ICOT Technical Report: TR-549

TR-549

Abstract Interpretation Based on Alexander Templates

by
T. Kanamori (Mitsubishi)

April, 1990

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

Abstract Interpretation based on Alexander Templates

Tadashi KANAMORI

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

Abstract

Recently, several bottom-up query evaluation methods for logic databases, e.g., magic set, Alexander method, Magic Templates, etc., have been developed. Given a logic program and a top-level query, "Alexander Templates (AT)" by Seki, which is the most refined of such query evaluation methods, once transforms the program and query, and then evaluates the transformed program and query in the bottom-up manner. This query evaluation method is proved to be as powerful as the top-down evaluation methods with memo-ization, e.g., OLDT resolution, SLD-AL resolution, Extension Tables, etc. On the other hand, several unified frameworks for abstract interpretation based on those top-down methods with memo-ization have been developed as well. Given a logic program and a top-level query, this approach analyzes various run-time properties by approximately executing the query in some abstract domain using the top-down evaluation with memo-ization. Utilizing the correspondence between AT and OLDT resolution, this paper presents a framework for abstract interpretation based on AT, and, in particular, shows the relation to Mellish's abstract interpretation.

Keywords : Program Analysis, Abstract Interpretation, Prolog.

Contents

1. Introduction

Recently, several bottom-up query evaluation methods for logic databases, e.g., magic set [1], Alexander method [16], Magic Templates [15], etc., have been developed. Given a logic program and a top-level query, "Alexander Templates (AT)" by Seki [17],[18], which is the most refined of such query evaluation methods, once transforms the program and query, and then evaluates the transformed program and query in the bottom-up manner. This query evaluation method is proved to be as powerful as the top-down evaluation methods with memo-ization, e.g., OLDT resolution [19], SLD-AL resolution [20], Extension Tables [5], etc. On the other hand, several unified frameworks for abstract interpretation based on those top-down methods with memo-ization have been developed as well. Given a logic program and a top-level query, this approach analyzes various run-time properties by approximately executing the query in some abstract domain using the top-down execution with memo-ization. This paper presents a framework for abstract interpretation based on "Alexander Templates" utilizing the correspondence between AT and OLDT resolution, and, in particular, shows the relation to Mellish's abstract interpretation.

The rest of this paper is organized as follows: Section 2 introduces "Alexander Templates (AT)" by Seki, and Section 3 shows the framework for abstract interpretation based on it using a mode inference problem as one of its examples. (Extracting a general framework and instantiating it to other abstract domains is immediate.) Section 4 points out the relation to Mellish's abstract interpretation, and discusses the correspondence between the two frameworks for abstract interpretation, one based on AT and the other based on OLDT resolution.

2. Alexander Templates (AT)

This section introduces a slightly modified version of "Alexander Templates" by Seki [17] starting with its naive version in Section 2.1 and shifting to its refined version in Section 2.2. In the following, a *program* is a finite set of definite clauses, and a *query* is an expression of the form "?- B," where B is an atom. We do not make a distinction between a set of atoms and a set consisting of their variant atoms.

2.1 Naive Alexander Templates

(1) Outline of Naive Alexander Templates

The naive version "$AT0$" receives a program P and a query Q and returns a set of atoms. In "$AT0$," a subprocedure "$transform0$" is first applied to P and Q to obtain a pair of a program $\tilde{P}$ and an atom $\tilde{Q}$, and then a subprocedure "$evaluate0$" is applied to $\tilde{P}$ and $\tilde{Q}$ to obtain the result. The subprocedures "$transform0$" and "$evaluate0$" are explained in the following subsections.

Example 2.1.1 Let P be a program as below:

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

The first clause of "*reach*" says that node Y is reachable from node X if node Z is reachable from X and there is an edge from Z to Y, while the second clause says that any node is reachable from itself. The unit clauses of "*edge*" give the edges of the directed graph of Figure 2.1. (This program is a typical *left recursive program*.)

Let Q be a query "?- $reach(a, Z_0)$." Then, the execution of "$reach(a, Z_0)$" immediately calls "$reach(a, Z_1)$" recursively at the leftmost in the body of the first clause to repeat the execution of the goal of the same form.

This program and query were used by Tamaki and Sato to explain their OLDT resolution in [19]. We use them throughout Section 2 to explain "Alexander Templates."

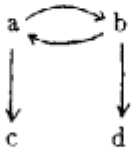

Figure 2.1 Graph Reachability Problem

In the following, we assume that each clause in P is assigned a unique natural number, called the *clause number*. For example, the six clauses in program P above are assigned clause numbers $1, 2, 3, 4, 5, 6$, respectively.

(2) Naive Transformation of Program and Query

A given pair (P, Q) is first transformed to another pair $(\tilde{P}, \tilde{Q})$, where the predicate symbols appearing in (P, Q) and those appearing in $(\tilde{P}, \tilde{Q})$ are disjoint. For each predicate "p" in (P, Q), we prepare the following two types of predicates:

- predicate "$call\_p$" with the same arity as "p,"
- predicate "sol_p" with the same arity as "p."

