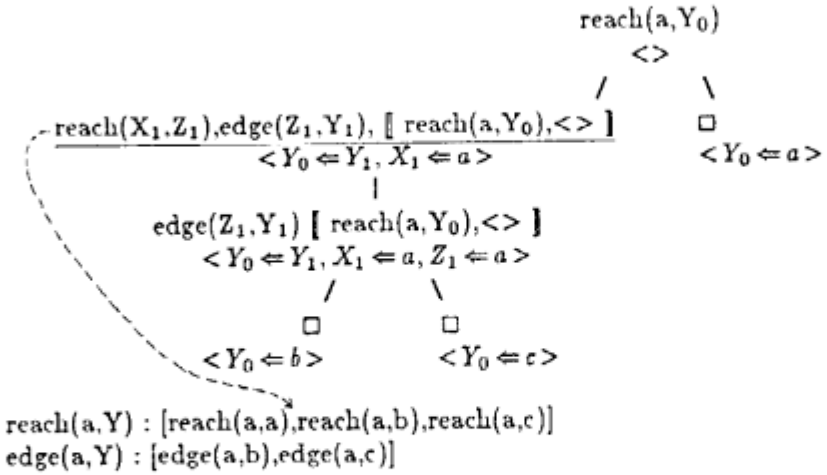



**Figure 2.2.5 Modified Hybrid Interpretation at Step 4**

### 3. Analysis of Success Patterns

Let **G** be a set of negative clauses to be given as top-level goals, and $G$ be a negative clause in **G**. Suppose that $A$ is the leftmost atom of a negative clause appearing in some top-down execution of $G$. Then, of what form is $A$? And, of what form isn't $A$? Similarly, of what form is its solution $A\tau$? Can we say that, if $A$ is not an atom in some class of atoms, it never occurs at the leftmost in any successfull execution of $G$?

Let us formalize the top-down execution first. The top-down Prolog interpreter is modeled by OLD resolution. The OLD resolution is defined using just search trees, called OLD trees. (Because there is neither a solution table nor an association, we have no distinction of solution nodes and lookup nodes. All nodes are solution nodes.) The relation between a node and its child nodes in OLD trees is specified in the same way as the OLDT resolution in Section 2.1, except that we have no resolution using lookup nodes and solution tables, hence no manipulation of solution table and association.

An atom $A$ appearing at the leftmost of the label of a node in some OLD tree of $G$ is called *calling pattern at local success of* $G$. Note that any calling pattern at local success of $G$ is an instance of some atom in the body of a clause in $P \cup \{G\}$. Each calling pattern at local success corresponds to some key in the solution table of OLDT structure.

An atom $A$ appearing at the leftmost of the label of a node in some OLD refutation of $G$ is called *calling pattern at global success of* $G$. Note that any calling pattern at global success of $G$ is a calling pattern at local success, but not vice versa, because the patterns appearing at calling time, which are cancelled by backtracking without leading to global success, are not considered calling patterns at global success.

A solution $A\tau$ of a subrefutation in an OLD tree of $G$ is called an *exiting pattern at local success of* $G$. Note that any exiting pattern at local success of $G$ is also an instance of some atom in the body of a clause in $P \cup \{G\}$. Each exiting pattern at local success corresponds to some element in the solution lists of OLDT structure.

A solution $A\tau$ of a subrefutation in an OLD refutation of $G$ is called an *exiting pattern at global success of* $G$. Note that any exiting pattern at global success of $G$ is an exiting pattern at local success, but not vice versa, because of the same reason as calling patterns.

Let $\mathcal{C}_{local}(\mathbf{G})$, $\mathcal{C}_{global}(\mathbf{G})$, $\mathcal{E}_{local}(\mathbf{G})$, $\mathcal{E}_{global}(\mathbf{G})$ be the sets defined for the extensions of the initial OLD tree of $G$ in $\mathbf{G}$ as follows:

$\mathcal{C}_{local}(\mathbf{G})$ : the set of all calling patterns at local success of $G$ in $\mathbf{G}$,
$\mathcal{C}_{global}(\mathbf{G})$ : the set of all calling patterns at global success of $G$ in $\mathbf{G}$,
$\mathcal{E}_{local}(\mathbf{G})$ : the set of all exiting patterns at local success of $G$ in $\mathbf{G}$,
$\mathcal{E}_{global}(\mathbf{G})$ : the set of all exiting patterns at global success of $G$ in $\mathbf{G}$.

The *analysis of success patterns* w.r.t. $\mathbf{G}$ is the problem to compute

(a) some superset of $\mathcal{C}_{local}(\mathbf{G})$,
(b) some superset of $\mathcal{C}_{global}(\mathbf{G})$,
(c) some superset of $\mathcal{E}_{local}(\mathbf{G})$, and
(d) some superset of $\mathcal{E}_{global}(\mathbf{G})$.

In general, there might exist infinite number of calling patterns and exiting patterns so that we can't enumerate $\mathcal{C}_{local}(\mathbf{G})$, $\mathcal{C}_{global}(\mathbf{G})$, $\mathcal{E}_{local}(\mathbf{G})$ or $\mathcal{E}_{global}(\mathbf{G})$ exactly in finite steps. In order to obtain appropriate supersets in *finite* steps, we need some *approximation*. Moreover, since what we would like to compute is *supersets* of the sets of calling patterns and exiting patterns, we need to *overestimate* them somehow. In Section 4, we present a method for depth-abstracted pattern enumeration as an introductory example of the analysis of success patterns.

## 4. Depth-abstracted Pattern Enumeration by Abstract Hybrid Interpretation

Suppose that a top-level goal is executed with its some arguments instantiated to terms of specific form. Then, how can we know of what form the arguments of the goals are, when they are invoked (or they succeed) during the top-down execution of the top-level goal? In particular, can we say that some predicate is always invoked (or succeeds) with its argument instantiated to some instance of a specific term?

In this section, we will reformulate the work by Sato and Tamaki [12] from the point of view in Section 2.2.

### 4.1 Depth-abstracted Pattern Enumeration

The *depth* of term $t$ is defined as follows:

(a) When $t$ is a variable, the depth of $t$ is 0.
(b) When $t$ is a constant $c$, the depth of $t$ is 0.
(c) When $t$ is a term of the form $f(t_1, t_2, \ldots, t_n)$, the depth of $t$ is the maximum of the depths of $t_1$, $t_2$, ..., $t_n$, plus 1.

A *depth $d$ abstracted term* is a term whose depth is at most $d$, and denotes the set of all its instances. Note that the set of all depth $d$ abstracted terms is finite for any given $d$, because program $P$ is a finite list of definite clauses, hence there are only finite function symbols. A depth-abstracted term $s$ is said to be *smaller than* a depth-abstracted term $t$ *w.r.t. the instantiation ordering* if and only if $t$ is an instance of $s$, and denoted by $s \preceq t$. A depth-abstracted term $s$ is said to be *smaller than* a depth-abstracted term $t$ *w.r.t. the set inclusion ordering* if and only if the set of terms $s$ denotes is a subset of that $t$ does, and denoted by $s \subseteq t$. (Though the instantiation ordering is just the reverse of the set inclusion ordering here, the coincidence is not always the case. See mode anlysis in Section 6.)

A *depth $d$ abstracted substitution* is an expression of the form
$< X_1 \Leftarrow \underline{t_1}, X_2 \Leftarrow \underline{t_2}, \ldots, X_l \Leftarrow \underline{t_l} >$.

where $\underline{t_1}, \underline{t_2}, \ldots, \underline{t_l}$ are depth $d$ abstracted terms. The depth $d$ abstracted term assigned to variable $X$ by depth $d$ abstracted substitution $\mu$ is denoted by $\mu(X)$.

Let $A$ be an atom in the body of some clause in $P \cup \{G\}$ and $\mu$ be a depth $d$ abstracted substitution of the form

$< X_1 \Leftarrow \underline{t_1}, X_2 \Leftarrow \underline{t_2}, \ldots, X_l \Leftarrow \underline{t_l} >$.

Then $A\mu$ is called a *depth-abstracted atom*, and denotes the set of all atoms obtained by replacing each $X_i$ in $A$ with a term in $\underline{t_i}$. A list of depth-abstracted atoms $[A_1\mu_1, A_2\mu_2, \ldots, A_n\mu_n]$ denotes the set union $\cup_{i=1}^{n} A_i\mu_i$. Similarly, $G\mu$ (or the pair $(G, \mu)$) is called a *depth-abstracted negative clause*, and denotes the set of negative clauses obtained by replacing each $X_i$ in $G$ with a term in $\underline{t_i}$. When $G$ is of the form "$A_1$, $A_2$, ..., $A_n$", the depth-abstracted atom $A_1\mu$ is called the leftmost depth-abstracted atom of $G\mu$.

The *depth-abstracted pattern enumeration* w.r.t. $G\mu$ is the problem to compute

(a) some list of depth-abstracted atoms which is a superset of $\mathcal{C}_{local}(G\mu)$,
(b) some list of depth-abstracted atoms which is a superset of $\mathcal{C}_{global}(G\mu)$,
(c) some list of depth-abstracted atoms which is a superset of $\mathcal{E}_{local}(G\mu)$, and
(d) some list of depth-abstracted atoms which is a superset of $\mathcal{E}_{global}(G\mu)$.

## 4.2 Abstract Hybrid Interpretation for Depth-abstracted Pattern Enumeration

### 4.2.1 OLDT Structure for Depth-abstracted Pattern Enumeration

A *search tree for depth-abstracted pattern enumeration* is a tree with its nodes labelled with a pair of a (generalized) negative clause and a depth-abstracted substitution. (For brevity, we will sometimes omit the term "for depth-abstracted pattern enumeration" hereafter in Section 4.) A *search tree of* $(G, \mu)$ is a search tree whose root node is labelled with $(G, \mu)$. The clause part of each pair is a sequence "$\alpha_1$, $\alpha_2$, ..., $\alpha_n$" consisting of either atoms in the body of $P \cup \{G\}$ or *call-exit markers* of the form $[A, \mu']$. A *refutation of* $(G, \mu)$ is a path in a search tree of $(G, \mu)$ from the root to a node labelled with $(\Box, \nu)$. The *answer substitution of the refutation* is the depth-abstracted substitution $\nu$, and the *solution of the refutation* is $G\nu$.

A *solution table for depth-abstracted pattern enumeration* is a set of entries. Each entry is a pair of the *key* and the *solution list*. The key is a depth-abstracted atom. The solution list is a list of depth-abstracted atoms, called *solutions*, whose all solutions are greater than the key w.r.t. the instantiation ordering, i.e., instances of the key.

Let $Tr$ be a search tree whose nodes labelled with non-null clauses are classified into either *solution nodes* or *lookup nodes*, and let $Tb$ be a solution table. An *association* of $Tr$ and $Tb$ *for depth-abstracted pattern enumeration* is a set of pointers pointing from each lookup node in $Tr$ *into* some solution list in $Tb$ such that the leftmost depth-abstracted atom of the lookup node's label and the key of the solution list are variants of each other.

An *OLDT structure for depth-abstracted pattern enumeration* is a triple of a search tree, a solution table and an association. The relation between a node and its child nodes is specified by *OLDT resolution for depth-abstracted pattern enumeration* in Section 4.2.3.

### 4.2.2 Overestimation of Depth-abstracted Patterns

Because the purpose of the depth-abstracted pattern enumeration is to compute supersets of the sets of calling patterns and exiting patterns using lists of depth-abstracted atoms, we need to overestimate them somehow by manipulating depth-abstracted atoms. A term $t$ is a *level* 0 subterm of $t$ itself. $t_1, t_2, \ldots, t_n$ are *level* $d+1$ subterms of $t$, when $f(t_1, t_2, \ldots, t_n)$ is a level $d$ subterm of $t$. A term obtained from term $t$ by replacing every level $d$ non-variable non-constant subterm of $t$ with a newly created distinct variable is called the *depth $d$ abstraction of $t$*, and denoted by $[t]_d$. Let $\theta$ be a substitution of the form

$< X_1 \Leftarrow t_1, X_2 \Leftarrow t_2, \ldots, X_l \Leftarrow t_l >$.

Then the substitution

$< X_1 \Leftarrow [t_1]_d, X_2 \Leftarrow [t_2]_d, \ldots, X_l \Leftarrow [t_l]_d >$

is called the *depth $d$ abstraction of $\theta$*, and denoted by $[\theta]_d$. Now on, we assume a fixed natural number $d$, and will omit the suffix $d$.

*Example 4.2* Let $t$ be a term $f(g(X,a),Y,b)$ and $U, V$ be fresh variables ([13] p.642). Then

$$[t]_d = \begin{cases} U, & \text{when } d = 0; \\ f(V,Y,b), & \text{when } d = 1; \\ t, & \text{when } d \geq 2. \end{cases}$$

Note that $Y$ and $b$ are not replaced with new variables when $d = 1$, and neither are $X$ and $a$ when $d = 2$.

### 4.2.3 OLDT Resolution for Depth-abstracted Pattern Enumeration

Using the depth-abstraction operation, the relation between a node and its child nodes of a search tree is specified by the following *OLDT resolution for depth-abstracted pattern enumeration.*

A node of OLDT structure for depth-abstracted pattern enumeration $(Tr, Tb, As)$ labelled with ("$\alpha_1, \alpha_2, \ldots, \alpha_n$", $\mu$) is said to be *OLDT resolvable* when it satisfies either of the following conditions:

(a) The node is a terminal solution node of $Tr$, and there is some definite clause "$B_0$ :- $B_1, B_2, \ldots, B_m$" ($m \geq 0$) in program $P$ such that $\alpha_1\mu$ and $B_0$ are unifiable, say by an m.g.u. $\eta$.

(b) The node is a lookup node of $Tr$, and there is some solution $B\nu$ in the associated solution list of the lookup node such that (a fresh variant of) $B\nu$ is an instance of $\alpha_1\mu$, say by an instantiation $\eta$.

The precise algorithm of OLDT resolutuion for depth-abstracted pattern enumeration is shown in Figure 4.2.3. Note that the operations at steps (A), (B) and (C) in Figure 2.2.1 are modified.

A node labelled with ("$\alpha_1, \alpha_2, \ldots, \alpha_n$", $\mu$) is a lookup node when a variant of $\alpha_1\mu$ already exists as a key in the solution table, and is a solution node otherwise ($n \geq 1$).

The *initial OLDT structure*, *immediate extension of OLDT structure*, *extension of OLDT structure*, *answer substitution of OLDT refutation* and *solution of OLDT refutation* are defined in the same way as in Section 2.2.

OLDT-resolve(("$\alpha_1, \alpha_2, \ldots, \alpha_n$", $\mu$) : label) : label ;
  $i := 0$;
  **case**
    **when** a solution node is OLDT resolved with "$B_0 :- B_1, B_2, \ldots, B_m$" in $P$
      let $\eta$ be the m.g.u. of $\alpha_1\mu$ and $B_0$ ;
      let $G_0$ be a negative clause "$B_1, B_2, \ldots, B_m, [\![\alpha_1, \mu]\!], \alpha_2, \ldots, \alpha_n$" ;
      let $\nu_0$ be $[\eta]_d$ ; — (A)
    **when** a lookup node is OLDT resolved with $B\nu$ in $Tb$
      let $\eta$ be the instantiation of $\alpha_1\mu$ to (a fresh variant of) $B\nu$ ;
      let $G_0$ be a negative caluse "$\alpha_2, , \ldots, \alpha_n$" ;
      let $\nu_0$ be $[\mu\eta]_d$ ; — (B)
  **endcase**
  **while** the leftmost of $G_i$ is a call-exit marker $[\![A_{i+1}, \mu_{i+1}]\!]$ **do**
      let $G_{i+1}$ be $G_i$ other than the leftmost call-exit marker ;
      let $\nu_{i+1}$ be $[\mu_{i+1}\nu_i]_d$ ; — (C)
      add $A_{i+1}\nu_{i+1}$ to the last of $A_{i+1}\mu_{i+1}$'s solution list if it is not in it ;
      $i := i + 1$ ;
  **endwhile**
  $(G_{new}, \mu_{new}) := (G_i, \nu_i)$ ;
  **return** $(G_{new}, \mu_{new})$.

**Figure 4.2.3 OLDT Resolution for Depth-abstracted Pattern Enumeration**

*Remark.* We should avoid unnecessary depth-abstraction, because checking all the terms at depth $d$ costs much. For example, we need to depth-abstract only the assignments in $\eta$ for the variables in $B_0$ at step (A) in Figure 4.2.3. Moreover, we may throw away the assignments in $\eta$ for the variables in $B\nu$ at step (B), and the assignements in $\nu_i$ for the variables not in $A_{i+1}\mu_{i+1}$ at step (C).

### 4.3 Correctness of the Depth-abstracted Pattern Enumeration

Note that the number of nodes generated from the initial OLDT structure of $(G, \mu)$ is finite, because the number of depth-abstracted atoms is finite. Due to the finiteness, the process of extension under the depth-first from-left-to-right strategy (or any other strategy) always terminates. Moreover, due to the overestimation at step (A), (B) and (C) in OLDT resolution, the information obtained by extending the initial OLDT structures is correct, that is, some supersets of the sets of calling patterns and exiting patterns are obtained. In this section, we will prove the correctness.

The set inclusion ordering between usual terms and depth-abstracted terms is defined by regarding each usual term as the set of all its instances. Then, the set inclusion ordering between substitutions and depth-abstracted substitutions is defined as well as follows: Let $\sigma$ be a substitution, $\mu$ a depth $d$ abstracted substitution, and $\mathcal{V}$ a set of variables. Then $\sigma \subseteq \mu$ *in* $\mathcal{V}$ if and only if $\sigma(X) \subseteq \mu(X)$ for any variable $X$ in $\mathcal{V}$.