The predicates with prefix "*call_*" are called *call-predicates*, while those with prefix "*sol_*" are called *sol-predicates*. Similarly, the atoms with call-predicates are called *call-atoms*, while those with sol-predicates are called *sol-atoms*. When an atom A is of the form $p(t_1, t_2, \ldots, t_n)$, we denote $call\_p(t_1, t_2, \ldots, t_n)$ by $call\_A$, and $sol\_p(t_1, t_2, \ldots, t_n)$ by sol_A.

Let C be a clause in P, say of the form

A_0 :- $A_1, A_2, \ldots, A_m$.

The top-down execution using this clause proceeds as follows.

- If an atom unifiable with the head atom is called, then the first body atom under the m.g.u. is called.
- If an atom unifiable with the head atom is called, and the first body atom under the m.g.u. is solved with some answer substitution, then the second body atom under the composed substitution is called.

  ⋮
- If an atom unifiable with the head atom is called, and all the body atoms are solved with some answer substitution, then the first atom is solved with the composed substitution.

If we simulate the behavior of the top-down execution by the bottom-up reading of another definite clauses, the new definite clauses corresponding to the behavior above are as below:

$call\_A_1$:- $call\_A_0$.

$call\_A_2$:- $call\_A_0, sol\_A_1$.

⋮

$call\_A_m$:- $call\_A_0, sol\_A_1, \ldots, sol\_A_{m-1}$.
sol_A_0 :- $call\_A_0, sol\_A_1, \ldots, sol\_A_{m-1}, sol\_A_m$.

Taking this correspondence into account, the subprocedure "$transform0$" is as below:

Algorithm "transform0"

Input : a program P and a query Q.
Output : a pair of a program and an atom $(\tilde{P}, \tilde{Q})$.
Procedure : Let Q be of the form "?- B."
step 0: Initialize $\tilde{P}$ to $\emptyset$.
step 1: For each clause in P, say of the form
"A_0 :- $A_1, A_2, \ldots, A_m$" $(m \geq 0)$,
add the following $m + 1$ clauses to $\tilde{P}$.
$call\_A_1$:- $call\_A_0$.
$call\_A_2$:- $call\_A_0, sol\_A_1$.
$\vdots$
$call\_A_m$:- $call\_A_0, sol\_A_1, sol\_A_2, \ldots, sol\_A_{m-1}$.
sol_A_0 :- $call\_A_0, sol\_A_1, sol\_A_2, \ldots, sol\_A_{m-1}, sol\_A_m$.
step 2 : Let $\tilde{Q}$ be "$call\_B$."
step 3: Return $(\tilde{P}, \tilde{Q})$.

Example 2.1.2 Let P and Q be as before. Then, "$transform0$" applied to (P, Q) returns a pair of the program and atom below:

$\tilde{P}$:
```
call_reach(X,Z) :- call_reach(X,Y).
call_edge(Z,Y) :- call_reach(X,Y), sol_reach(X,Z).
sol_reach(X,Y) :- call_reach(X,Y), sol_reach(X,Z), sol_edge(Z,Y).
sol_reach(X,X) :- call_reach(X,X).
sol_edge(a,b) :- call_edge(a,b).
sol_edge(a,c) :- call_edge(a,c).
sol_edge(b,a) :- call_edge(b,a).
sol_edge(b,d) :- call_edge(b,d).
```
$\tilde{Q}$:
```
call_reach(a,Z).
```

(3) Naive Evaluation of the Transformed Program and Query

Let $\tilde{C}$ be a clause in $\tilde{P}$, and Γ be a set of atoms. Then, atom $B\theta$ is said to be *generated from* Γ *using* $\tilde{C}$ when

- B is the head atom of $\tilde{C}$, and
- there exists a sequence of atoms in Γ unifiable with the sequence of the body atoms of $\tilde{C}$, and θ is the restriction of an m.g.u. to the variables appearing in the head atom.

Then, the subprocedure "$evaluate0$" is as below:

Algorithm "$evaluate0$"

Input : a program $\tilde{P}$ and an atom $\tilde{Q}$.
Output : a set of atoms.
Procedure : Let $\tilde{Q}$ be of the form "$call\_B$."

step 0: Initialize Γ to Γ_0, where Γ_0 is $\{call\_B\}$.
step 1: Update Γ to $\Gamma' \cup \Gamma_0$, where Γ' is the set of all the atoms generated from Γ using some clause in $\tilde{P}$. Repeat this step until Γ does not increase.
step 2: Return the set of all the atoms B' such that sol_B' is in Γ and B' is an instance of B.