Let "$\alpha_1, \alpha_2, \ldots \alpha_n$" be the clause part of a label. (The labels here are possibly intermediate labels, hence $\alpha_1$ might be a call-exit marker.) Then, note that the sequence is divided into the following consecutive segments.

($S_0$) A (possibly empty) sequence of atoms at the leftmost "$A_1, A_2, \ldots, A_{m_1-1}$" ($m_1 > 1$) and

($S_1$) a sequence of the form "$\alpha_{m_1}, A_{m_1+1}, \ldots, A_{m_2-1}$", where $\alpha_{m_1}$ is a call-exit marker and $A_{m_1+1}, \ldots, A_{m_2-1}$ are atoms ($m_2 > m_1$), and

($S_2$) a sequence of the form "$\alpha_{m_2}, A_{m_2+1}, \ldots, A_{m_3-1}$", where $\alpha_{m_2}$ is a call-exit marker and $A_{m_2+1}, \ldots, A_{m_3-1}$ are atoms ($m_3 > m_2$), and

$\vdots$

Now, the set inclusion ordering is defined between labels of OLDT structures and those for depth-abstracted pattern enumeration as follows: Let ("$\alpha_1, \alpha_2, \ldots, \alpha_n$", $\sigma$) be a label of an OLDT structure and ("$\beta_1, \beta_2, \ldots, \beta_n$", $\mu$) be a label of an OLDT structure for depth-abstracted pattern enumeration such that

(a) $\alpha_i$ is a call-exit marker $[\![A_i, \sigma_i]\!]$ if and only if $\beta_i$ is a call-exit marker $[\![A_i, \mu_i]\!]$, and

(b) $\alpha_j$ and $\beta_j$ are identical atoms (modulo renaming of variables) when they are not call-exit markers.

Then, ("$\alpha_1, \alpha_2, \ldots, \alpha_n$", $\sigma$) $\subseteq$ ("$\beta_1, \beta_2, \ldots, \beta_n$", $\mu$) if and only if

($S_0$) $\sigma \subseteq \mu$ in $\mathcal{V}(A_1, A_2, \ldots, A_{m_1-1})$, and

($S_1$) $\sigma_{m_1}\sigma \subseteq \mu_{m_1}\mu$ in $\mathcal{V}(A_{m_1}, A_{m_1+1}, \ldots, A_{m_2-1})$ for the corresponding segments $[\![A_{m_1}, \sigma_{m_1}]\!]$, $A_{m_1+1}, \ldots, A_{m_2-1}$ and $[\![A_{m_1}, \mu_{m_1}]\!]$, $A_{m_1+1}, \ldots, A_{m_2-1}$ ($m_2 > m_1$), and

($S_2$) $\sigma_{m_2}\sigma_{m_1}\sigma \subseteq \mu_{m_2}\mu_{m_1}\mu$ in $\mathcal{V}(A_{m_2}, A_{m_2+1}, \ldots, A_{m_3-1})$ for the corresponding segments $[\![A_{m_2}, \sigma_{m_2}]\!]$, $A_{m_2+1}, \ldots, A_{m_3-1}$ and $[\![A_{m_2}, \mu_{m_2}]\!]$, $A_{m_2+1}, \ldots, A_{m_3-1}$ ($m_3 > m_2$), and

$\vdots$

where $\mathcal{V}(A_{m_i}, A_{m_i+1}, \ldots, A_{m_{i+1}-1})$ denotes the set of all variables in the sequence of atoms "$A_{m_i}, A_{m_i+1}, \ldots, A_{m_{i+1}-1}$".

The depth-abstraction satisfies the following conditions, which play an important role in the correctness of the depth-abstracted pattern enumeration.

**Lemma 4.3.1 (Overestimation of Resolvent)**

Let ("$\alpha_1, \alpha_2, \ldots, \alpha_n$", $\sigma$) be a label of node $u$ in OLDT structure, and ("$\beta_1, \beta_2, \ldots, \beta_n$", $\mu$) be a label of node $v$ in OLDT structure for depth-abstracted pattern enumeration such that ("$\alpha_1, \alpha_2, \ldots, \alpha_n$", $\sigma$) $\subseteq$ ("$\beta_1, \beta_2, \ldots, \beta_n$", $\mu$). Then

(1) When $u$ and $v$ are OLDT resolvable using a definite clause $C$, let $(G', \sigma')$ and $(H', \mu')$ be the respective OLDT resolvents. Then $(G', \sigma') \subseteq (H', \mu')$.

(2) When $u$ and $v$ are OLDT resolvable using solutions $B\tau$ and $B\nu$ such that $B\tau \subseteq B\nu$, let $(G', \sigma')$ and $(H', \mu')$ be the respective OLDT resolvents. Then $(G', \sigma') \subseteq (H', \mu')$.

*Proof.* We will prove the following three sublemmas.

(A) Let "$B_0 :- B_1, B_2, \ldots, B_m$" be a definite clause. When $\alpha_1\sigma$ is unifiable with $B_0$ by an m.g.u. $\theta$ and $\alpha_1\mu$ is unifiable with $B_0$ by an m.g.u. $\eta$, then
("$B_1, B_2, \ldots, B_m, [\![\alpha_1, \sigma]\!], \alpha_2, \ldots, \alpha_n$",$\theta$) $\subseteq$ ("$B_1, B_2, \ldots, B_m, [\![\beta_1, \mu]\!], \beta_2, \ldots, \beta_n$",$[\eta]$).

(B) Let $B\tau$ be an atom and $B\nu$ a depth-abstracted atom such that $B\tau \subseteq B\nu$. When (a fresh variant of) $B\tau$ is an instance of $\alpha_1\sigma$ by an instantiation $\theta$ and (a fresh variant of) $B\nu$ is an instance of $\alpha_1\mu$ by an instantiation $\eta$, then
("$\alpha_2, \ldots, \alpha_n$",$\sigma\theta$) $\subseteq$ ("$\beta_2, \ldots, \beta_n$",$[\mu\eta]$).

(C) When $\alpha_1$ is a call-exit marker $[\![A_1, \sigma_1]\!]$ and $\beta_1$ is a call-exit marker $[\![A_1, \mu_1]\!]$, then
("$\alpha_2, \ldots, \alpha_n$",$\sigma_1\sigma$) $\subseteq$ ("$\beta_2, \ldots, \beta_n$",$[\mu_1\mu]$).

Then, since OLDT resolvents are obtained through successive application of operations at steps (A), (B) and (C), the lemma is obvious.

**Case (A)** : $\alpha_1\sigma\theta = B_0\theta$ and $\alpha_1\mu\eta = B_0\eta$ hold from the condition. Because $\alpha_1\sigma$ is an instance of $\alpha_1\mu$, the result of unification $\alpha_1\sigma\theta = B_0\theta$ is more instantiated than $\alpha_1\mu\eta = B_0\eta$, i.e., $\theta(X)$ is more instantiated than $\eta(X)$ for every variable $X$ in $B_0$, and $\sigma\theta(Y)$ is more instantiated than $\mu\eta(Y)$ for every variable $Y$ in $\alpha_1$. Hence, $\theta \subseteq \eta$ in $\mathcal{V}(B_1, B_2, \ldots, B_m)$, and $\sigma\theta \subseteq \mu\eta$ in $\mathcal{V}(\alpha_1, \alpha_2, \ldots, \alpha_{m_1-1})$, since the variables in the head segment "$\alpha_1, \alpha_2, \ldots, \alpha_{m_1-1}$" but not in $\alpha_1$ are affected by neither $\theta$ nor $\eta$. Then, (A) is obvious from $\eta \subseteq [\eta]$.
**Case (B)** : $\alpha_1\sigma\theta = B\tau$ and $\alpha_1\mu\eta = B\nu$ hold from the condition. Because $B\tau$ is more instantiated than $B\nu$, $\sigma\theta(X)$ is more instantiated than $\mu\eta(X)$ for any variable in $\alpha_1$, i.e., $\sigma\theta \subseteq \mu\eta$ in $\mathcal{V}(\alpha_1)$. Because variables not appearing in $\alpha_1$ but in the new head segment "$\alpha_2, \ldots, \alpha_{m_1-1}$" are affected by neither $\theta$ nor $\eta$, and $\sigma \subseteq \mu$ in those variables holds from the condition, $\sigma\theta \subseteq \mu\eta$ in those variables. Hence $\sigma\theta \subseteq \mu\eta$ in the set of variables in the new head segment. Then, (B) is obvious from $\mu\eta \subseteq [\mu\eta]$.
**Case (C)** : (C) is immediate from the condition.

**Lemma 4.3.2 (Overestimation of Solution)**
Let $u$ be a node in an OLDT structure $S$ labelled with ("$\alpha_1, \alpha_2, \ldots, \alpha_n, G$", $\sigma$), and let $v$ be a node in an OLDT structure $T$ for depth-abstracted pattern enumeration labelled with ("$\beta_1, \beta_2, \ldots, \beta_n, H$", $\mu$) such that
(a) ("$\alpha_1, \alpha_2, \ldots, \alpha_n$", $\sigma$) $\subseteq$ ("$\beta_1, \beta_2, \ldots, \beta_n$", $\mu$), and
(b) the leftmost atoms of $G$ and $H$ are not call-exit markers.
Suppose that there exists an OLDT subrefutation $r$ of ("$\alpha_1, \alpha_2, \ldots, \alpha_n$", $\sigma$) starting from $u$ with its answer substitution $\tau$. Then there exists an extension of $T$ such that
(a) the extension contains an OLDT subrefutation $s$ of ("$\beta_1, \beta_2, \ldots, \beta_n$", $\mu$) for depth-abstracted pattern enumeration starting from $v$ with its answer substitution $\nu$, and
(b) ("$\alpha_1, \alpha_2, \ldots, \alpha_n$", $\tau$) $\subseteq$ ("$\beta_1, \beta_2, \ldots, \beta_n$", $\nu$).

*Proof.* The proof is by induction on the quadruple $(r, T, u, v)$, ordered by the following well-founded ordering : $(r, T, u, v)$ precedes $(r', T', u', v')$ if and only if
(a) $|r| < |r'|$, or
(b) $|r| = |r'|$ and $u$ is not a solution node but $u'$ is, or
(c) $|r| = |r'|$, the nodes $u$ and $u'$ are of a same kind and $v$ is not a solution node but $v'$ is,
where $|r|$ means the length of the corresponding OLD subrefutation. (Note that it does not mean the length of the *OLDT* subrefutation. By replacing every OLDT resolution at a lookup node with its corresponding OLD unit subrefutation, we can always have a corresponding OLD subrefutation.)
**Base Case** : When $|r| = 1$, the lemma is trivial since $r$ is a subrefutation of the null clause.
**Induction Step** : When $|r| > 1$, we consider four cases depending on whether $u$ and $v$ are lookup nodes or not. Let $\alpha_1 \equiv \beta_1 \equiv A$ and $\alpha_2, \alpha_3, \ldots, \alpha_i$ and $\beta_2, \beta_3, \ldots, \beta_i$ be the longest consecutive call-exit markers following $A$.
**Case 1** : When $u$ is a solution node and $v$ is a solution node, let $u'$ be the first node labelled with ("$G', G$", $\sigma'$) and $r'$ be the remaining path of the subrefutation $r$. Suppose $C$ is the definite clause in $P$ used in the first step of the subrefutation $r$. Then $r'$ is a subrefutation of ("$G'$", $\sigma'$) with some answer substitution $\tau'$. By the assumption $\sigma \subseteq \mu$ in $\mathcal{V}(A)$, the node $v$ is also OLDT resolvable with $C$, and the resolvent ("$H', H$", $\mu'$) is such that ("$G', G$", $\sigma'$) $\subseteq$ ("$H', H$", $\mu'$) due to Lemma 4.3.1. Extending $T$ (if necessary) by the OLDT resolution for depth-abstracted pattern enumeration on the node $v$, we can get an OLDT structure $T'$ in which $v$ has a child node $v'$ labelled with ("$H', H$", $\mu'$). Then by the induction hypothesis, we have an extension $T''$ of $T'$ which contains a subrefutation $s'$ of ("$H'$", $\mu'$) starting from $v'$ with its answer substitution $\nu'$ such that ("$G'$", $\tau'$) $\subseteq$ ("$H'$", $\nu'$). The path in $T''$ starting from $v$ and followed by $s'$ constitutes the required subrefutation of ("$\alpha_1, \alpha_2, \ldots, \alpha_n$", $\mu$).

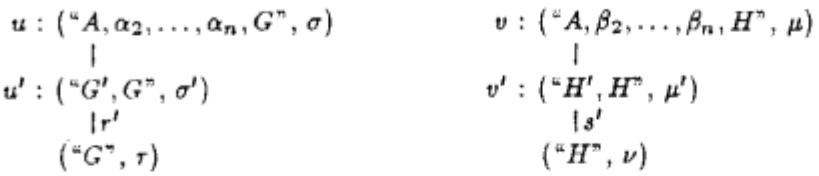



**Figure 4.3.1 Overestimation of Solution (Case 1)**

**Case 2** : When $u$ is a solution node and $v$ is a lookup node, there is a corresponding solution node $v'$ in $T$ labelled with $(\text{“}A, H'\text{”}, \mu')$ such that $A\mu \equiv A\mu'$. Let $r$ be divided as concatenation of two subrefutations $r_1$ and $r_2$ so that $r_1$ is the minimum path containing a subrefutation of $(\text{“}A\text{”}, \sigma)$ with its solution $A\tau_1$. Since $|r_1| \leq |r|$, and $A\sigma \subseteq A\mu \equiv A\mu'$, by the induction hypothesis, we have an extension $T'$ of $T$ such that $T'$ contains a subrefutation of $(\text{“}A\text{”}, \mu')$ which starts from $v'$ with its solution $A\nu_1$ such that $A\tau_1 \subseteq A\nu_1$. By the operation immediately after step (C) in OLDT resolution in Figure 4.2.3, the solution list of $A\mu'(\equiv A\mu)$ includes the solution $A\nu_1$.

Now consider the label $(\text{“}A,\ \beta_2,\ \ldots,\ \beta_n\text{”},\ \mu)$ and the solution $A\nu_1$. Since $(\text{“}A,\ \alpha_2,\ \ldots,\ \alpha_n\text{”},\ \sigma)$ and $A\tau_1$ have an OLDT resolvent $(\text{“}\alpha_{i+1},\ \ldots,\ \alpha_n\text{”},\ \sigma'')$, they also have an OLDT resolvent $(\text{“}\beta_{i+1},\ \ldots,\ \beta_n\text{”},\ \mu'')$ such that $(\text{“}\alpha_{i+1},\ \ldots,\ \alpha_n\text{”},\ \sigma'') \subseteq (\text{“}\beta_{i+1},\ \ldots,\ \beta_n\text{”},\ \mu'')$. This means that $T'$ can be extended (if necessary) to $T''$ by the operation at step (B) in OLDT resolution in Figure 4.2.3, so that the node $v$ has a child node $v''$ labelled with $(\text{“}\beta_{i+1}, \ldots, \beta_n\text{”}, \mu'')$.

Since $r_2$ is a subrefutation of $(\text{“}\alpha_{i+1}, \ldots, \alpha_n\text{”}, \sigma'')$ and $|r_2| < |r|$, again by the induction hypothesis, we have an extension $T'''$ of $T''$ which contains an OLDT subrefutation $s_2$ of $(\text{“}\beta_{i+1} \ldots \beta_n\text{”}, \mu'')$ starting from $v''$ and subsuming $r_2$. The path in $T'''$ starting from $v$ and followed by the subrefutation $s_2$ constitutes that required subrefuation of $(\text{“}\beta_1, \beta_2, \ldots \beta_n\text{”}, \mu)$.

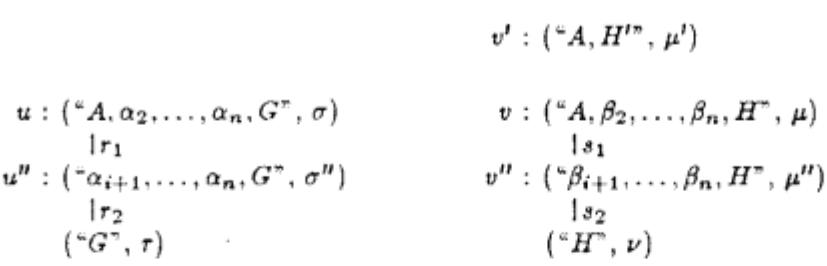



**Figure 4.3.2 Overestimation of Solution (Case 2)**

**Case 3** : When $u$ is a lookup node and $v$ is a solution node, there exists a corresponding solution node $u'$ in $S$ labelled with $(\text{“}A, G'\text{”}, \sigma')$. Let $u''$ be the first node of the subrefutation $r$ labelled with $(\text{“}\alpha_{i+1}, \alpha_2, \ldots, \alpha_n, G\text{”}, \sigma'')$, $r_1$ be the one-step subrefutation of $(\text{“}A\text{”}, \sigma)$ starting from $u$ ending in $u''$, and $r_2$ be the remaining path of the subrefutation $r$. Because $A\sigma' \equiv A\sigma \subseteq A\mu$ and there is a refutation of $(\text{“}A\text{”}, \sigma')$ in $S$, by the induction hypothesis, we have an extension $T'$ of $T$ which contains a subrefutation $s_1$ of $(\text{“}A\text{”}, \mu)$ starting from $v$ ending in a node $v''$ labelled with $(\text{“}\beta_{i+1}, \ldots, \beta_n, H\text{”}, \mu'')$ such that $(\text{“}\alpha_{i+1}, \ldots, \alpha_n\text{”}, \sigma'')$ $\subseteq (\text{“}\beta_{i+1}, \ldots, \alpha_n\text{”}, \mu'')$. By the induction hypothesis, we have an extension $T''$ of $T'$ which contains a subrefutation $s_2$ of $(\text{“}\beta_{i+1}, \ldots, \beta_n\text{”}, \mu'')$, starting from $v''$ and subsuming $r_2$. The path in $T''$ starting from $v$ and followed by $s_2$ constitutes the required subrefutation of $(\text{“}\beta_1, \beta_2, \ldots, \beta_n\text{”}, \mu)$.