Example 2.1.3 Let $\tilde{P}$ and $\tilde{Q}$ be as before. Then, "*evaluate*0" generates the atoms below at step 1:

0th Repetetion: $call\_reach(a, Z)$.
1st Repetetion: $sol\_reach(a, a)$.
2nd Repetetion: $call\_edge(a, Y)$.
3rd Repetetion: $sol\_edge(a, b), sol\_edge(a, c)$.
4th Repetetion: $sol\_reach(a, b), sol\_reach(a, c)$.
5th Repetetion: $call\_edge(b, Y), call\_edge(c, Y)$.
6th Repetetion: $sol\_edge(b, a), sol\_edge(b, d)$.
7th Repetetion: $sol\_reach(a, d)$.

Then, atoms $reach(a, a), reach(a, b), reach(a, c), reach(a, d)$ are returned at step 2.

2.2 Refined Alexander Templates

(1) Outline of Refined Alexander Templates

The refined version "$AT1$" is the same as the naive version "$AT0$" except that "*transform*1" and "*evaluate*1" are used instead of "*transform*0" and "*evaluate*0."

(2) Refined Transformation of Program and Query

Let C be a clause in P, say with clause number n of the form

$$A_0 \text{ :- } A_1, A_2, \ldots, A_m$$

and $\tilde{C}_0, \tilde{C}_1, \ldots, \tilde{C}_m$ be the corresponding clauses in $\tilde{P}$ of the form

$$call\_A_1 \text{ :- } call\_A_0.$$
$$call\_A_2 \text{ :- } call\_A_0, sol\_A_1.$$
$$\vdots$$
$$call\_A_m \text{ :- } call\_A_0, sol\_A_1, \ldots, sol\_A_{m-1}.$$
$$sol\_A_0 \text{ :- } call\_A_0, sol\_A_1, \ldots, sol\_A_{m-1}, sol\_A_m.$$

Then, for two atom sequences

$$call\_A_0, sol\_A_1, \ldots, sol\_A_{i-1},$$
$$call\_A_0, sol\_A_1, \ldots, sol\_A_{i-1}, \ldots, sol\_A_{j-1}$$

in the body of the clauses in $\tilde{P}$, the same combination for the first sequnce must be checked for the second sequence over again in the "evaluation" phase ($i < j$). To save the information about the same combination, we need to specify

- the sequence "$call\_A_0, sol\_A_1, \ldots, sol\_A_{i-1}$," and
- the binding of the variables when the sequence "$call\_A_0, sol\_A_1, \ldots, sol\_A_{i-1}$" is unified with a sequence of atoms in Γ.

To specify the sequence, we need the location of atom A_i in the clause C. To specify the binding of the variables, we need the list of all the variables in "$call\_A_0, sol\_A_1, \ldots, sol\_A_{i-1}$," or, more precisely, the list of the variables necessary for another such sequences.

To store the information, we employ a binary predicate "*cont*" as follows:

- Its first argument is the list with two elements $[n, i]$ to denote the location of the occurrence of atom A_i in the clause with clause number n. (For example, the location of $reach(X, Z)$ in the first clause in the program before is denoted by $[1, 1]$.")
- Its second argument is the list of variables $l_{n,i}$ to denote all the variables that occur among both "$A_0, A_1, \ldots, A_{i-1}$" and "$A_i, A_{i+1}, \ldots, A_m, A_0$" simultaneously in the clause with clause number n. (The variables in the list are assumed to be ordered according to the order of textual appearance in the clause.)

Atoms with predicate "*cont*" are called *cont-atoms*. (The predicate "*cont*" is used in [17] to stand for "continuation.")

The subprocedure "$transform1$" is as below, where a clause with two head atoms

A_0, A'_0 :- $A_1, A_2, \ldots, A_k$

is just a convention of writing two clauses

A_0 :- $A_1, A_2, \ldots, A_k$,
A'_0 :- $A_1, A_2, \ldots, A_k$.

Algorithm "transform1"

Input : a program P and a query Q.
Output : a pair of a program and an atom $(\tilde{P}, \tilde{Q})$.
Procedure : Let Q be of the form "?- B."
step 0: Initialize $\tilde{P}$ to $\emptyset$.
step 1: For each clause in P, say with clause number n of the form
"A_0 :- $A_1, A_2, \ldots, A_m$" $(m \geq 0)$,
add the following $m + 1$ clauses to $\tilde{P}$.

$call\_A_1, cont([n, 1], l_{n,1})$:- $call\_A_0$,
$call\_A_2, cont([n, 2], l_{n,2})$:- $cont([n, 1], l_{n,1}), sol\_A_1$,
$\vdots$
$call\_A_m, cont([n, m], l_{n,m})$:- $cont([n, m-1], l_{n,m-1}), sol\_A_{m-1}$,
sol_A_0 :- $cont([n, m], l_{n,m}), sol\_A_m$

step 2: Let $\tilde{Q}$ be "$call\_B$."
step 3: Return $(\tilde{P}, \tilde{Q})$.

Note that each clause in $\tilde{P}$ obtained by "$transform1$" possibly has two head atoms, but at most two body atoms.

Example 2.2.1 Let P and Q be as before. Then, "$transform1$" applied to (P, Q) returns a pair of the program and atom below:

$\tilde{P}$:
```
call_reach(X,Z),cont([1,1],[X,Y]) :- call_reach(X,Y).
call_edge(Z,Y),cont([1,2],[X,Y,Z]) :-
            cont([1,1],[X,Y]),sol_reach(X,Z).
sol_reach(X,Y) :- cont([1,2],[X,Y,Z]),sol_edge(Z,Y).
sol_reach(X,X) :- call_reach(X,X).
sol_edge(a,b) :- call_edge(a,b).
sol_edge(a,c) :- call_edge(a,c).
sol_edge(b,a) :- call_edge(b,a).
sol_edge(b,d) :- call_edge(b,d).
```
$\tilde{Q}$:
```
call_reach(a,Z).
```

(3) Refined Evaluation of the Transformed Program and Query

According to the refinement of the transformation phase, we need to generalize the evaluation phase to use clauses with two head atoms. In addition, we adopt the "semi-naive" bottom-up evaluation. (In the following definition, Δ_{new} is used to keep the set of atoms generated just one step before.)

Let $\tilde{C}$ be a clause in $\tilde{P}$, and Δ and Δ_{new} be sets of atoms. Then, atom $B\theta$ is said to be *generated from* (Δ, Δ_{new}) *using* $\tilde{C}$ when

- B is *in* the head of $\tilde{C}$, and
- there exists a sequence of atoms in Δ unifiable with the sequence of the body atoms of $\tilde{C}$, at least one of the atoms in the sequence is in Δ_{new}, and the substitution θ is the restriction of an m.g.u. to the variables appearing in the head atom.

The subprocedure "*evaluate*1" is as below:

Algorithm "*evaluate*1"

Input : a program $\tilde{P}$ and an atom $\tilde{Q}$.
Output : a set of atoms.
Procedure : Let $\tilde{Q}$ be of the form "*call*_B."
step 0: Initialize Δ and Δ_{new} to $\{\tilde{Q}\}$.
step 1: Update Δ to $\Delta' \cup \Delta$, where Δ' is the set of all the atoms generated from (Δ, Δ_{new}) using some clause in $\tilde{P}$. Update Δ_{new} to the difference between the new Δ and the previous Δ. Repeat this step until Δ does not increase.
step 2: Return the set of all the atoms B' such that *sol*_B' is in Δ and B' is an instance of B.

Example 2.2.2 Let $\tilde{P}$ and $\tilde{Q}$ be as before. Then, "*evaluate*1" generates the atoms below at step 1:

0th Repetetion: $call\_reach(a, Z)$.
1st Repetetion: $cont([1,1],[a,Y]), sol\_reach(a,a)$.
2nd Repetetion: $call\_edge(a,Y), cont([1,2],[a,Y,a])$.
3rd Repetetion: $sol\_edge(a,b), sol\_edge(a,c)$.
4th Repetetion: $sol\_reach(a,b), sol\_reach(a,c)$.
5th Repetetion: $call\_edge(b,Y), cont([1,2],[a,Y,b]), call\_edge(c,Y), cont([1,2],[a,Y,c])$.
6th Repetetion: $sol\_edge(b,a), sol\_edge(b,d)$.
7th Repetetion: $sol\_reach(a,d)$.

Then, atoms $reach(a,a), reach(a,b), reach(a,c), reach(a,d)$ are returned at step 2.

2.3 Correctness of Alexander Templates

"Alexander Templates" simulates how the top-down interpreter calls atoms and solves them by generating call-atoms and sol-atoms. It just avoids repeating the same computation in the top-down interpreter by utilizing atoms already generated so that, for any top-level query, AT generates atoms *call*_A and *sol*_B if and only if the top-down interpreter calls A and solves with answer B. In particular, the "if" part is the basis of our abstract interpretation.

Theorem (Correctness of Alexander Templates)
Let P be a program, Q be a query, and $(\tilde{P}, \tilde{Q})$ be the result of "$transform1(P, Q)$."

(a) An atom $A\sigma$ appears at the leftmost of a goal during OLD resolution of Q using P, if and only if "*evaluate*1($\tilde{P}, \tilde{Q}$)" generates *call_*$A\sigma$. (Correctness for Calling Patterns)
(b) An atom is solved with solution $A\tau$ during OLD resolution of Q using P, if and only if "*evaluate*1($\tilde{P}, \tilde{Q}$)" generates *sol_*$A\tau$. (Correctness for Exiting Patterns)

Proof. The proof of the "if" parts is by induction on the number of steps required to generate the atoms. Due to space limit, we will omit it. The proof of the "only if" parts is similar to that of the correctness of abstract interpretation based on AT. See Appendix.