$u'$ : $(\ulcorner A, G'\urcorner, \sigma')$

$u$ : $(\ulcorner A, \alpha_2, \ldots, \alpha_n, G\urcorner, \sigma)$ — $v$ : $(\ulcorner A, \beta_2, \ldots, \beta_n, H\urcorner, \mu)$

$|r_1$ — $|s_1$

$u''$ : $(\ulcorner \alpha_{i+1}, \ldots, \alpha_n, G\urcorner, \sigma'')$ — $v''$ : $(\ulcorner \beta_{i+1}, \ldots, \beta_n, H\urcorner, \mu'')$

$|r_2$ — $|s_2$

$(\ulcorner G\urcorner, \tau)$ — $(\ulcorner H\urcorner, \nu)$

**Figure 4.3.3 Overestimation of Solution (Case 3)**

**Case 4** : When $u$ is a lookup node and $v$ is a lookup node, there exist a corresponding solution node $u'$ in $S$ labelled with $(\ulcorner A, G'\urcorner, \sigma')$, and a corresponding solution node $v'$ in $T$ labelled with $(\ulcorner A, H'\urcorner, \mu')$. Because $A\sigma' \equiv A\sigma \subseteq A\mu \equiv A\mu'$ and there is a subrefutation of $(\ulcorner A\urcorner, \sigma')$ starting from $u'$, by the induction hypothesis, we have an extension of $T'$ of $T$ which contains a subrefutation of $(\ulcorner A\urcorner, \mu')$. By the operation immediately after step (C) in OLDT resolution in Figure 4.2.3, the solution list of $A\mu' \equiv A\mu$ in $T'$ includes the solution $A\nu_1$.

Let $u''$ be the first node of the subrefutation $r$ labelled with $(\ulcorner \alpha_{i+1}, \alpha_2, \ldots, \alpha_n, G\urcorner, \sigma'')$, starting from $u$ ending in $u''$, and $r_2$ be the remaining path of the subrefutation $r$. Now consider $(\ulcorner A, \alpha_2, \ldots, \alpha_n\urcorner, \mu)$ and the solution $A\nu_1$. Since $(\ulcorner A, \alpha_2, \ldots, \alpha_n\urcorner, \sigma) \subseteq (\ulcorner A, \beta_2, \ldots, \beta_n\urcorner, \mu)$, $A\tau_1 \subseteq A\nu_1$ and $(\ulcorner A, \alpha_2, \ldots, \alpha_n\urcorner, \sigma)$ and $A\tau_1$ has an OLDT resolvent $(\ulcorner \alpha_{i+1}, \ldots, \alpha_n\urcorner, \sigma'')$, they also have an OLDT resolvent for depth-abstracted pattern enumeration $(\ulcorner \beta_{i+1}, \ldots, \beta_n\urcorner, \mu'')$ such that $(\ulcorner \alpha_{i+1}, \ldots, \alpha_n\urcorner, \sigma'') \subseteq (\ulcorner \beta_{i+1}, \ldots, \beta_n\urcorner, \mu'')$. This means that $T'$ can be extended (if necessary) to $T''$ by the operation at step (B) in OLDT resolution in Figure 4.2.3, so that the node $v$ has a child node $v''$ labelled with $(\ulcorner \beta_{i+1}, \ldots, \beta_n\urcorner, \mu'')$.

Since $r_2$ is a subrefutation of $(\ulcorner \alpha_{i+1}, \ldots, \alpha_n\urcorner, \sigma'')$ and $|r_2| < |r|$, again by the induction hypothesis, we have an extension $T'''$ of $T''$ which contains an OLDT subrefutation $s_2$ of $(\ulcorner \beta_{i+1}, \ldots \beta_n\urcorner, \mu'')$ starting from $v''$ and subsuming $r_2$. The path in $T'''$ starting from $v$ and followed by the subrefutation $s_2$ constitutes that required subrefuation of $(\ulcorner \beta_1, \beta_2, \ldots \beta_n\urcorner, \mu)$.

$u'$ : $(\ulcorner A, G'\urcorner, \sigma')$ — $v'$ : $(\ulcorner A, H'\urcorner, \mu')$

$u$ : $(\ulcorner A, \alpha_2, \ldots, \alpha_n, G\urcorner, \sigma)$ — $v$ : $(\ulcorner A, \beta_2, \ldots, \beta_n, H\urcorner, \mu)$

$|r_1$ — $|s_1$

$u''$ : $(\ulcorner \alpha_{i+1}, \ldots, \alpha_n, G\urcorner, \sigma'')$ — $v''$ : $(\ulcorner \beta_{i+1}, \ldots, \beta_n, H\urcorner, \mu'')$

$|r_2$ — $|s_2$

$(\ulcorner G\urcorner, \tau)$ — $(\ulcorner H\urcorner, \nu)$

**Figure 4.3.4 Overestimation of Solution (Case 4)**

**Theorem 4 (Correctness of the Depth-abstracted Pattern Enumeration)**
Let $\overline{C}_{local}(G\mu)$, $\overline{C}_{global}(G\mu)$, $\overline{\mathcal{E}}_{local}(G\mu)$, $\overline{\mathcal{E}}_{global}(G\mu)$ be the sets defined for the OLDT structure of $(G, \mu)$ for depth-abstracted pattern enumeration (with the depth-first from-left-to-right strategy or any other strategy) as follows:

$\overline{C}_{local}(G\mu)$ : the set of all calling patterns at local success of $G\mu$,

$\overline{C}_{global}(G\mu)$ : the set of all calling patterns at global success of $G\mu$,

$\overline{\mathcal{E}}_{local}(G\mu)$ : the set of all exiting patterns at local success of $G\mu$,

$\overline{\mathcal{E}}_{global}(G\mu)$ : the set of all exiting patterns at global success of $G\mu$.

Then

(a) $\overline{\mathcal{C}}_{local}(G\mu) \supseteq \mathcal{C}_{local}(G\mu)$,
(b) $\overline{\mathcal{C}}_{global}(G\mu) \supseteq \mathcal{C}_{global}(G\mu)$,
(c) $\overline{\mathcal{E}}_{local}(G\mu) \supseteq \mathcal{E}_{local}(G\mu)$,
(d) $\overline{\mathcal{E}}_{global}(G\mu) \supseteq \mathcal{E}_{global}(G\mu)$.

*Proof.* We only need to show that, when a negative clause $G\sigma$, such that $G\sigma \subseteq G\mu$, has an OLD refutation with its solution $G\tau$, any extension of the initial OLDT structure of $(G, \mu)$ for depth-abstracted pattern enumeration can be further extended to contain an OLDT refutation of $(G, \mu)$ with its solution $G\nu$ such that $G\tau \subseteq G\nu$. But it is obvious from the completeness of OLDT resolution (Theorem 2.1), and Lemma 4.3.2 by letting $u$, $v$ be the root nodes and $G$, $H$ be the empty sequences.

## 5. Type Inference by Abstract Hybrid Interpretation

Suppose that a top-level goal "*multiply*$(X, Y, Z)$" is executed with their arguments instantiated to any terms, where *multiply* and *add* are defined by

```
multiply(0,Y,0).
multiply(suc(X),Y,Z) :- multiply(X,Y,W), add(Y,W,Z).
add(0,Y,Y).
add(suc(X),Y,suc(Z)) :- add(X,Y,Z).
```

Then the third argument of *multiply* is always a number when the execution succeeds. How can we show it mechanically?

In this section, we will reformulate our work [4] from the point of view in Section 2.2. We are mainly concerned with exiting patterns at global success, not calling patterns. Unlike the depth-abstracted pattern enumeration in Section 4, we need to keep the m.g.u. at OLDT resolution in each call-exit marker.

### 5.1 Type Inference

We introduce **type** construct into Prolog to separate definite clauses defining data structures from other clauses defining procedures, e.g.,

```
type.
   list([ ]).
   list([X|L]) :- list(L).
end.
```

**type** defines a unary relation by definite clauses. The head of definite clause takes a term defining a data structure as its argument, either a constant $b$ called a *bottom element* or a term of the form $c(t_1, t_2, \ldots, t_n)$ where $c$ is called a *constructor*. The body shows type conditions on proper subterms of the argument.

Here notice the sets of terms prescribed by type predicates. The set of all terms $t$ such that the execution of $p(t)$ succeeds without instantiating the variables in it is called the *type* of $p$, and denoted by $\underline{p}$.

*Example 5.1* Let the definition of a type *num* be

```
type.
```

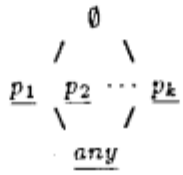
```
    num(0).
    num(suc(X)) :- num(X).
end.
```

Then $\underline{num}$ is a set $\{0, suc(0), suc(suc(0)), \ldots\}$. Note that terms in each type is not necessarily ground, since the execution of $p(t)$ sometimes succeeds without instantiation of variables in $t$. For example, we include $[X]$ in $\underline{list}$, since the execution of $list([X])$ succeeds without instantiation of the variable $X$.

Suppose that there are $k$ type predicates $p_1, p_2, \ldots, p_k$ defined using the **type** construct such that bottom elements and constructors for each $p_i$ are disjoint, hence $\underline{p_1}, \underline{p_2}, \ldots, \underline{p_k}$ are disjoint. A *type* is one of the following $k+2$ sets of terms.

$\underline{any}$ : the set of all terms,
$\underline{p_1}$ : the set of all terms satisfying the definition of type predicate $p_1$,
$\underline{p_2}$ : the set of all terms satisfying the definition of type predicate $p_2$,
$\vdots$
$\underline{p_k}$ : the set of all terms satisfying the definition of type predicate $p_k$,
$\underline{\emptyset}$ : the empty set.

Types are ordered by the instantiation ordering $\preceq$ depicted below.




Again, the instantiation ordering is just the reverse of the set inclusion ordering here.

A *type substitution* $\mu$ is an expression of the form

$$< X_1 \Leftarrow \underline{t_1}, X_2 \Leftarrow \underline{t_2}, \ldots, X_l \Leftarrow \underline{t_l} >,$$

where $\underline{t_1}, \underline{t_2}, \ldots, \underline{t_l}$ are types. The type assigned to variable $X$ by type substitution $\mu$ is denoted by $\mu(X)$. We stipulate that a type substitution assigns $\underline{any}$, the minimum element w.r.t. the instantiation ordering, to variable $X$ when $X$ is not in the domain of the type substitution explicitly. Hence the empty type substitution $<>$ assigns $\underline{any}$ to every variable.

Let $A$ be an atom in the body of some clause in $P \cup \{G\}$ and $\mu$ be a type substitution of the form

$$< X_1 \Leftarrow \underline{t_1}, X_2 \Leftarrow \underline{t_2}, \ldots, X_l \Leftarrow \underline{t_l} >.$$

Then $A\mu$ is called a *type-abstracted atom*, and denotes the set of all atoms obtained by replacing each variable $X_i$ in $A$ with a term in $\underline{t_i}$. Two type-abstracted atoms $A\mu$ and $B\nu$ are said to be *unifiable* when $A\mu \cap B\nu \neq \emptyset$. A list of type-abstracted atoms $[A_1\mu_1, A_2\mu_2, \ldots, A_n\mu_n]$ denotes the set union $\cup_{i=1}^{n} A_i\mu_i$. Similarly, $G\mu$ (or the pair $(G, \mu)$) is called a *type-abstracted negative clause*, and denotes the set of negative clauses obtained by replacing each $X_i$ in $G$ with a term in $\underline{t_i}$. When $G$ is of the form "$A_1$, $A_2$, ..., $A_n$", the type-abstracted atom $A_1\mu$ is called the leftmost depth-abstracted atom of $G\mu$.

The *type inference* w.r.t. $G\mu$ is the problem to compute

(a) some list of type-abstracted atoms which is a superset of $\mathcal{C}_{local}(G\mu)$,
(b) some list of type-abstracted atoms which is a superset of $\mathcal{C}_{global}(G\mu)$,
(c) some list of type-abstracted atoms which is a superset of $\mathcal{E}_{local}(G\mu)$, and
(d) some list of type-abstracted atoms which is a superset of $\mathcal{E}_{global}(G\mu)$.

## 5.2 Abstract Hybrid Interpretation for Type Inference

### 5.2.1 OLDT Structure for Type Inference

A *search tree*, a *solution table* and an *association for type inference* are defined in the same way as depth-abstracted pattern enumeration, except that call-exit markers are of the form $[A, \mu, \eta]$. (For brevity, we will sometimes omit the term "for type inference" hereafter in Section 5.) An *OLDT structure for type inference* is a triple of a search tree, a solution table and an association. The relation between a node and its child nodes is specified by *OLDT resolution for type inference* in Section 5.2.3.

### 5.2.2 Overestimation of Data Types

We need to overestimate the sets of calling patterns and exiting patterns somehow by manipulating type-abstracted atoms, as we did for the depth-abstracted pattern enumeration in Section 4. Again, we would like to do it by specifying the corresponding operations for type inference at steps (A), (B) and (C) in Figure 2.2.1. As for the type inference problem, we need to consider the following situation. Let $A$ be an atom, $X_1, X_2, \ldots, X_k$ all the variables in $A$, $\mu$ a type substitution of the form

$< X_1 \Leftarrow \underline{t_1}, X_2 \Leftarrow \underline{t_2}, \ldots, X_k \Leftarrow \underline{t_k} >$,

$B$ an atom, $Y_1, Y_2, \ldots, Y_l$ all the variables in $B$, and $\nu$ a type substitution of the form

$< Y_1 \Leftarrow \underline{s_1}, Y_2 \Leftarrow \underline{s_2}, \ldots, Y_l \Leftarrow \underline{s_l} >$.

Then

(a) How can we know whether there is an atom $A\sigma = B\tau$ in $A\mu \cap B\nu$?

(b) If there is such an atom, what terms are expected to be assigned to each $Y_j$ by $\tau$?

(These problems were easily solved by unification and depth-abstraction for the depth-abstracted pattern enumeration. They are not very easy for the type inference.)

*Example 5.2.2.1* The follwoing two type-abstracted atoms

$p(L_1, suc(N_1)) < L_1 \Leftarrow \underline{list}, N_1 \Leftarrow \underline{num} >$,

$p([X_2|L_2], N_2) < X_2 \Leftarrow \underline{any}, L_2 \Leftarrow \underline{any}, N_2 \Leftarrow \underline{any} >$

are unifiable. The common atom is of the form $p([X|L], suc(N))$, and terms in $\underline{any}$, $\underline{list}$ and $\underline{number}$ must be assigned to variables $X_2$, $L_2$ and $N_2$.

#### (1) Overestimation of Unifiability

When two type-abstracted atoms $A\mu$ and $B\nu$ are unifiable, two atoms $A$ and $B$ must be unifiable in the usual sense. Let $\eta$ be an m.g.u of $A$ and $B$ of the form

$< X_1 \Leftarrow t_1, X_2 \Leftarrow t_2, \ldots, X_k \Leftarrow t_k, Y_1 \Leftarrow s_1, Y_2 \Leftarrow s_2, \ldots, Y_l \Leftarrow s_l >$.

If we can overestimate the type assigned to each occurrence of $Z$ in $t_i$ from the type substitution $\mu$, and that of $Z$ in $s_j$ from the type substitution $\nu$, we can overestimate the type assigned to the variable $Z$ by taking the join of these types w.r.t. the instantiation ordering for all occurrences of $Z$. If it is the emptyset $\emptyset$ for some variable, we can't expect that there exist an atom $A\sigma = B\tau$ in $A\mu \cap B\nu$.

When a term $t$ containing occurrences of variable $Z$ is instantiated to a term in $\underline{t}$, we denote a type containing all instances of some occurrence of $Z$ by $Z/ < t \Leftarrow \underline{t} >$, and compute

as follows:

$$Z/ < t \Leftarrow \underline{t} >= \begin{cases} \underline{t}, & \text{when } t \text{ is } Z; \\ \underline{any}, & \text{when } \underline{t} \text{ is } \underline{any}; \\ \overline{Z/ < t_i \Leftarrow \underline{t_i} >}, & \text{when } t \text{ is of the form } c(t_1, t_2, \ldots, t_n),\ Z \text{ is in } t_i, \\ & t \text{ includes a type } \underline{p}, \\ & c \text{ is a constructor of the data type } p \text{ and} \\ & \underline{t_i} \text{ is a type set assigned to the } i\text{-th argument } t_i; \\ \emptyset, & \text{otherwise.} \end{cases}$$

*Example 5.2.2.2* Let $t$ be $[X|L]$ and $\underline{t}$ be $\underline{list}$. Then
$X/ < [X|L] \Leftarrow \underline{list} >= \underline{any}$, $L/ < [X|L] \Leftarrow \underline{list} >= \underline{list}$.
Let $t$ be $[X|L]$ and $\underline{t}$ be $\underline{num}$. Then
$X/ < [X|L] \Leftarrow \underline{num} >= \emptyset$, $L/ < [X|L] \Leftarrow \underline{num} >= \emptyset$.

If we would like to check the unifiability of type-abstracted atoms $A\mu$ and $B\nu$ exactly, i.e., would like a procedure returning true *if and only if* they are unifiable, we can check it using the estimation $Z/ < t[Z] \Leftarrow \underline{t} >$. The exact unifiability check, however, takes more computational time than that of depth-abstracted atoms, because it can't be reduced to the unifiability of terms. But, if we just would like overestimate the unifiability, i.e., would like a procedure returning true *if* they are unifiable, we may use the unifiability check of $A$ and $B$ instead of the more time-consuming one.

*Example 5.2.2.3* We can check the unifiability of $p(X) < X \Leftarrow \underline{list} >$ and $p(suc(Y)) < Y \Leftarrow \underline{list} >$ by computing $Z/ < Z \Leftarrow \underline{list} >= \underline{list}$ and $Z/ < suc(Z) \Leftarrow \underline{list} >= \emptyset$. If we use the unifiability check of $p(X)$ and $p(suc(Y))$ instead of the exact one, we would consider these type-abstracted atoms unifiable.

### (2) One Way Propagation of Type Substitutions

Recall the situation we are considering. First, we will restrict our attention to the case when $\nu = <>$. Suppose that there is an atom $B\tau$ in $A\mu \cap B <>$. Then, what terms are expected to be assigned to variables in $B$ by $\tau$?

As has been just shown, we can overestimate the type assigned to the variable $Z$ due to the type substitution $\mu$. By collecting these type assignments for all variables, we can overestimate the type substitution $\lambda$ for the variables in $t_1, t_2, \ldots, t_n$. Then, if we can overestimate the type assigned to $s_j$, from the type substitution $\lambda$ obtained above, we can obtain the type substitution $\nu'$
$< Y_1 \Leftarrow s'_1, Y_2 \Leftarrow s'_2, \ldots, Y_l \Leftarrow s'_l >$
by collecting the types for all variables $Y_1, Y_2, \ldots, Y_l$ in $B$.

When each variable $Z$ in term $s$ is instantiated to a term in $\lambda(Z)$, we denote a type containing all instances of $s$ by $s/\lambda$, and compute it as follows:

$$s/\lambda = \begin{cases} \emptyset, & \lambda(Z) = \emptyset \text{ for some } Z \text{ in } s; \\ \lambda(Z), & \text{when } s \text{ is a variable } Z; \\ \underline{p}, & \text{when } s \text{ is a bottom element } b \text{ of a data type } p \text{ or} \\ & \text{when } s \text{ is of the form } c(s_1, s_2, \ldots, s_n), \\ & c \text{ is a constructor of a data type } p \text{ and} \\ & s_1/\lambda, s_2/\lambda, \ldots, s_n/\lambda \text{ satisfy the type conditions;} \\ \underline{any}, & \text{otherwise.} \end{cases}$$

*Example 5.2.2.4* Let $s$ be $[X|L]$ and $\lambda$ be $< X \Leftarrow \underline{any}, L \Leftarrow \underline{list} >$. Then
$s/\lambda = \underline{list}$.
Let $s$ be $[X|L]$ and $\lambda$ be $< X \Leftarrow \underline{any}, L \Leftarrow \underline{any} >$. Then
$s/\lambda = \underline{any}$.

Let $A$, $B$ be atoms, $\mu$ a type substitution for the variables in $A$, and $\eta$ an m.g.u. of $A$ and $B$. The type substitution for the variables in $B$, that is obtaind from $\mu$ and $\eta$ using $Z/ < t[Z] \Leftarrow \underline{t} >$ and $s/\lambda$ above, is denoted by $propagate(\mu, \eta)$. (Note that $propagate(\mu, \eta)$ depends on just $\mu$ and $\eta$.)

### (3) Overestimation of Type Substitutions

As for the operation at step (A) for type inference, we can adopt the one way type propagation

$$propagate(\mu, \eta),$$

since the destination side type substitution is <>. As for the operations at steps (B) and (C) for type inference where the destination side type substitution is not necessarily <>, we can adopt the join w.r.t. the instantiation ordering

$$\mu \vee propagate(\nu, \eta)$$

i.e., variable-wise join of the type assigned by the previous type substitution $\mu$ and the one by the one-way propagation $propagate(\nu, \eta)$.

## 5.2.3 OLDT Resolution for Type Inference

The relation between a node and its child nodes of a search tree is specified by *OLDT resolution for type inference* as follows:

A node of OLDT structure $(Tr, Tb, As)$ labelled with ("$\alpha_1, \alpha_2, \ldots, \alpha_n$", $\mu$) is said to be *OLDT resolvable* ($n \geq 1$) when $\mu(X) \neq \emptyset$ for any variable $X$ and $\alpha_1$ satisfies either of the following conditions.

(a) The node is a terminal solution node of $Tr$, and there is some definite clause "$B_0$ :- $B_1, B_2, \ldots, B_m$" ($m \geq 0$) in program $P$ such that $\alpha_1$ and $B_0$ are unifiable, say by an m.g.u. $\eta$.

(b) The node is a lookup node of $Tr$, and there is some type-abstracted atom $B\nu$ in the associated solution list of the lookup node such that $\alpha_1$ and $B$ are variants of each other. Let $\eta$ be the renaming of $B$ to $\alpha_1$.

The precise algorithm of OLDT resolutuion for type inference is shown in Figure 5.2.3.1. Note that the operations at steps (A), (B) and (C) are modified.

A node labelled with ("$\alpha_1, \alpha_2, \ldots, \alpha_n$", $\mu$) is a lookup node when the type-abstracted atom $\alpha_1\mu$ is a key in the solution table, and is a solution node otherwise.

The *initial OLDT structure*, *immediate extension of OLDT structure*, *extension of OLDT structure*, *answer substitution of OLDT refutation* and *solution of OLDT refutation* are defined in the same way as in Section 2.2.

OLDT-resolve(("$\alpha_1, \alpha_2, \ldots, \alpha_n$", $\mu$) : label) : label ;
  $i := 0$;
  **case**
    **when** a solution node is OLDT resolved with "$B_0$ :- $B_1, B_2, \ldots, B_m$" in $P$
      let $\eta$ be the m.g.u. of $\alpha_1$ and $B_0$ ;
      let $G_0$ be a negative clause "$B_1, B_2, \ldots, B_m, [\![\alpha_1, \mu, \eta]\!], \alpha_2, \ldots, \alpha_n$";
      let $\nu_0$ be $propagate(\mu, \eta)$ ; — (A)
    **when** a lookup node is OLDT resolved with "$B\nu$" in $Tb$
      let $\eta$ be the renaming of $B$ to $\alpha_1$ ;
      let $G_0$ be a negative caluse "$\alpha_2, , \ldots, \alpha_n$" ;
      let $\nu_0$ be $\mu \vee propagate(\nu, \eta)$ ; — (B)
  **endcase**
  **while** the leftmost of $G_i$ is a call-exit marker $[\![A_{i+1}, \mu_{i+1}, \eta_{i+1}]\!]$ **do**
      let $G_{i+1}$ be $G_i$ other than the leftmost call-exit marker ;
      let $\nu_{i+1}$ be $\mu_{i+1} \vee propagate(\nu_i, \eta_{i+1})$ ; — (C)
      add $A_{i+1}\nu_{i+1}$ to the last of $A_{i+1}\mu_{i+1}$'s solution list if it is not in it ;
      $i := i + 1$ ;
  **endwhile**
  $(G_{new}, \mu_{new}) := (G_i, \nu_i)$ ;
  **return** $(G_{new}, \mu_{new})$.

**Figure 5.2.3.1 OLDT Resolution for Type Inference**

*Example 5.2.3* Let *multiply* and *add* be predicates as before defined by

```
multiply(0,Y,0).
multiply(suc(X),Y,Z) :- multiply(X,Y,W), add(Y,W,Z).
add(0,Y,Y).
add(suc(X),Y,suc(Z)) :- add(X,Y.Z).
```

Then the type inference of $multiply(X_0, Y_0, Z_0)$ proceeds as follows:

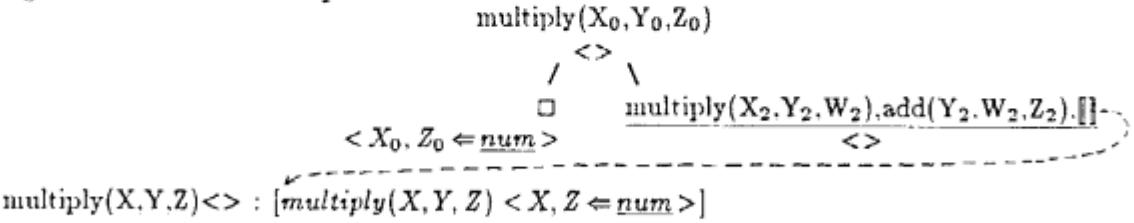
First, the initial OLDT structure below is generated. The root node is a solution node.

multiply($X_0$,$Y_0$,$Z_0$)
<>

multiply(X,Y,Z)<> : [ ]

**Figure 5.2.3.2 Type Inference by Hybrid Interpretation at Step 1**

Secondly, the root node ("$multiply(X_0, Y_0, Z_0)$",<>) is OLDT resolved using the program. The left child node gives a solution $multiply(X_0, Y_0, Z_0) < X_0, Z_0 \Leftarrow \underline{num} >$. The right child node is a lookup node.




multiply(X,Y,Z)<> : [$multiply(X, Y, Z) < X, Z \Leftarrow \underline{num} >$]

**Figure 5.2.3.3 Type Inference by Hybrid Interpretation at Step 2**

Thirdly, the lookup node is OLDT resolved using the solution table. The generated child node is a solution node.

Fourthly, the solution node is OLDT resolved further using the program. The left child node gives two solutions $add(Y_2, W_2, Z_2) < Y_2, W_2, Z_2 \Leftarrow \underline{num} >$ and $multiply(X_0, Y_0, Z_0) < X_0, Y_0, Z_0 \Leftarrow \underline{num} >$. The right child node is a lookup node.

Fifthly, the lookup node is OLDT resolved using the solution table. The child node gives a new solution $multiply(X_0, Y_0, Z_0) < X_0, Y_0, Z_0 \Leftarrow \underline{num} >$.

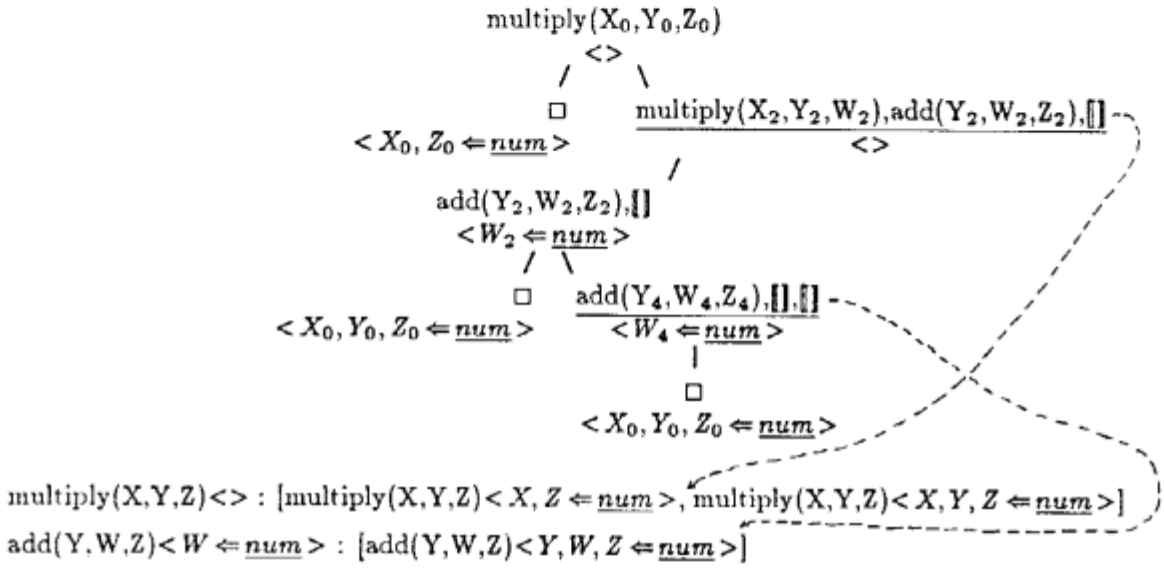



**Figure 5.2.3.4 Type Inference by Hybrid Interpretation at Step 5**

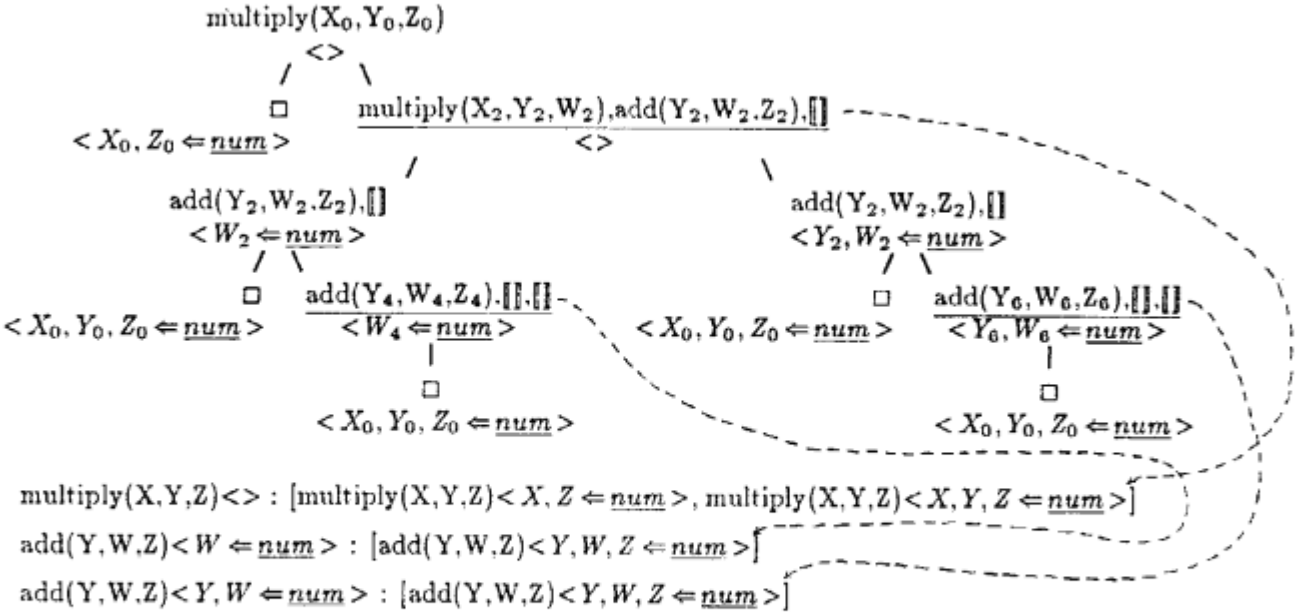



**Figure 5.2.3.5 Type Inference by Hybrid Interpretation at Step 8**

Sixthly, the first lookup node is OLDT resolved using the new solution. The generated child node is a solution node.

Seventhly, the generated solution node is OLDT resolved using the program. The left child node gives a new solution $add(Y_4, W_4, Z_4) < Y_4, W_4, Z_4 \Leftarrow \underline{num} >$ and an already existing solution $multiply(X_0, Y_0, Z_0) < X_0, Y_0, Z_0 \Leftarrow \underline{num} >$. The right child node is a lookup node.

Lastly, the lookup node is OLDT resolved using the solution table. Because the generated child node gives no new solution, the extension process stops.

This problem is not so trivial as one might think at first glance. For example, Suppose that the predicate *multiply* is defined by

multiply(0,Y,0).

multiply(suc(X),Y,Z) :- multiply(X,Y,W), add(W,Y,Z).

by exchanging the first and the second arguments of *add*. Then, one of the exit pattern of $multiply\ (X, Y, Z) <>$ is $multiply(X, Y, Z) < X \Leftarrow \underline{num} >$, hence, we can't conclude that the third argument is a number. For example, $multiply(suc(0), Y, Y)$ succeeds for any $Y$.

### 5.3 Correctness of the Type Inference

The information obtained by extending the initial OLDT structures for type inference is correct, that is, some supersets of the sets of calling patterns and exiting patterns are obtained. We can prove the correctness in the same way as the depth-abstracted pattern enumeration except a few modifications.

The set inclusion ordering between labels of OLDT structures and those for type inference is defined in a slightly different way as follows: $(\text{“}\alpha_1, \alpha_2, \ldots, \alpha_n\text{”}, \sigma) \subseteq (\text{“}\beta_1, \beta_2, \ldots, \beta_n\text{”}, \mu)$ if and only if

($S_0$) $\sigma \subseteq \mu$ in $\mathcal{V}(A_1, A_2, \ldots, A_{m_1-1})$, and

($S_1$) $\sigma_{m_1}\sigma \subseteq \mu_{m_1} \vee propagate(\mu, \eta)$ in $\mathcal{V}(A_{m_1}, A_{m_1+1}, \ldots, A_{m_2-1})$ for the corresponding segments $[A_{m_1}, \sigma_{m_1}]$, $A_{m_1+1}$, …, $A_{m_2-1}$ and $[A_{m_1}, \mu_{m_1}, \eta]$, $A_{m_1+1}$, …, $A_{m_2-1}$ $(m_2 > m_1)$, and

($S_2$) $\sigma_{m_2}\sigma_{m_1}\sigma \subseteq \mu_{m_2} \vee propagate(\mu_{m_1} \vee propagate(\mu, \eta), \eta_1)$ in $\mathcal{V}(A_{m_2}, A_{m_2+1}, \ldots, A_{m_3-1})$ for the corresponding segments $[A_{m_2}, \sigma_{m_2}]$, $A_{m_2+1}$, …, $A_{m_3-1}$ and $[A_{m_2}, \mu_{m_2}, \eta_{m_2}]$, $A_{m_2+1}$, …, $A_{m_3-1}$ $(m_3 > m_2)$, and

⋮

where $\mathcal{V}(A_{m_i}, A_{m_i+1}, \ldots, A_{m_{i+1}-1})$ denotes the set of all variables in the sequence of atoms $\text{“}A_{m_i}, A_{m_i+1}, \ldots, A_{m_{i+1}-1}\text{”}$.