Though all solutions were found in the example of Section 2.1 and 2.2, this is not always the case, that is, the generation at step 1 in "*evaluate*1" might continue forever. The reason is that there might be generated infinitely many different atoms. (However, when this AT is applied to an abstract domain with finite elements, it always terminates. See Section 3.3.)

3. Abstract Interpretation based on Alexander Templates

3.1 An Example of Mode Inference based on AT

(1) Mode Inference Problem

Suppose that a program

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

is given and a top-level query "?- *reverse*(L_0, M_0)" is executed with its first argument L_0 instantiated to a ground term. 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 second arguments of "*append*" at calling time and the third argument at exiting time. How can we show it mechanically?

(2) Transformation of Program and Query

Let us transform the given program and query in the same way as "Alexander Templates." As for the program, the transformed program $\tilde{P}$ is as below:

```
call_reverse(L,N),cont([1,1],[X,L,M]) :- call_reverse([X|L],M).
call_append(N,[X],M),cont([1,2],[X,L,M,N]) :-
            cont([1,1],[X,L,M]),sol_reverse(L,N).
sol_reverse([X|L],M) :- cont([1,2],[X,L,M,N]),sol_append(N,[X],M).
sol_reverse([ ],[ ]) :- call_reverse([ ],[ ]).
call_append(N,K,M),cont([3,1],[Y,N,K,M]) :- call_append([Y|N],K,[Y|M]).
sol_append([Y|N],K,[Y|M]) :- cont([3,1],[Y,N,K,M]),sol_append(N,K,M).
sol_append([ ],K,K) :- call_append([ ],K,K).
```

As for the query, however, instead of each query "?- *reverse*(t, M)" with its first argument t ground, we consider a pair of a query and a mode substitution

?- *reverse*(L, M) $< L \Leftarrow \underline{ground} >$.

Hence, the transformed query $\tilde{Q}$ is as below:

call_reverse(L, M) $< L \Leftarrow \underline{ground} >$.

(3) Evaluation in the Domain of Modes

Let us evaluate the transformed program and query in the same way as "Alexander Templates." Then, similarly to the transformed query, we need to consider pairs of an atom and a mode substitution

$call\_A\ \mu$,

$sol\_A\ \nu$,

$cont([n,i],l_{n,i})\ \lambda$

to denote the mode information of the variables in the atoms $call\_A, sol\_A, cont([n,i],l_{n,i})$, where μ, ν, λ are mode substitutions. In the following, $sol\_A\mu$ is called the *corresponding mode-abstracted sol-atom of* $call\_A\mu$. Then, the bottom-up evaluation proceeds as follows:

Before the repetetion at step 1 in "$evaluate1$," Δ is initialized to a singleton set $\{call\_reverse(L,M) < L \Leftarrow \underline{ground} >\}$.

At the 1st repetetion, using the first clause in $\bar{P}$, new pairs

$call\_reverse(L,N) < L \Leftarrow \underline{ground} >$,

$cont([1,1],[X,L,M]) < X, L \Leftarrow \underline{ground} >$

are obtained, because, when $call\_reverse([X|L],M)$ and $call\_reverse(L',M')$, are unified under the condition that L' be ground, X and L are ground. ($call\_reverse(L',M') < L' \Leftarrow \underline{ground} >$ is a variant of $call\_reverse(L,M) < L \Leftarrow \underline{ground} >$ in Δ, and used to avoid the variable names conflict.)

Similarly, using the fourth clause in $\bar{P}$, a new pair

$sol\_reverse(L,M) < L, M \Leftarrow \underline{ground} >$

is obtained, because, when $call\_reverse([\,],[\,])$ and $call\_reverse(L',M')$ are unified under the condition that L' be ground, and when $sol\_reverse(L,M)$ and the head $sol\_reverse([\,],[\,])$ are unified under the condition that L be ground, L and M are ground. ($sol\_reverse(L,M) < L \Leftarrow \underline{ground} >$ is the corresponding mode-abstracted sol-atom of $call\_reverse(L,M) < L \Leftarrow \underline{ground} >$ in Δ, and used to re-form the mode-abstracted sol-atoms generated.)

The evaluation proceeds similarly to generate the following pairs:

2nd Repetetion: $call\_append(N,[X],M) < X, N \Leftarrow \underline{ground} >$,
$cont([1,2],[X,L,M,N]) < X, L, N \Leftarrow \underline{ground} >$.

3rd Repetetion: $call\_append(N,K,M) < N, K \Leftarrow \underline{ground} >$,
$cont([3,1],[Y,N,K,M]) < Y, N, K \Leftarrow \underline{ground} >$,
$sol\_append(N,[X],M) < N, X, M \Leftarrow \underline{ground} >$.

4th Repetetion: $sol\_append(N,K,M) < N, K, M \Leftarrow \underline{ground} >$.