Then the type inference satisfies the same condition as in Section 4.3.

**Lemma 5.3.1 (Overestimation of Resolvent)**

Let $(\text{“}\alpha_1, \alpha_2, \ldots, \alpha_n\text{”}, \sigma)$ be a label of OLDT structure and $(\text{“}\beta_1, \beta_2, \ldots, \beta_n\text{”}, \mu)$ be a label of OLDT structure for type inference such that $(\text{“}\alpha_1, \alpha_2, \ldots, \alpha_n\text{”}, \sigma) \subseteq (\text{“}\beta_1, \beta_2, \ldots, \beta_n\text{”}, \mu)$. Then

(1) When $u$ and $v$ are OLDT resolvable using a definite clause $C$, let $(G', \sigma')$ and $(H', \mu')$ be the respective OLDT resolvents. Then $(G', \sigma') \subseteq (H', \mu')$.

(2) When $u$ and $v$ are OLDT resolvable using solutions $B\tau$ and $B\nu$ such that $B\tau \subseteq B\nu$, let $(G', \sigma')$ and $(H', \mu')$ be the respective OLDT resolvents. Then $(G', \sigma') \subseteq (H', \mu')$.

*Proof.* Again, we will prove the following three sublemmas.

(A) Let $\text{“}B_0 :- B_1, B_2, \ldots, B_m\text{”}$ be a definite clause. When $\alpha_1\sigma$ is unifiable with $B_0$ by an m.g.u. $\theta$ and $\alpha_1$ is unifiable with $B_0$ by an m.g.u. $\eta$, then

$(\text{“}B_1, B_2, \ldots, B_m, [\alpha_1, \sigma]\text{”}, \alpha_2, \ldots, \alpha_n\text{”}, \theta)$
$\subseteq (\text{“}B_1, B_2, \ldots, B_m, [\beta_1, \mu, \eta], \beta_2, \ldots, \beta_n\text{”}, propagate(\mu, \eta))$.

(B) Let $B\tau$ be an atom and $B\nu$ a type-abstracted atom such that $B\tau \subseteq B\nu$. When (a fresh variant of) $B\tau$ is an instance of $\alpha_1\sigma$ by an instantiation $\theta$, and $\alpha_1$ is a variant of $B$ by a renaming $\eta$, then
$(\text{“}\alpha_2, \ldots, \alpha_n\text{”}, \sigma\theta) \subseteq (\text{“}\beta_2, \ldots, \beta_n\text{”}, \mu \vee propagate(\nu, \eta))$.

(C) When $\alpha_1$ is a call-exit marker $[A_1, \sigma_1]$ and $\beta_1$ is a call-exit marker $[A_1, \mu_1, \eta_1]$, then
$(\text{“}\alpha_2, \ldots, \alpha_n\text{”}, \sigma_1\sigma) \subseteq (\text{“}\beta_2, \ldots, \beta_n\text{”}, \mu_1 \vee propagate(\mu, \eta_1))$.

**Case (A)** : Since $\sigma \subseteq \mu$ in $\mathcal{V}(\alpha_1)$ from the condition, $\theta \subseteq propagate(\mu, \eta)$ in $\mathcal{V}(B_0)$, hence in $\mathcal{V}(\text{“}B_1, B_2, \ldots, B_m\text{”})$, holds from the meaning of *propagate*. By appying the same discussion for $B_0\theta$ and $B_0 propagate(\mu, \eta)$, $\sigma\theta \subseteq \mu \vee propagate(propagate(\mu, \eta), \eta)$ in $\mathcal{V}(\alpha_1)$. Since the variables in the head segment $\text{“}\alpha_1, \alpha_2, \ldots, \alpha_{m_1-1}\text{”}$ but not appearing in $\alpha_1$ are affected by neither $\sigma\theta$ nor $\eta$, it holds in $\mathcal{V}(\text{“}\alpha_1, \alpha_2, \ldots, \alpha_{m_1-1}\text{”})$.

**Case (B)** : By appying the same discussion for $B\tau$ and $B\nu$, $\sigma\theta \subseteq propagate(\nu, \eta)$ in $\mathcal{V}(\alpha_1)$ from the condition $\tau \subseteq \nu$ in $\mathcal{V}(B)$. In addition, $\sigma\theta \subseteq \mu$ in $\mathcal{V}(\text{“}\alpha_1, \alpha_2, \ldots, \alpha_{m_1-1}\text{”})$ holds from the condition. Hence, $\sigma\theta \subseteq \mu \vee propagate(\nu, \eta)$ in $\mathcal{V}(\alpha_1)$. Since the variables in the new head segment $\text{“}\alpha_2, \ldots, \alpha_{m_1-1}\text{”}$ but not appearing in $\alpha_1$ are affected by neither $\sigma\theta$ nor $\eta$, it holds in $\mathcal{V}(\text{“}\alpha_2, \ldots, \alpha_{m_1-1}\text{”})$.

**Case (C)** : (C) is immediate from the condition.

**Lemma 5.3.2 (Overestimation of Solution)**

Let $u$ be a node in an OLDT structure $S$ labelled with $(\text{“}\alpha_1, \alpha_2, \ldots, \alpha_n, G\text{”}, \sigma)$, and let $v$ be a node in an OLDT structure $T$ for type infereuce labelled with $(\text{“}\beta_1, \beta_2, \ldots, \beta_n, H\text{”}, \mu)$ such that

(a) $(\text{“}\alpha_1, \alpha_2, \ldots, \alpha_n\text{”}, \sigma) \subseteq (\text{“}\beta_1, \beta_2 \ldots, \beta_n\text{”}, \mu)$, and

(b) the leftmost atoms of $G$ and $H$ are not call-exit markers.

Suppose that there exists an OLDT subrefutation $r$ of $(\text{“}\alpha_1, \alpha_2, \ldots, \alpha_n\text{”}, \sigma)$ starting from $u$ with its answer substitution $\tau$. Then there exists an extension of $T$ such that

(a) the extension contains an OLDT subrefutation $s$ of $(\text{“}\beta_1, \beta_2, \ldots, \beta_n\text{”}, \mu)$ for type inference starting from $v$ with its answer substitution $\nu$, and

(b) $(\text{“}\alpha_1, \alpha_2, \ldots, \alpha_n\text{”}, \tau) \subseteq (\text{“}\beta_1, \beta_2, \ldots, \beta_n\text{”}, \nu)$.

*Proof.* It is proved similarly to Lemma 4.3.2.

**Theorem 5 (Correctness of the Type inference)**

Let $\overline{\mathcal{C}}_{local}(G\mu)$, $\overline{\mathcal{C}}_{global}(G\mu)$, $\overline{\mathcal{E}}_{local}(G\mu)$, $\overline{\mathcal{E}}_{global}(G\mu)$ be the sets defined for the OLDT structure of $(G, \mu)$ for type inference (with the depth-first from-left-to-right strategy or any other strategy) as follows:

$\mathcal{C}_{local}(G\mu)$ : the set of all calling patterns at local success of $G\mu$,
$\overline{\mathcal{C}}_{global}(G\mu)$ : the set of all calling patterns at global success of $G\mu$,
$\overline{\mathcal{E}}_{local}(G\mu)$ : the set of all exiting patterns at local success of $G\mu$,
$\overline{\mathcal{E}}_{global}(G\mu)$ : the set of all exiting patterns at global success of $G\mu$.

Then

(a) $\overline{\mathcal{C}}_{local}(G\mu) \supseteq \mathcal{C}_{local}(G\mu)$
(b) $\overline{\mathcal{C}}_{global}(G\mu) \supseteq \mathcal{C}_{global}(G\mu)$
(c) $\overline{\mathcal{E}}_{local}(G\mu) \supseteq \mathcal{E}_{local}(G\mu)$
(d) $\overline{\mathcal{E}}_{global}(G\mu) \supseteq \mathcal{E}_{global}(G\mu)$

*Proof.* It is proved similarly to Theorem 4.

## 6. Mode Analysis by Abstract Hybrid Interpretation

Suppose that a top-level goal "*reverse*$(L,M)$" is executed with its first argument $L$ instantiated to a ground term, where *reverse* and *append* are defined by

```
reverse([ ],[ ]).
reverse([X|L],M) :- reverse(L,N), append(N,[X],M).
append([ ],M,M).
append([X|L],M,[X|N]) :- append(L,M,N).
```

Then, the first argument of *reverse* invoked from the top-level goal is always a ground term at calling time, and the second argument is always a ground term at exiting time. Similarly, so are the first and the second arguments of *append* at calling time and the third argument at exiting time. How can we show it mechanically?

In this section, we will reformulate the work by Mellish [10],[11] and Debray [3] from the point of view in Section 2.2. Unlike the type inference in Section 5, we need to keep an additional restriction about sharing of structures.

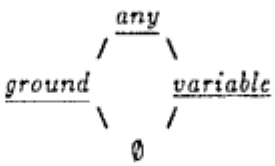
### 6.1 Mode Analysis

A *mode* is one of the following 4 sets of terms.

$\underline{any}$ : the set of all terms,
$\underline{ground}$ : the set of all ground terms,
$\underline{variable}$ : the set of all variables,
$\emptyset$ : the emptyset.

Modes are ordered by the instantiation ordering $\preceq$ depicted below.


Note that this is not the reverse of the set inclusion ordering $\subseteq$ depicted below.




A *mode substitution* is an expression of the form

$$< X_1 \Leftarrow \underline{m_1}, X_2 \Leftarrow \underline{m_2}, \ldots, X_l \Leftarrow \underline{m_l} >,$$

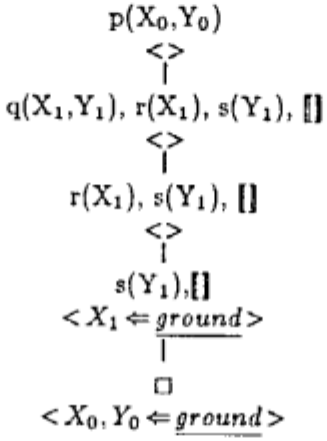
where $\underline{m_1}, \underline{m_2}, \ldots, \underline{m_l}$ are modes. The mode assigned to variable $X$ by mode substitution $\mu$ is denoted by $\mu(X)$. We stipulate that a mode substitution assigns $\underline{variable}$, the minimum element w.r.t. the instantiation ordering, to variable $X$ when $X$ is not in the domain of the mode substitution explicitly. Hence the empty mode substitution <> assigns $\underline{variable}$ to every variable.

So far, we have introduced the notions for mode analysis in the same way as depth-abstracted pattern enumeration and type inference. Now we have to manage a complication inherent to the mode analysis problem. If we had just mode substitutions, we could not propagate the mode information correctly. We need to take the sharings of structures into consideration in addition to mode substitutions.

*Example 6.1.1* The following is an example given by Debray [3]. Let the given program $P$ be

```
p(X,Y) :- q(X,Y), r(X), s(Y).
q(Z,Z).
r(a).
s(Z).
```

If we had not the additional information about sharing, the mode analysis of $p(X_0, Y_0)$ <> by abstract interpretation would proceed as follows:




```
p(X,Y) <> : [p(X,Y)<X ⇐ ground>]
q(X,Y) <> : [q(X,Y) <>]
r(X) <> : [r(X) <X ⇐ ground>]
s(Y) <> : [s(X) <>]
```

**Figure 6.1.1 Wrong Mode Analysis**

Though $Y_1$ must be a ground term when the goal $s(Y_1)$ is called, it is not correctly estimated above, because the sharing of structures between the variables $X_1$ and $Y_1$ caused by the unification of $q(X_1, Y_1)$ and $q(Z, Z)$ are not considered.

In order to prepare sharings for correct propagation of mode information, we need to consider graph representations of terms, atoms and negative clauses rather than tree representations of them. Terms, atoms and negative clauses are represented by directed acyclic graphs (DAG) possibly with sharing of structures, i.e., sharing of subgraphs.

*Example 6.1.2* When negative clauses "$q(X,Y), r(X), s(Y)$" and "$q(Z,Z), r(a), s(W)$ are unified, the result is "$q(a,a), r(a). s(a)$". When negative clauses "$q(X,Y), r(X), s(Y)$" and "$q(Z,W), r(a), s(a)$ are unified, the result is also "$q(a,a), r(a), s(a)$". In order to distinguish these unifications, we represent them in directed acyclic graphs as below.


**Figure 6.1.2 Graph Representation**

A *sharing* $\mathcal{M}$ on variables $X_1, X_2, \ldots, X_l$ is an equivalence relation on the set $\{X_1, X_2, \ldots, X_l\}$. We denote that $X$ and $Y$ are in $\mathcal{M}$ by $X \bowtie_{\mathcal{M}} Y$. A sharing $\mathcal{M}$ says that, if variable $X$ and variable $Y$ have the possibility to be bound to terms with shared structures, $X \bowtie_{\mathcal{M}} Y$ holds. By $\mathbf{1}$, we denote the identity relation, that is, $X \bowtie Y$ iff $X$ and $Y$ are identical, which means there is no possibility of sharing structures. $\mathcal{M}$ is said to be *smaller than* $\mathcal{N}$ *w.r.t. the instantiation ordering* when $X \bowtie_{\mathcal{N}} Y$ holds for any $X$ and $Y$ such that $X \bowtie_{\mathcal{M}} Y$, and denoted by $\mathcal{M} \preceq \mathcal{N}$. $\mathcal{M}$ is said to be *smaller than* $\mathcal{N}$ *w.r.t. the set inclusion ordering* when $X \bowtie_{\mathcal{M}} Y$ holds for any $X$ and $Y$ satisfying $X \bowtie_{\mathcal{N}} Y$, and denoted by $\mathcal{M} \subseteq \mathcal{N}$. (These two relations are the reverse of each other.)

Let $A$ be an atom in the body of some clause in $P \cup \{G\}$, $\mu$ be a mode substitution of the form

$< X_1 \Leftarrow \underline{m_1}, X_2 \Leftarrow \underline{m_2}, \ldots, X_l \Leftarrow \underline{m_l} >$.

and $\mathcal{M}$ be a sharing on variables $X_1, X_2, \ldots, X_l$. Then $A\mu\mathcal{M}$ is called a *mode-abstracted atom*, and denotes the set of all atoms obtained by replacing each $X_i$ in $A$ with a term in $\underline{m_i}$ without contradicting $\mathcal{M}$. Two mode-abstracted atoms $A\mu\mathcal{M}$ and $B\nu\mathcal{N}$ are said to be *unifiable* when $A\mu\mathcal{M} \cap B\nu\mathcal{N} \neq \emptyset$. A list of mode-abstracted atoms $[A_1\mu_1\mathcal{M}_1, A_2\mu_2\mathcal{M}_2, \ldots, A_n\mu_n\mathcal{M}_n]$ denotes the set union $\cup_{i=1}^{n} A_i\mu_i\mathcal{M}_i$. Similarly, $G\mu\mathcal{M}$ (or the pair $(G, \mu\mathcal{M})$) is called a *mode-abstracted negative clauses*, and denotes the set of negative clauses obtained by replacing each $X_i$ in $G$ with a term in $\underline{m_i}$ without contradicting $\mathcal{M}$. When $G$ is of the form "$A_1, A_2, \ldots, A_n$", the mode-abstracted atom $A_1\mu\mathcal{M}$ is called the leftmost mode-abstracted atom of $G\mu\mathcal{M}$.

The *mode analysis* w.r.t. $G\mu\mathcal{M}$ is the problem to compute

(a) some list of mode-abstracted atoms which is a superset of $\mathcal{C}_{local}(G\mu\mathcal{M})$,
(b) some list of mode-abstracted atoms which is a superset of $\mathcal{C}_{global}(G\mu\mathcal{M})$,
(c) some list of mode-abstracted atoms which is a superset of $\mathcal{E}_{local}(G\mu\mathcal{M})$, and
(d) some list of mode-abstracted atoms which is a superset of $\mathcal{E}_{global}(G\mu\mathcal{M})$.

## 6.2 Abstract Hybrid Interpretation for Mode Analysis

### 6.2.1 OLDT Structure for Mode Analysis

A *search tree*, a *solution table* and an *association for mode analysis* are defined in the same way as type inference, except that labels are of the form $(G, \mu\mathcal{M})$ and call-exit markers of the form $[A, \mu\mathcal{M}, \eta]$. (For brevity, we will sometimes omit the term "for mode analysis" hereafter in Section 6.) An *OLDT structure for mode analysis* is a triple of search tree, solution table and association. The relation between a node and its child nodes is specified by *OLDT resolution for mode analysis* in Section 6.2.3.

### 6.2.2 Overestimation of of Modes

Similarly to the type inference in Section 5, we need to consider the following situation in order to specify the operations for mode analysis at steps (A), (B) and (C) in Figure 2.2.1. Let $A$ be an atom, $X_1, X_2, \ldots, X_k$ all the variables in $A$, $\mu$ a mode substitution of the form

$< X_1 \Leftarrow \underline{m_1}, X_2 \Leftarrow \underline{m_2}, \ldots, X_k \Leftarrow \underline{m_k} >$,

$\mathcal{M}$ a sharing on $X_1, X_2, \ldots, X_k$. $B$ an atom, $Y_1, Y_2, \ldots, Y_l$ all the variables in $B$, and $\nu$ a mode substitution of the form

$< Y_1 \Leftarrow \underline{n_1}, Y_2 \Leftarrow \underline{n_2}, \ldots, Y_l \Leftarrow \underline{n_l} >$,

Then

(a) How can we know whether there is an atom $A\sigma = B\tau$ in $A\mu\mathcal{M} \cap B\nu\mathcal{N}$?

(b) If there is such an atom, what terms are expected to be assigned to each $Y_j$ by $\tau$?

(c) If there is such an atom, what sharing of structures is expected to occur between $Y_1, Y_2, \ldots, Y_l$ by $\tau$?

*Example 6.2.2.1* When a term in $p(X,Y) < X \Leftarrow \underline{ground}, Y \Leftarrow \underline{variable} >$ and a term in $p(f(U), g(V)) < U \Leftarrow \underline{variable}, V \Leftarrow \underline{variable} >$ are unified, $X, U$ must be a term in $\underline{ground}$, $Y$ must be a term in $\underline{any}$, and $V$ must be a term in $\underline{variable}$.

### (1) Overestimation of Unifiability

Along the same line of the discussion as the type inference in Section 5, what we need to compute is a mode containing all instances of some occurrence of $Z$ when an instance of term $t[Z]$ is in mode $\underline{m}$. We denote it by $Z/ < t[Z] \Leftarrow \underline{m} >$. Due to the choice of modes (see [3]), it is computed simply as follows:

$$Z/ < t[Z] \Leftarrow \underline{m} > = m.$$

*Example 6.2.2.2* Let $t$ be $[X|L]$ and $\underline{m}$ be $\underline{ground}$. Then

$X/ < [X|L] \Leftarrow \underline{ground} > = \underline{ground}$, $L/ < [X|L] \Leftarrow \underline{ground} > = \underline{ground}$.

Let $t$ be $[X|L]$ and $\underline{m}$ be $\underline{variable}$. Then

$X/ < [X|L] \Leftarrow \underline{variable} > = \underline{variable}$, $L/ < [X|L] \Leftarrow \underline{variable} > = \underline{variable}$.

Note that $Z/ < t[Z] \Leftarrow \underline{m} >$ is not $\emptyset$ when $\underline{m}$ is not $\emptyset$. Because the join $\vee$ of non-empty modes w.r.t. the instantiation ordering is always non-empty, the mode assigned to each variable is non-empty when $\mu$ and $\nu$ do not assign $\emptyset$ to any variable. This means that the unifiability of $A\mu\mathcal{M}$ and $B\nu\mathcal{N}$ can be reduced to the unifiability of $A$ and $B$, unlike the type inference in Section 5.

### (2) One Way Propagation of Mode Substitutions and Sharings

Recall the situation we are considering. Similarly to the type inference, we will first restrict our attentions to the case when $\nu = <>$ and $\mathcal{N} = \mathbf{1}$. Suppose that there is an atom $B\tau$ (represented by a directed acyclic graph) in $A\mu\mathcal{M} \cap B <> \mathbf{1}$. Then, what terms are expected to be assigned to variables in $B$ by $\tau$? And what sharing of structures is expected to occur among variables in $B$?

Let $\lambda$ be a mode substitution. Again, along the same line of the discussion, what we need to compute is a mode containing all instances of $s$ when each variable $X$ is assigned a

term in mode $\lambda(X)$. We denote it by $s/\lambda$, and compute it as follows:

$$s/\lambda = \begin{cases} \emptyset, & \lambda(X) = \emptyset \text{ for some } X \text{ in } s; \\ \lambda(X), & \text{when } s \text{ is a variable } X; \\ \underline{ground}, & \text{when } \lambda(X) = \underline{ground} \text{ for every variable } X \text{ in } s; \\ \underline{any}, & \text{otherwise.} \end{cases}$$

*Example 6.2.2.3* Let $s$ be $[X|L]$ and $\lambda$ be $<X \Leftarrow \underline{ground}, L \Leftarrow \underline{ground}>$. Then
$s/\lambda = \underline{ground}$.
Let $s$ be $[X|L]$ and $\lambda$ be $<X \Leftarrow \underline{any}, L \Leftarrow \underline{ground}>$. Then
$s/\lambda = \underline{any}$.

Simultaneously, we can overestimate the sharing $\mathcal{N}'$ on $Y_1, Y_2, \ldots, Y_l$ as follows: $Y_p \bowtie_{\mathcal{N}'} Y_q$ if and only if

(a) $\eta(Y_p)$ and $\eta(Y_q)$ contains a common variable, or
(b) there exist different variables $X_i$ and $X_j$ such that $X_i \bowtie_{\mathcal{M}} X_j$, the substituted terms $\eta(X_i)$ and $\eta(Y_p)$ contain some common variable, and the substituted terms $\eta(X_j)$ and $\eta(Y_q)$ contain some common variable.

*Example 6.2.2.4* When $\mu$ is a mode substitution
$<X \Leftarrow \underline{variable}>$,
$\mathcal{M}$ is the identity sharing
$\{X \bowtie X\}$,
and $\eta$ is a substitution
$<Y_1 \Leftarrow X, Y_2 \Leftarrow X>$,
then the sharing on $Y_1, Y_2$ is
$\{Y_1 \bowtie Y_1, Y_1 \bowtie Y_2, Y_2 \bowtie Y_1, Y_2 \bowtie Y_2\}$.

Let $A$, $B$ be atoms, $\mu$ a mode substitution for the variables in $A$, $\mathcal{M}$ a sharing on variables in $A$, and $\eta$ an m.g.u. of $A$ and $B$. The pair of the mode substitution for the variables in $B$ and the sharing on the variables in $B$, that is obtaind from $\mu$ and $\eta$ using $Z/<t[Z] \Leftarrow \underline{t}>$ and $s/\lambda$ above, is denoted by $propagate(\mu\mathcal{M}, \eta)$. (Note that $propagate(\mu\mathcal{M}, \eta)$ depends on just $\mu\mathcal{M}$ and $\eta$.)

### (3) Overestimation of Mode Substitutions and Sharings

As for the operation at step (A) for mode analysis, we can adopt the one way substitution-sharing propagation

$$propagate(\mu\mathcal{M}, \eta)$$

since the destination side substitution-sharing is $<>\mathbf{1}$. As for the operations at steps (B) and (C) for mode analysis where the destination side substitution-sharing is not necessarily equal to $<>\mathbf{1}$, we can adopt the join w.r.t. the instantiation ordering

$$\mu\mathcal{M} \vee propagate(\nu\mathcal{N}, \eta)$$

where $\vee$ is defined as follows:

$$\mu\mathcal{M} \vee \mu'\mathcal{M}' = \nu_0\mathcal{N}_0$$

where

$$\nu_0(X_i) = \begin{cases} \emptyset, & \text{for some variable } X_j,\ X_i \bowtie_{\mathcal{N}_0} X_j \text{ and } \mu(X_j) \vee \mu'(X_j) \text{ is } \emptyset; \\ \underline{any}, & \mu(X_i) \vee \mu'(X_i) \text{ is } \underline{variable}, \text{ and for some variable } X_j, \\ & X_i \bowtie_{\mathcal{N}_0} X_j \text{ and } \mu(X_j) \vee \mu'(X_j) \text{ is either } \underline{any} \text{ or } \underline{ground}; \\ \mu(X_i) \vee \mu'(X_i), & \text{otherwise.} \end{cases}$$

$$\mathcal{N}_0 = \mathcal{M} \vee \mathcal{M}'$$

*Example 6.2.2.5* Let $\mu$, $\mu'$ be mode substitutions and $\mathcal{M}$, $\mathcal{M}'$ sharings as follows:

$\mu : < X_1 \Leftarrow \underline{variable}, Y_1 \Leftarrow \underline{variable} >$,

$\mathcal{M} : \{X_1 \bowtie X_1, Y_1 \bowtie Y_1\}$,

$\mu' : < X_1 \Leftarrow \underline{ground}, Y_1 \Leftarrow \underline{variable} >$,

$\mathcal{M}' : \{X_1 \bowtie X_1, X_1 \bowtie Y_1, Y_1 \bowtie X_1, Y_1 \bowtie Y_1\}$.

and $\nu_0 \mathcal{N}_0$ be $\mu\mathcal{M} \vee \mu'\mathcal{M}'$. Then

$\nu_0 = < X_1 \Leftarrow \underline{ground}, Y_1 \Leftarrow \underline{any} >$,

$\mathcal{N}_0 = \{X_1 \bowtie X_1, X_1 \bowtie Y_1, Y_1 \bowtie X_1, Y_1 \bowtie Y_1\}$.

### 6.2.3 OLDT Resolution for Mode Analysis

The relation between a node and its child nodes of a search tree is specified by *OLDT resolution for mode anlysis* as follows:

A node of OLDT structure $(Tr, Tb, As)$ labelled with ("$\alpha_1, \alpha_2, \ldots, \alpha_n$", $\mu\mathcal{M}$) is said to be *OLDT resolvable* ($n \geq 1$) when $\alpha_1$ satisfies either of the following conditions.

(a) The node is a terminal solution node of $Tr$, and there is some definite clause "$B_0$ :- $B_1, B_2, \ldots, B_m$" ($m \geq 0$) in program $P$ such that $\alpha_1\mu$ and $B_0$ is unifiable, say by an m.g.u. $\eta$.

(b) The node is a lookup node of $Tr$, and there is some mode-abstracted atom $B\nu\mathcal{N}$ in the associated solution list of the lookup node such that $\alpha_1$ and $B$ are variants of each other. Let $\eta$ be the renaming of $B$ to $\alpha_1$.

The precise algorithm of OLDT resolutuion for mode analysis is shown in Figure 6.2.3.1. Note that the operations at steps (A), (B) and (C) are modified.

A node labelled with ("$\alpha_1, \alpha_2, \ldots, \alpha_n$", $\mu\mathcal{M}$) is a lookup node when the mode-abstracted atom $\alpha_1\mu\mathcal{M}$ is a key in the solution table, and is a solution node otherwise ($n \geq 1$).

The *initial OLDT structure*, *immediate extension of OLDT structure*, *extension of OLDT structure*, *answer substitution of OLDT refutation* and *solution of OLDT refutation* are defined in the same way as in Section 2.2.

OLDT-resolve(("$\alpha_1, \alpha_2, \ldots, \alpha_n$", $\mu\mathcal{M}$) : label) : label ;
  $i := 0$;
  **case**
    **when** a solution node is OLDT resolved with "$B_0$ :- $B_1, B_2, \ldots, B_m$" in $P$
      let $\eta$ be the m.g.u. of $\alpha_1$ and $B_0$ ;
      let $G_0$ be a negative clause "$B_1, B_2, \ldots, B_m, [\alpha_1, \mu, \mathcal{M}, \eta]$, $\alpha_2, \ldots, \alpha_n$" ;
      let $\nu_0\mathcal{N}_0$ be $propagate(\mu\mathcal{M}, \eta)$ ; — (A)
    **when** a lookup node is OLDT resolved with "$B\nu\mathcal{N}$" in $Tb$
      let $\eta$ be the renaming of $B$ to $\alpha_1$ ;
      let $G_0$ be a negative caluse "$\alpha_2, \ldots, \alpha_n$" ;
      let $\nu_0\mathcal{N}_0$ be $\mu\mathcal{M} \vee propagate(\nu\mathcal{N}, \eta)$ ; — (B)
  **endcase**
  **while** the leftmost of $G_i$ is a call-exit marker $[A_{i+1}, \mu_{i+1}, \mathcal{M}_{i+1}, \eta_{i+1}]$ **do**
      let $G_{i+1}$ be $G_i$ other than the leftmost call-exit marker ;
      let $\nu_{i+1}\mathcal{N}_{i+1}$ be $\mu_{i+1}\mathcal{M}_{i+1} \vee propagate(\nu_i\mathcal{N}_i, \eta_{i+1})$ ; — (C)
      add $A_{i+1}\nu_{i+1}\mathcal{N}_{i+1}$ to the last of $A_{i+1}\mu_{i+1}\mathcal{M}_{i+1}$'s solution list if it is not in it ;
      $i := i + 1$ ;
  **endwhile**
  $(G_{new}, \mu_{new}, \mathcal{M}_{new}) := (G_i, \nu_i, \mathcal{N}_i)$ ;
  **return** $(G_{new}, \mu_{new}, \mathcal{M}_{new})$.

**Figure 6.2.3.1 OLDT Resolution for Mode Analysis**

*Example 6.2.3* Let *reverse* and *append* be predicates as before. Then, extension of OLDT structure proceeds similarly to *reach* in Example 2.2.

First, the initial OLDT structure below is generated. The root node of the search tree is a solution node. The solution table contains only one entry with its key $reverse(L_0, M_0) < L_0 \Leftarrow \underline{ground} > \mathbf{1}$ and its solution list is [ ].

reverse($L_0$,$M_0$)
$< L_0 \Leftarrow \underline{ground} >$
**1**

reverse(L,M) $< L \Leftarrow \underline{ground} >$ **1** : [ ]

**Figure 6.2.3.2 Extension of OLDT Structure for Mode Analysis at Step 1**

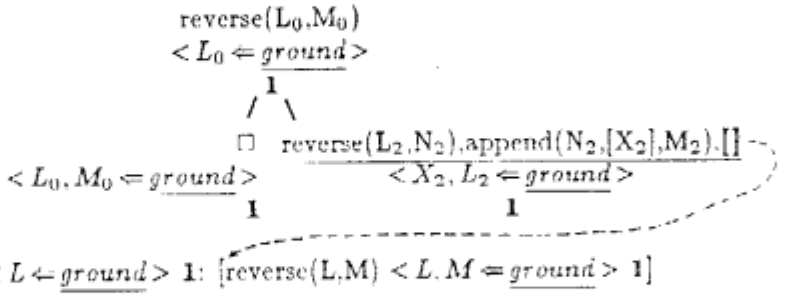


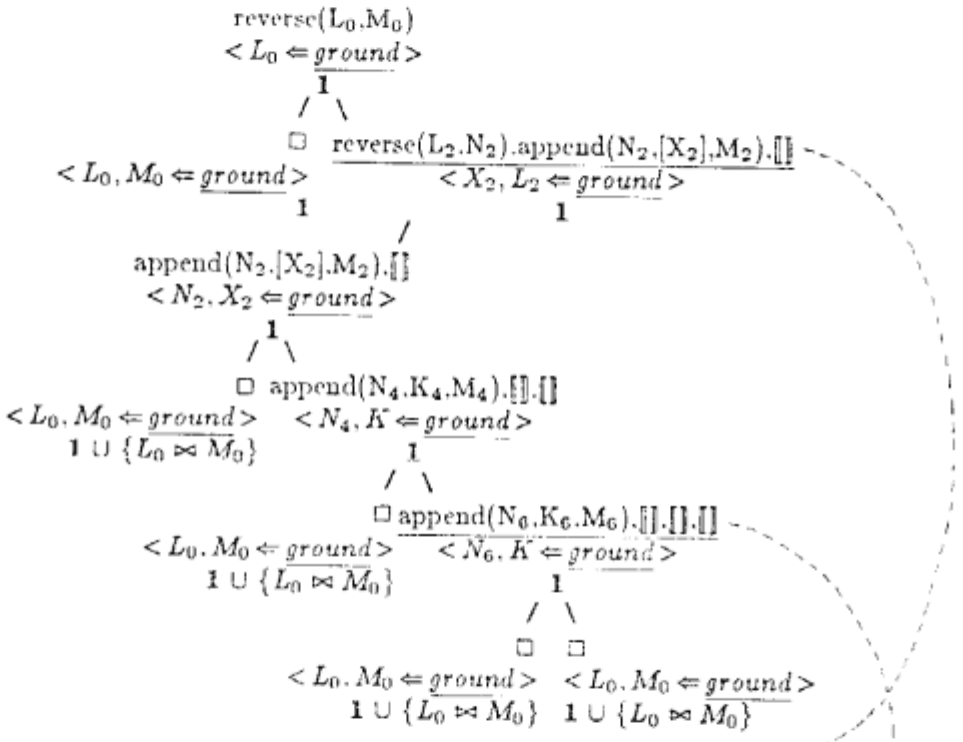

**Figure 6.2.3.3 Extension of OLDT Structure for Mode Analysis at Step 2**

Secondly, the root node is OLDT resolved using the program as before. The generated left node is the end of a unit subrefutation. Its solution $reverse(L_0, M_0) < L_0, M_0 \Leftarrow$

$\underline{ground} > \mathbf{1}$ is added to the solution list. The generated right node is a lookup node, because the mode abstraction of its leftmost atom is a variant of the label of the root node. The association associates the lookup node to the head of the solution list of $reverse(L_0, M_0) < L_0 \Leftarrow \underline{ground} > \mathbf{1}$.

Thirdly, the lookup node is OLDT resolved using the solution table to generate one child node. The association associates the lookup node to the last of the solution list.