3.2 A Formalization of the Mode Inference based on AT

Let us formalize the notions used in the previous example. Because our purpose is the explanation of the framework for the abstract interpretation based on AT, here we consider the simplest mode structure to make our explanation as simple as possible.

(1) Mode

A *mode* is one of the following 3 sets of terms:

$\underline{any}$: the set of all terms,

$\underline{ground}$: the set of all ground terms,

$\emptyset$: the empty set of terms.

The *instantition ordering of modes* is the ordering $\prec$ 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. Mode substitutions are denoted by μ, ν, λ. We assume that a mode 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 mode substitution explicitly. Hence, the empty mode substitution $<>$ assigns $\underline{any}$ to every variable.

The *joined mode substitution* of two mode substitutions μ and ν, denoted by $\mu \vee \nu$, is the substitution such that $(\mu \vee \nu)(X)$ is the least upper bound of $\mu(X)$ and $\nu(X)$ w.r.t. the instantiation ordering.

(2) Mode-abstracted Atom

Let A be an atom and μ be a mode substitution of the form

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

Then $A\mu$ is called a *mode-abstracted atom*, and denotes the set of all the atoms obtained by replacing each X_i in A with a term in $\underline{m_i}$. (Hereafter, we consider only the restriction of μ to the variables in A when $A\mu$ is considered.) A mode-abstracted atom $A\nu$ is called an *instance* of a mode-abstracted atom $A\mu$ when there exists a mode substitution λ such that $A\nu$ is $A(\mu \vee \lambda)$. A mode-abstracted atom $B\nu$ is called a *variant* of a mode-abstracted atom $A\mu$ when B is a variant of A and ν is obtained from μ by renaming the variables in the domain of μ accordingly.

(3) Unification of Mode-abstracted Atoms

Two mode-abstracted atoms $A\mu$ and $B\nu$ are said to be *unifiable* when $A\mu \cap B\nu \neq \emptyset$. Let A be an atom, $X_1, X_2, \ldots, X_k$ all the variables in A, μ a mode substitution

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

B an atom, $Y_1, Y_2, \ldots, Y_l$ all the variables in B, and ν a mode substitution

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

Then, how can we know whether $A\mu$ and $B\nu$ are unifiable, that is, whether there exists a unification of $A\sigma$ in $A\mu$ and $B\tau$ in $B\nu$? And, if there exists such a unification, what modes of terms are expected to be assigned to $Y_1, Y_2, \ldots, Y_l$ by the unifier?

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

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

The mode information of μ is propagated to the variables in B through η. Let's divide the mode propagation through η into two phases, *inwards mode propagation* and *outwards mode propagation*.

When a term t containing an occurrence of term s is instantiated to a term in $\underline{m}$, a mode containing all instances of *the occurrence* of term s is called an *inwards mode propagation of $\underline{m}$ from t to s*, denoted by $s/<t \Leftarrow \underline{m}>$. (Exactly speaking, some notation denoting the *occurrence* of s should be used instead of the term s itself.) It is computed as below:

$$s/ < t \Leftarrow \underline{m} >= \underline{m}.$$

Example 3.2.1 Let t be $[X|L]$ and $\underline{m}$ be $\underline{ground}$. Then
$X/ < [X|L] \Leftarrow \underline{ground} >= \underline{ground}$,
$L/ < [X|L] \Leftarrow \underline{ground} >= \underline{ground}$.

When each variable Z in term s is instantiated to a term in $\lambda(Z)$, a mode containing all instances of s is called an *outwards mode propagation of* λ *to* s, and denoted by s/λ. It is computed as below:

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

Example 3.2.2 Let s be $[X|L]$ and λ be $< X \Leftarrow \underline{ground}, L \Leftarrow \underline{ground} >$. Then
$s/\lambda = \underline{ground}$.

Let A, $X_1, X_2, \ldots, X_k$, μ, B, $Y_1, Y_2, \ldots, Y_l$ and ν be as before. Then, we can overestimate the unification of $A\mu$ and $B\nu$ as follows:

1. First, we can check the unifiability of $A\mu$ and $B\nu$ by the unifiability of A and B. If A and B are not unifiable, $A\mu$ and $B\nu$ are not unifiable. Otherwise, let η be an m.g.u. of A and B of the form
   $$< X_1 \Leftarrow t_1, X_2 \Leftarrow t_2, \ldots, X_k \Leftarrow t_k, Y_1 \Leftarrow s_1, Y_2 \Leftarrow s_2, \ldots, Y_l \Leftarrow s_l >.$$
2. Next, for each occurrence of a constant in $t_1, t_2, \ldots, t_k$, we can compute the mode assigned to the occurrence using the inwards mode propagation of μ. Similarly, for each occurrence of variable Z in $t_1, t_2, \ldots, t_k$, we can compute a mode containing all instances of the occurrence using the inwards mode propagation. By taking their least upper bound w.r.t. the instantiation ordering for all the occurrences of Z in t, we can compute a mode containing all instances of Z. If
   - the mode assigned to some occurrence of a constant is $\emptyset$, or
   - the mode assigned to some variable is $\emptyset$,

   $A\mu$ and $B\nu$ are not unifiable. Otherwise, we can compute the mode substitution λ for all the variables in $t_1, t_2, \ldots, t_k$ by collecting these mode assignments for the variables.
3. Then, we can overestimate the mode $\underline{n'_j}$ assigned to s_j using the outwards mode propagation of λ, hence, we can obtain a mode substitution ν' of the form
   $$< Y_1 \Leftarrow \underline{n'_1}, Y_2 \Leftarrow \underline{n'_2}, \ldots, Y_l \Leftarrow \underline{n'_l} >$$
   by collecting the modes for all the variables $Y_1, Y_2, \ldots, Y_l$ in B.
4. Last, $A\mu \cap B\nu$ is overestimated by $B(\nu \vee \nu')$.

The mode substitution $\nu \vee \nu'$ is called the *propagated mode substitution from* μ *to* ν *through* η, and denoted by "$\mu \xrightarrow{\eta} \nu$" or "$\nu \xleftarrow{\eta} \mu$."

(4) Transformation of Program and Query

The subprocedure "*transform*1" receives a program P and a mode-abstracted query $Q\lambda$, and returns a pair of a program and a mode-abstracted atom $(\tilde{P}, \tilde{Q}\lambda)$ in the same way as in Section 2.2.

(5) Evaluation in the Domain of Modes

In the bottom-up evaluation of AT in Section 2.2, if a new sol-atom sol_A' is generated at some repetetion, then there always exists a call-atom $call\_A$ in Δ generated at some previous repetetion such that A' is an instance of A and $call\_A$ is the initial source of the generation of sol_A'. As for the bottom-up evaluation in the domain of modes, a mode-abstracted atom generated is defined in the same way as AT except that, when a new mode-abstracted sol-atom $sol\_A'\mu$ is generated, it is re-formed to the mode-abstracted sol-atom $sol\_A\lambda$ to conform to the call-atom $call\_A\nu$ already in Δ.

Let $\tilde{C}$ be a clause in $\tilde{P}$, and Δ, Δ_{new} be sets of mode-abstracted atoms. Then, a mode substitution μ is said to be *generated from* (Δ, Δ_{new}) *using the body of* $\tilde{C}$ when either of the following conditions are satisfied.

1. The body of $\tilde{C}$ is "$call\_A$," and
   - there exists a mode-abstracted atom $call\_A'\mu'$ in Δ_{new} such that $call\_A'$ is unifiable with $call\_A$, say with m.g.u. θ,
   - μ is $\mu' \xrightarrow{\theta} <>$.
2. The body of $\tilde{C}$ is "$cont([n,i], l_{n,i}), sol\_A_i$," and
   - there exists a mode-abstracted atom $cont([n,i], l_{n,i})\nu$ in Δ,
   - there exists a mode-abstracted atom $sol\_A_i'\mu'$ in Δ such that sol_A_i' is unifiable with sol_A_i, say with m.g.u. θ,
   - μ is $\nu \vee (\mu' \xrightarrow{\theta} <>)$,
   - at least one of $cont([n,i], l_{n,i})\nu$ and $sol\_A_i'\mu$ is in Δ_{new}.

Let $\tilde{C}$ be a clause in $\tilde{P}$, Δ, Δ_{new} be sets of mode-abstracted atoms, μ be a mode substitution generated from (Δ, Δ_{new}) using the body of $\tilde{C}$. Then, a mode-abstracted atom $B\lambda$ is said to be *generated from* (Δ, Δ_{new}) *using* $\tilde{C}$ when one of the following conditions is satisfied.

1. B is a call-atom in the head of $\tilde{C}$, and λ is the restriction of μ to the variables in B.
2. B is a cont-atom in the head of $\tilde{C}$, and λ is the restriction of μ to the variables in B.
3. B' is a sol-atom in the head of $\tilde{C}$, and there exists a mode-abstracted call-atom $B\nu$ in Δ such that B' and B are unifiable, say with m.g.u. η, and λ is $\nu \xleftarrow{\eta} \mu$.

The subprocedure "*evaluate*1" receives $\tilde{P}$ and $\tilde{Q}\lambda$, and returns a set of mode-abstracted atoms in the same way as Section 2.2.

3.3 Correctness of the Mode Inference based on AT

This mode inference is safe, i.e., it does not miss any atoms at calling time and exiting time during the top-down execution. More precisely, the correctness is stated as below. The proof of the theorem crucially depends on the fact mentioned before that $B(\mu \xrightarrow{\eta} \nu)$ is a superset of $A\mu \cap B\nu$.