Fourthly, the new solution node labelled with $append(N_2, [X_2], M_2) < N_2, X_2 \Leftarrow \underline{ground} > \mathbf{1}$ is OLDT resolved using the program. The generated left child node labelled with the null clause gives two solutions $append(N_2, [X_2], M_2) < N_2, X_2, M_2 \Leftarrow \underline{ground} > \mathbf{1} \cup \{X_2 \bowtie M_2\}$ and $reverse(L_0, M_0) < L_0, M_0 \Leftarrow \underline{ground} > \mathbf{1} \cup \{L_0 \bowtie M_0\}$ to the solution table. The generated right child node is a solution node.




reverse(L,M) $< L \Leftarrow \underline{ground} > \mathbf{1}$ :
[reverse(L,M) $< L, M \Leftarrow \underline{ground} > \mathbf{1}$,
reverse(L,M) $< L, M \Leftarrow \underline{ground} > \mathbf{1} \cup \{L \bowtie M\}$]
append(N,[X],M) $< N, X \Leftarrow \underline{ground} > \mathbf{1}$ :
[append(N,[X],M) $< N, X, M \Leftarrow \underline{ground} > \mathbf{1} \cup \{X \bowtie M\}$,
append(N,[X],M) $< N, X, M \Leftarrow \underline{ground} > \mathbf{1} \cup \{N \bowtie X, X \bowtie M, M \bowtie N\}$]
append(N,K,M) $< N, K \Leftarrow \underline{ground} > \mathbf{1}$ :
[append(N,K,M) $< N, K, M \Leftarrow \underline{ground} > \mathbf{1} \cup \{K \bowtie M\}$,
append(N,K,M) $< N, K, M \Leftarrow \underline{ground} > \mathbf{1} \cup \{N \bowtie K, K \bowtie M, M \bowtie N\}$]

**Figure 6.2.3.4 Extension of OLDT Structure for Mode Analysis at Step 7**

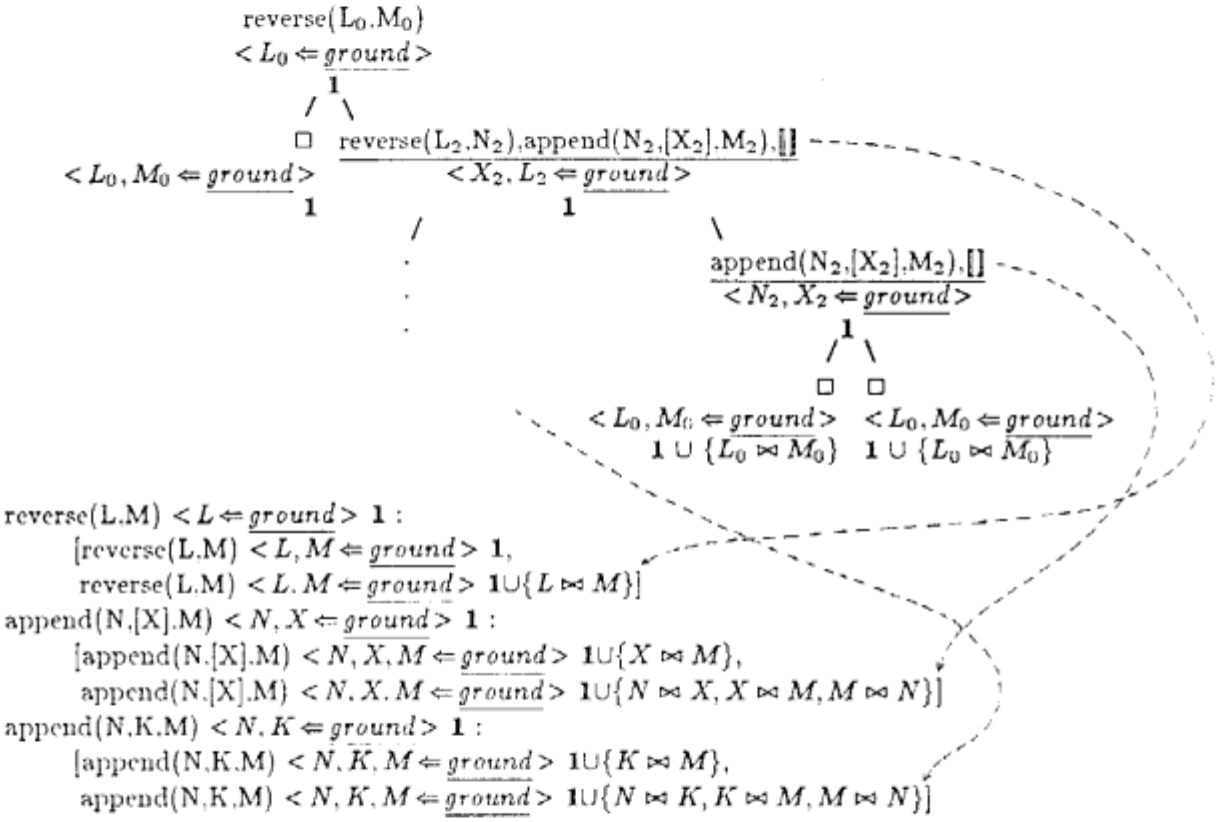
Fifthly, the new solution node labelled with $append(N_4, K_4, M_4) < N_4, X_4 \Leftarrow \underline{ground} > \mathbf{1}$ is OLDT resolved using the program. The generated left child node labelled with the null clause adds new solutions $append(N_4, K_4, M_4) < N_4, X_4, M_4 \Leftarrow \underline{ground} > \mathbf{1} \cup \{K_4 \bowtie M_4\}$

and $append(N_2,[X_2],M_2) < N_2, X_2, M_2 \Leftarrow \underline{ground} > \mathbf{1} \cup \{N_2 \bowtie X_2, X_2 \bowtie M_2, M_2 \bowtie N_2\}$ to the solution table. The generated right child node is a lookup node.

Sixthly, the lookup node labelled with $append(N_6, K_6, M_6) < N_6, X_6 \Leftarrow \underline{ground} > \mathbf{1}$ is OLDT resolved further using the solution table. The new node adds a solution $append(N_4, K_4, M_4) < N_4, X_4, M_4 \Leftarrow \underline{ground} > \mathbf{1} \cup \{N_4 \bowtie K_4, K_4 \bowtie M_4, M_4 \bowtie N_4\}$.

Seventhly, the lookup node is OLDT-resolved using a new solution of $append(N, K, M) < N, K \Leftarrow \underline{ground} > \mathbf{1}$.

Using a new solution of $reverse(L, M) < L \Leftarrow \underline{ground} > \mathbf{1}$, the right child of the root node is further OLDT resolved. At step 9, all nodes are OLDT-resolved up.




reverse(L,M) $< L \Leftarrow \underline{ground} > \mathbf{1}$ :
[reverse(L,M) $< L, M \Leftarrow \underline{ground} > \mathbf{1}$,
reverse(L,M) $< L, M \Leftarrow \underline{ground} > \mathbf{1} \cup \{L \bowtie M\}$]
append(N,[X],M) $< N, X \Leftarrow \underline{ground} > \mathbf{1}$ :
[append(N,[X],M) $< N, X, M \Leftarrow \underline{ground} > \mathbf{1} \cup \{X \bowtie M\}$,
append(N,[X],M) $< N, X, M \Leftarrow \underline{ground} > \mathbf{1} \cup \{N \bowtie X, X \bowtie M, M \bowtie N\}$]
append(N,K,M) $< N, K \Leftarrow \underline{ground} > \mathbf{1}$ :
[append(N,K,M) $< N, K, M \Leftarrow \underline{ground} > \mathbf{1} \cup \{K \bowtie M\}$,
append(N,K,M) $< N, K, M \Leftarrow \underline{ground} > \mathbf{1} \cup \{N \bowtie K, K \bowtie M, M \bowtie N\}$]

**Figure 6.2.3.5 Extension of OLDT Structure for Mode Analysis at Step 9**

### 6.3 Correctness of the Mode Analysis

We can prove the correctness of the mode analysis in the same way as the type inference in Section 5.3. The set inclusion ordering between labels of OLDT structures and those for mode analysis is defined in the same way as in Section 4.3.

**Lemma 6.3.1 (Overestimation of Resolvent)**

Let $(\text{“}\alpha_1, \alpha_2, \ldots, \alpha_n\text{”}, \sigma)$ be a label of OLDT structure and $(\text{“}\beta_1, \beta_2, \ldots, \beta_n\text{”}, \mu\mathcal{M})$ be a label of OLDT structure for mode analysis such that $(\text{“}\alpha_1, \alpha_2, \ldots, \alpha_n\text{”}, \sigma) \subseteq (\text{“}\beta_1, \beta_2, \ldots, \beta_n\text{”}, \mu\mathcal{M})$. Then

(1) When $u$ and $v$ are OLDT resolvable using a definite clause $C$, let $(G', \sigma')$ and $(H', \mu'\mathcal{M}')$ be the respective OLDT resolvents. Then $(G', \sigma') \subseteq (H', \mu'\mathcal{M}')$.

(2) When $u$ and $v$ are OLDT resolvable using solutions $B\tau$ and $B\nu$ such that $B\tau \subseteq B\nu$, let $(G', \sigma')$ and $(H', \mu'\mathcal{M}')$ be the respective OLDT resolvents. Then $(G', \sigma') \subseteq (H', \mu'\mathcal{M}')$.

*Proof.* It is proved similarly to Lemma 5.3.1.

**Lemma 6.3.2 (Overestimation of Solution)**

Let $u$ be a node in an OLDT structure $S$ labelled with $(\ulcorner \alpha_1, \alpha_2, \ldots, \alpha_n, G \urcorner, \sigma)$, and let $v$ be a node in an OLDT structure $T$ for mode analysis labelled with $(\ulcorner \beta_1, \beta_2, \ldots, \beta_n, H \urcorner, \mu\mathcal{M})$ such that

(a) $(\ulcorner \alpha_1, \alpha_2, \ldots, \alpha_n \urcorner, \sigma) \subseteq (\ulcorner \beta_1, \beta_2, \ldots, \beta_n \urcorner, \mu\mathcal{M})$, and

(b) the leftmost atoms of $G$ and $H$ are not call-exit markers.

Suppose that there exists an OLDT subrefutation $r$ of $(\ulcorner \alpha_1, \alpha_2, \ldots, \alpha_n \urcorner, \sigma)$ starting from $u$ with its answer substitution $\tau$. Then there exists an extension of $T$ such that

(a) the extension contains an OLDT subrefutation $s$ of $(\ulcorner \beta_1, \beta_2, \ldots, \beta_n \urcorner, \mu\mathcal{M})$ for mode analysis starting from $v$ with its answer substitution $\nu$ and its answer sharing $\mathcal{N}$, and

(b) $(\ulcorner \alpha_1, \alpha_2, \ldots, \alpha_n \urcorner, \tau) \subseteq (\ulcorner \beta_1, \beta_2, \ldots, \beta_n \urcorner, \nu\mathcal{N})$.

*Proof.* It is proved similarly to Lemma 4.3.2.

**Theorem 6 (Correctness of the Mode Analysis)**

Let $\overline{C}_{local}(G\mu\mathcal{M})$, $\overline{C}_{global}(G\mu\mathcal{M})$, $\overline{\mathcal{E}}_{local}(G\mu\mathcal{M})$, $\overline{\mathcal{E}}_{global}(G\mu\mathcal{M})$ be the sets defined for the OLDT structure of $(G, \mu, \mathcal{M})$ for mode analysis (with the depth-first from-left-to-right strategy or any other strategy) as follows:

$\overline{C}_{local}(G\mu\mathcal{M})$ : the set of all calling patterns at local success of $G\mu\mathcal{M}$,

$\overline{C}_{global}(G\mu\mathcal{M})$ : the set of all calling patterns at global success of $G\mu\mathcal{M}$,

$\overline{\mathcal{E}}_{local}(G\mu\mathcal{M})$ : the set of all exiting patterns at local success of $G\mu\mathcal{M}$,

$\overline{\mathcal{E}}_{global}(G\mu\mathcal{M})$ : the set of all exiting patterns at global success of $G\mu\mathcal{M}$.

Then

(a) $\overline{C}_{local}(G\mu\mathcal{M}) \supseteq C_{local}(G\mu\mathcal{M})$

(b) $\overline{C}_{global}(G\mu\mathcal{M}) \supseteq C_{global}(G\mu\mathcal{M})$

(c) $\overline{\mathcal{E}}_{local}(G\mu\mathcal{M}) \supseteq \mathcal{E}_{local}(G\mu\mathcal{M})$

(d) $\overline{\mathcal{E}}_{global}(G\mu\mathcal{M}) \supseteq \mathcal{E}_{global}(G\mu\mathcal{M})$

*Proof.* It is proved similarly to Theorem 5.

*Remark.* The reason the mode analysis needs the sharings while the type inference does not is the difference of the ordering structures. The instantiation ordering and the set inclusion ordering for mode analysis are not the reverse of each other, while those for type inference are. Underestimation of modes w.r.t. the instantiation ordering does not mean overestimation of them w.r.t. the set inclusion ordering, while it does for type inference.

## 7. A General Framework for Analysis of Success Patterns

In this section, we will generalize the approaches in Section 4, 5 and 6 as follows:

(a) Term sets, abstracted substitution and abstracted atoms are introduced so that general descriptions of term sets, substitutions and atom sets are considered.

(b) The overestimation operations at steps (A), (B) and (C) are generalized so that general approximations are considered.

(c) The information of sharings is generalized so that general constraints are considered.

### 7.1 Finite Approximation of Atom Sets

In general, a *term set* is one of the following $k+2$ subsets of the Herbrand universe (possibly with intersections).

$\underline{any}$ : the set of all terms,
$\underline{T_1}$ : a set of terms,
$\underline{T_2}$ : a set of terms,
$\vdots$
$T_k$ : a set of terms,
$\emptyset$ : the empty set.

We assume that there is an ordering $\preceq$ on term sets called the *instantiation ordering* such that the set of term sets is a finite lattice w.r.t. $\preceq$. This means that the join operation $\vee$ w.r.t. the instantiation ordering is always possible and there is a minimum term set w.r.t. the instantiation ordering. Because the term sets are sets of terms, we can naturally define the set inclusion ordering $\subseteq$.

An *abstracted substitution* $\mu$ is an expression of the form

$$< X_1 \Leftarrow \underline{t_1}, X_2 \Leftarrow \underline{t_2}, \ldots, X_l \Leftarrow \underline{t_l} >,$$

where $\underline{t_1}, \underline{t_2}, \ldots, \underline{t_l}$ are term sets. The term set assigned to variable $X$ by abstracted substitution $\mu$ is denoted by $\mu(X)$. We stipulate that an abstracted substitution assigns the minimum element w.r.t. the instantiation ordering to variable $X$ when $X$ is not in the domain of the abstracted substitution explicitly. Hence the empty substitution <> assigns the minimum element to every variable.

A *constraint* $\mathcal{M}$ on variables $X_1, X_2, \ldots, X_l$ is a restriction on the substituted terms to $X_1, X_2, \ldots, X_l$. We assume that there are two orderings, the instantiation ordering and the set inclusion ordering on constraints. (The more stonger a constraint is, the greater it is w.r.t. the instantiation ordering, and the smaller w.r.t. the set inclusion ordering.)

Let $A$ be an atom in the body of some clause in $P \cup \{G\}$, $\mu$ be an abstracted substitution of the form

$$< X_1 \Leftarrow \underline{t_1}, X_2 \Leftarrow \underline{t_2}, \ldots, X_n \Leftarrow \underline{t_n} >.$$

and $\mathcal{M}$ be a constraint on variables $X_1, X_2, \ldots, X_n$. Then $A\mu\mathcal{M}$ is called an *abstracted atom*, and denotes the set of all atoms obtained by replacing each $X_i$ in $A$ with a term in $\underline{t_i}$ without contradicting $\mathcal{M}$. Two abstracted atoms $A\mu\mathcal{M}$ and $B\nu\mathcal{N}$ are said to be *unifiable* when $A\mu\mathcal{M} \cap B\nu\mathcal{N} \neq \emptyset$. A list of abstracted atoms $[A_1\mu_1\mathcal{M}_1, A_2\mu_2\mathcal{M}_2, \ldots, A_n\mu_n\mathcal{M}_n]$ denotes the set union $\cup_{i=1}^{n} A_i\mu_i\mathcal{M}_i$. Similarly, $G\mu\mathcal{M}$ (or the pair $(G, \mu\mathcal{M})$) is called an *abstracted negative clause*, and denotes the set of negative clauses obtained by replacing each $X_i$ in $G$ with a term in $\underline{t_i}$ without contradicting $\mathcal{M}$. When $G$ is of the form "$A_1, A_2, \ldots, A_n$", the abstracted atom $A_1\mu\mathcal{M}$ is called the leftmost abstracted atom of $G\mu\mathcal{M}$.

The *analysis of success patterns* w.r.t. $G\mu\mathcal{M}$ is the problem to compute
(a) some list of abstracted atoms which is a superset of $\mathcal{C}_{local}(G\mu\mathcal{M})$,
(b) some list of abstracted atoms which is a superset of $\mathcal{C}_{global}(G\mu\mathcal{M})$,
(c) some list of abstracted atoms which is a superset of $\mathcal{E}_{local}(G\mu\mathcal{M})$, and
(d) some list of abstracted atoms which is a superset of $\mathcal{E}_{global}(G\mu\mathcal{M})$.

### 7.2 Abstarct Hybrid Interpretation for Analysis of Success Patterns

#### 7.2.1 OLDT Structure for Analysis of Success Patterns

A *search tree*, a *solution table* and an *association for analysis of success patterns* are defined in the same way as mode analysis. (For brevity, we will sometimes omit the term "for analysis of success patterns" hereafter in Section 7.) An *OLDT structure for analysis of success patterns* is a triple of a search tree, a solution table and an association. The relation between a node and its child nodes is specified by *OLDT resolution for analysis of success patterns* in Section 7.2.3.

### 7.2.2 Overestimation of Abstracted Patterns

Now we will generalize the operations at steps (A), (B) and (C) in Figure 2.2.1. We need to consider the situation abstracted from those in Section 4.2, 5.2.2 and 6.2.2.

#### (1) Overestimation of Unifiability

We need a procedure returning true if (and hopefully only if) two abstracted atoms $A\mu\mathcal{M}$ and $B\nu\mathcal{N}$ are unifiable. By defining $Z/ < t[Z] \Leftarrow \underline{t} >$ similarly, we can check the unifiability. If the procedure is time-consuming, we can use the unifiability check of $A$ and $B$ instead of the exact one.

#### (2) One Way Propagation of Abstracted Substitutions and Constraints

We need a one way substitution-constraint propagation operation $propagate(\mu\mathcal{M}, \eta)$. By defining $s/\lambda$ similarly, we can define $propagate(\mu\mathcal{M}, \eta)$ if the propagation of constraints is well-defined.

#### (3) Overestimation of Abstracted Substitutions and Constraints

We need to define the join operation $\vee$ for pairs of abstracted substitution and constraint. If atom sets and constraints are appropriately chosen, we can define $\vee$ in such a way that the operations at steps (A), (B) and (C) are defined as follows:

$$propagate(\mu\mathcal{M}, \eta),$$

$$\mu\mathcal{M} \vee propagate(\nu\mathcal{N}, \eta),$$

$$\mu_{i+1}\mathcal{M}_{i+1} \vee propagate(\nu_i\mathcal{N}_i, \eta_{i+1}).$$

### 7.2.3 OLDT Resolution for Analysis of Success Patterns

The relation between a node and its child nodes of a search tree is specified by *OLDT resolution for analysis of success patterns* as follows:

A node of OLDT structure $(Tr, Tb, As)$ labelled with ("$\alpha_1, \alpha_2, \ldots, \alpha_n$", $\mu\mathcal{M}$) is said to be *OLDT resolvable* ($n \geq 1$) when $\mu(X) \neq \emptyset$ for any variable $X$ and $\alpha_1$ satisfies either of the following conditions.

(a) The node is a terminal solution node of $Tr$, and there is some definite clause "$B_0$ :- $B_1, B_2, \ldots, B_m$" ($m \geq 0$) in program $P$ such that $\alpha_1$ and $B_0$ are unifiable. Let $\eta$ be an m.g.u. of $\alpha_1$ and $B_0$.

(b) The node is a lookup node of $Tr$, and there is some abstracted atom "$B\nu\mathcal{N}$" in the associated solution list of the lookup node such that $\alpha_1$ and $B$ are variants of each other. Let $\eta$ be the renaming of $B$ to $\alpha_1$.

The precise algorithm of OLDT resolution for analysis of success patterns is shown in Figure 7.2.3. Note that the operations at steps (A), (B) and (C) are modified.

OLDT-resolve(("$\alpha_1, \alpha_2, \ldots, \alpha_n$", $\mu\mathcal{M}$) : label) : label ;
  $i := 0$;
  **case**
    **when** a solution node is OLDT resolved with "$B_0$ :- $B_1, B_2, \ldots, B_m$" in $P$
      let $\eta$ be the m.g.u. of $\alpha_1$ and $B_0$ ;
      let $G_0$ be a negative clause "$B_1, B_2, \ldots, B_m, [\![\alpha_1, \mu, \mathcal{M}, \eta]\!], \alpha_2, \ldots, \alpha_n$" ;
      let $\nu_0\mathcal{N}_0$ be $propagate(\mu\mathcal{M}, \eta)$ ; — (A)
    **when** a lookup node is OLDT resolved with "$B\nu\mathcal{N}$" in $Tb$
      let $\eta$ be the renaming of $B$ to $\alpha_1$ ;
      let $G_0$ be a negative caluse "$\alpha_2, \ldots, \alpha_n$" ;
      let $\nu_0\mathcal{N}_0$ be $\mu\mathcal{M} \vee propagate(\nu\mathcal{N}, \eta)$ ; — (B)
  **endcase**
  **while** the leftmost of $G_i$ is a call-exit marker $[\![A_{i+1}, \mu_{i+1}, \mathcal{M}_{i+1}, \eta_{i+1}]\!]$ **do**
      let $G_{i+1}$ be $G_i$ other than the leftmost call-exit marker ;
      let $\nu_{i+1}\mathcal{N}_{i+1}$ be $\mu_{i+1}\mathcal{M}_{i+1} \vee propagate(\nu_i\mathcal{N}_i, \eta_{i+1})$ ; — (C)
      add $A_{i+1}\nu_{i+1}\mathcal{N}_{i+1}$ to the last of $A_{i+1}\mu_{i+1}\mathcal{M}_{i+1}$'s solution list if it is not in it ;
      $i := i + 1$ ;
  **endwhile**
  $(G_{new}, \mu_{new}\mathcal{M}_{new}) := (G_i, \nu_i\mathcal{N}_i)$ ;
  **return** $(G_{new}, \mu_{new}\mathcal{M}_{new})$.

**Figure 7.2.3 OLDT Resolution for Analysis of Success Patterns**

A node labelled with ("$\alpha_1, \alpha_2, \ldots, \alpha_n$", $\mu\mathcal{M}$) is a lookup node when the abstracted atom $\alpha_1\mu\mathcal{M}$ is a key in the solution table, and is a solution node otherwise ($n \geq 1$).

The *initial OLDT structure*, *immediate extension of OLDT structure*, *extension of OLDT structure*, *answer substitution of OLDT refutation* and *solution of OLDT refutation* are defined in the same way as in Section 2.2.

## 7.3 Correctness of the Analysis of Success Patterns

We can generalize the proofs of the correctness in Section 4.3, 5.3 and 6.3 for general analysis of success patterns by imposing the following conditions. The set inclusion ordering between labels of OLDT structures and those for analysis of success patterns is defined in the same way as in Section 4.3.

### Overestimation Condition

Let ("$\alpha_1, \alpha_2, \ldots, \alpha_n$", $\sigma$) be a label of OLDT structure and ("$\beta_1, \beta_2, \ldots, \beta_n$", $\mu\mathcal{M}$) be a label of OLDT structure for general analysis of success patterns such that ("$\alpha_1, \alpha_2, \ldots, \alpha_n$", $\sigma$) $\subseteq$ ("$\beta_1, \beta_2, \ldots, \beta_n$", $\mu\mathcal{M}$). Then

(A) Let "$B_0$ :- $B_1, B_2, \ldots, B_m$" be a definite clause. When $\alpha_1\sigma$ is unifiable with $B_0$ by an m.g.u. $\theta$ and $\alpha_1$ is unifiable with $B_0$ by an m.g.u. $\eta$, then
("$B_1, B_2, \ldots, B_m, [\![\alpha_1, \sigma]\!], \alpha_2, \ldots, \alpha_n$", $\theta$)
$\subseteq$ ("$B_1, B_2, \ldots, B_m, [\![\beta_1, \mu, \eta]\!], \beta_2, \ldots, \beta_n$", $propagate(\mu\mathcal{M}, \eta)$).

(B) Let $B\tau$ be an atom and $B\nu\mathcal{N}$ an abstracted atom such that $B\tau \subseteq B\nu\mathcal{N}$. When (a fresh variant of) $B\tau$ is an instance of $\alpha_1\sigma$ by an instantiation $\theta$, and $\alpha_1$ is a variant of $B$ by a renaming $\eta$, then

$(\lceil\alpha_2,\ldots,\alpha_n\rceil,\sigma\theta) \subseteq (\lceil\beta_2,\ldots,\beta_n\rceil,\mu \vee propagate(\nu\mathcal{N},\eta))$.

(C) When $\alpha_1$ is a call-exit marker $[\![A_1,\sigma_1]\!]$ and $\beta_1$ is a call-exit marker $[\![A_1,\mu_1,\mathcal{M}_1,\eta_1]\!]$, then

$(\lceil\alpha_2,\ldots,\alpha_n\rceil,\sigma_1\sigma) \subseteq (\lceil\beta_2,\ldots,\beta_n\rceil,\mu_1\mathcal{M}_1 \vee propagate(\mu\mathcal{M},\eta_1))$.

**Theorem 7. (Correctness of the Analysis of Success Patterns)**

Let $\overline{\mathcal{C}}_{local}(G\mu\mathcal{M})$, $\overline{\mathcal{C}}_{global}(G\mu\mathcal{M})$, $\overline{\mathcal{E}}_{local}(G\mu\mathcal{M})$, $\overline{\mathcal{E}}_{global}(G\mu\mathcal{M})$ be the sets defined for the OLDT structure of $(G,\mu,\mathcal{M})$ for analysis of success patterns (with the depth-first from-left-to-right strategy or any other strategy) as follows:

$\overline{\mathcal{C}}_{local}(G\mu\mathcal{M})$ : the set of all calling patterns at local success of $(G\mu\mathcal{M})$,
$\overline{\mathcal{C}}_{global}(G\mu\mathcal{M})$ : the set of all calling patterns at global success of $(G\mu\mathcal{M})$,
$\overline{\mathcal{E}}_{local}(G\mu\mathcal{M})$ : the set of all exiting patterns at local success of $(G\mu\mathcal{M})$,
$\overline{\mathcal{E}}_{global}(G\mu\mathcal{M})$ : the set of all exiting patterns at global success of $(G\mu\mathcal{M})$.

If the overestimation condition is satisfied, then

(a) $\overline{\mathcal{C}}_{local}(G\mu\mathcal{M}) \supseteq \mathcal{C}_{local}(G\mu\mathcal{M})$
(b) $\overline{\mathcal{C}}_{global}(G\mu\mathcal{M}) \supseteq \mathcal{C}_{global}(G\mu\mathcal{M})$
(c) $\overline{\mathcal{E}}_{local}(G\mu\mathcal{M}) \supseteq \mathcal{E}_{local}(G\mu\mathcal{M})$
(d) $\overline{\mathcal{E}}_{global}(G\mu\mathcal{M}) \supseteq \mathcal{E}_{global}(G\mu\mathcal{M})$

*Proof.* It is proved similarly to Theorem 5.

*Remark.* Though we have shown a general framework, we can simplify several steps of the OLDT resolution depending on the structure of the term sets. For example, We didn't need $\mathcal{M}$ in the type inference in Section 5, and we could check the unifiability of $\alpha_1\mu\mathcal{M}$ and $B\nu\mathcal{N}$ by checking the unifiability of $\alpha_1$ and $B$ in the mode analysis in Section 6.

## 8. Discussion

The hybrid interpretation in Section 2 is the basis of our abstract interpretation. The original OLDT resolution by Tamaki and Sato [13] is different from our hybrid interpretation in the following two respects.

(a) In order to avoid the generation of infinite number of solution nodes, the leftmost atom of the label of each newly generated node is depth-abstracted to $A$ by some depth $d$ after it is compared with existing keys of the solution tables to judge whether it is a solution node or a lookup node. When it is not classified into a lookup node and the depth of the leftmost atom is greater than $d$, a new initial OLDT tree of $A$ is added. (Hence the first element of their OLDT structure is a *forest*, not a tree.)

(b) A generated node is a lookup node when (the depth-abstraction of) its leftmost atom of the label is an *instance*, not necessarily *variant*, of some key in the solution table.

Our hybrid interpretation is still complete, but not necessarily complete with their multistage depth-first strategy ([13] pp.95–97). Nevertheless, the abstract interpretation based on our hybrid interpretation with depth-first from-left-to-right strategy is effective, because the number of abstracted atoms is finite due to each abstraction operation. The advantages of the standard hybrid interpretation are of more importance for abstract interpretation, because the top-down abstract interpretations are likely to dive into infinite loop due to the abstraction operations. For example, though the standard top-down interpretation of the

predicate *reverse* recurses with shorter first argument, the abstract top-down interpretation recurses, say from *list* to *list*, i.e., recurses with the same value in the abstracted domain.

The depth-abstracted pattern enumeration in Section 4 is a reformulation of the work by Sato and Tamaki [12] with the following modifications.

(a) Our approach applies the depth-abstraction to each substitution $\sigma$, while their original approach applies the depth-abstraction to each atom under the substitution. This modification exploits the fact that the success and exit patterns are instances of some atom in the body of some clause in $P \cup \{G\}$.

(b) Our depth $d$ abstraction does not replace variables and constants at depth $d$, while their original depth $d$ abstraction replaces every depth $d$ subterms even if it is a variable or a constant. This trivial modification makes the depth-abstracted pattern enumeration more precise.

The type inference in Section 5 is an improvement of our previous work [4]. Our previous approach was basically bottom-up so that we needed some device to exclude irrelevant goals. The device, called *generalization* or *closure*, employed in our previous approach is no longer necessary in our new approach.

The mode analysis in Section 6 is a modification and an improvement of the works by Mellish [10],[11] and Debray [3]. Melish's approach, following his general framework, first derives simultaneous recurrence equations for modes, and solves them in a bottom-up manner. Debray's approach is very close to ours except that he did not use *sharing* as an additional information. Though he pointed out the importance of the problem of "aliasing" (see Example 6.1.1), and gave a sufficient condition for safe use of his mode inference, the result of his mode inference is not correct *in general* when four modes *any*, *ground*, *variable* and $\emptyset$ are considered. The result of our mode anlysis is always correct due to the combination with "sharing analysis" in compensation for additional computational cost. (For the case when just three modes *any*, *ground* and $\emptyset$ are considered, see [4],[7],[8].)

The framework in Section 7 is motivated by the work by Mellish [11] to formalize a common framework for logic program analysis. As mentioned in Section 1, Mellish's approach derives simultaneous recurrence equations for the sets of goals at calling time and exiting time during the top-down execution of a given top-level goal, and obtains a superset of the least solution of the simultaneous recurrence equations using a bottom-up approximation. Our approach starts from a standard hybrid interpretation, and obtains the superset by abstracting the standard hybrid interpretation directly. We believe that, due to the more direct correspondence between the standard interpretation and the abstract interpretation, the name "abstract *interpretation*" fits with our abstract interpretation more naturally.

## 9. Conclusions

We have shown a common theoretical framework for logic program analysis and its effective examples for depth-abstracted pattern enumeration, type inference and mode analysis. This method is an element of our system for analysis of Prolog programs Argus/A under development [5],[7],[8],[9].

## Acknowledgements

As usual, we owe very much to Mr. Hisao Tamaki (Ibaraki University) and Dr. Taisuke Sato (Electrotechnical Laboratory). We would like to express deep gratitude to them for their perspicuous and stimulative works [12], [13].

Our analysis system Argus/A under development is a subproject of the Fifth Generation Computer System(FGCS) "Intelligent Programming System". The authors would like to thank Dr. K. Fuchi (Director of ICOT) for the opportunity of doing this research, and Dr. K. Furukawa (Vice Director of ICOT), Dr. T. Yokoi (Vice Director of ICOT) and Dr. H. Ito (Chief of ICOT 3rd Laboratory) for their advice and encouragement.

## References

[1] Cousot.P.and R.Cousot, "Abstract Interpretation : A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints," Conference Record of the 4th ACM Symposium on Principles of Programming Languages, Los Angeles, pp.238–252, 1977.

[2] Cousot.P.and R.Cousot, "Static Determination of Dynamic Properties of Recursive Procedures," in Formal Description of Programming Concepts (E.J.Neuhold Ed), pp. 237–277, North Holland, 1978.

[3] Debray.S.K.and D.S.Warren, "Detection and Optimization of Functional Computation in Prolog," Proc. of 3rd International Conference on Logic Programming, 1986.

[4] Debray.S.K., "Automatic Mode Inference for Prolog Programs," Proc. of 1986 Symposium on Logic Programming, Salt Lake City, 1986.

[5] Horiuchi.K.and T.Kanamori, "Polymorphic Type Inference in Prolog by Abstract Interpretation," Proc. of Logic Programming Conference, pp.107–116, Tokyo, 1987.

[6] Kanamori.T. and K.Horiuchi, "Type Inference in Prolog and its Application," Proc. of 9th International Joint Conference on Artificial Intelligence, pp.704–707, 1985.

[7] Kanamori.T., K.Horiuchi and T. Kawamura, "Detecting Functionality of Logic Programs Based on Abstract Hybrid Interpretation," to appear, ICOT Technical Report, 1987.

[8] Kanamori.T., K.Horiuchi and T. Kawamura, "Detecting Termination of Logic Programs Based on Abstract Hybrid Interpretation," to appear, ICOT Technical Report, 1987.

[9] Maeji.M.and T.Kanamori, "Top-down Zooming Diagnosis of Logic Programs," to appear, ICOT Technical Report, 1987.

[10] Mellish.C.S., "Some Global Optimizations for A Prolog Compiler," J. Logic Programming, pp.43–66, 1985.

[11] Mellish.C.S., "Abstract Interpretation of Prolog Programs," Proc. of 3rd International Conference on Logic Programming, pp.463–474, 1986.

[12] Sato.T.and H.Tamaki, "Enumeration of Success Patterns in Logic Programming," Proc. of International Colloquium of Automata, Language and Programming, pp.640–652, 1984.

[13] Tamaki.H.and T.Sato, "OLD Resolution with Tabulation," Proc. of 3rd International Conference on Logic Programming, pp.84–98, London, 1986.