Theorem (Correctness of the Mode Inference)

Let P be a program, $Q\lambda$ be a mode-abstracted query, and $(\tilde{P}, \tilde{Q}\lambda)$ be the result of "*transform*1(P, Q)."

(a) If an atom $A\sigma$ appears at the leftmost of a goal during OLD resolution of a query in $Q\lambda$ using P, then "*evaluate*1$(\tilde{P}, \tilde{Q}\lambda)$" generates $call\_A\mu$ such that $A\sigma$ is in $A\mu$. (Correctness for Calling Patterns)

(b) If an atom is solved with solution $A\tau$ during OLD resolution of a query in $Q\lambda$ using P, then "$evaluate1(\tilde{P}, \tilde{Q}\lambda)$" generates $sol\_A\nu$ such that $A\tau$ is in $A\nu$. (Correctness for Exiting Patterns)

Proof. See Apendix.

Note that, because the set of modes is finite, there exist only finite mode-abstracted atoms, hence the repetetion at step 1 in "$evaluate1$" always terminates.

4. Discussion

4.1 Classification of Prolog Abstract Interpreters

In the abstract interpretation of Prolog programs, what we would like to analyze are the run-time properties of a given query when it is executed using the usual top-down Prolog interpreter. However, if we try to execute the query in the abstract domain using the usual top-down interpreter, we will immediately encounter the problem that the interpreter is more likely to enter a non-terminating computation loop even if the program is not left-recursive. Hence, it is more appropriate to start with an interpreter that has some correspondence to the usual top-down interpreter and that is less likely to enter a non-terminating computation loop when executed in the abstract domain. In particular, to avoid a non-terminating computation loop, some operation that is *bottom-up in nature* is inevitable. According to how the bottom-up operation is integrated, the frameworks of abstract interpretation are classified into the following three:

The first one is the *pure bottom-up abstract interpretation* approach, in which the bottom-up interpreter, i.e., hyper-resolution, is directly applied to a given program in the abstract domain (without any pre-processing of the program). Though the bottom-up interpreter is simple, it does not take the given top-level goal into consideration so that it is likely to waste time working on goals irrelevant to the top-level goal and ignore the precise run-time behavior of the top-down interpreter. This approach was applied to type inference by Kanamori and Horiuchi [6], and generalized by Marriott and Søndergard [12].

The second one is the *two-phase hybrid abstract interpretation* approach, in which 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 are derived, and a superset of the least solution of the simultaneous recurrence equations is obtained using a bottom-up approximation. The reason for the separation into two phases, 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 our attention on just the goals relevant to the top-level goal and capture the precise run-time behavior of the top-down interpreter. The other is that, by solving by the bottom-up approximation, we can obtain solutions without entering a non-terminating computation loop. This approach was proposed by Mellish [14] in order to give a theoretical foundation to his practical techniques for analyzing determinacy, modes and shared structures [13]. The correspondence between Mellish's approach and our AT-based approach is discussed in Section 4.2.

The third one is the *one-phase hybrid abstract interpretation* approach, in which a given query is executed in the abstract domain using some top-down interpreter with memo-ization. The top-down interpreter with memo-ization proceeds in the same way as the usual top-down interpreter except that the solutions already obtained are memo-ed and utilized to

solve the same goal without repeating the same execution. (The utilization of solutions corresponds to the bottom-up interpretation.) Hence, it is less likely to enter a non-terminating computation loop (than the usual top-down interpretation) and wastes less time working on goals irrelevant to the top-level goal (than the usual bottom-up interpretation) so that the corresponding abstract interpreter achieves the same effects as Mellish's approach without the separation into two phases. This approach was investigated by Kanamori, Kawamura, Maeji and Horiuchi [7], [10], Bruynooghe [2], Debray [4] and Mannila and Ukkonen [11]. The correspondence between this approach and our AT-based approach is discussed in Section 4.3.

4.2 Correspondence to Abstract Interpretation by Mellish

Mellish's paper [14] first explains a framework that derives simultaneous recurrence equations for input (the set of atoms at calling time) and output (the set of atoms at exiting time) and obtains some supersets of their least solutions using the bottom-up approximation, and then later refines the framework by partially evaluating the simultaneous recurrence equations to make them more convenient for computing the supersets of input and output.

The latter framework of his approach is closely related to the naive version of our approach. First of all, deriving the partially evaluated simultaneous recurrence equations corresponds to our "$transform0$." Second, obtaining the supersets of their least solutions in some abstract domain by bottom-up approximation corresponds to our "$evaluate0$" applied to the abstract domain. Note that, due to the use of cont-atoms, our refined version with "$transform1$" and "$evaluate1$" in Section 2.2 is more time-saving than the naive version with "$transform0$" and "$evaluate0$."

4.3 Correspondence to Abstract Interpretation based on OLDT Resolution

OLDT resolution is one of the top-down interpreters with memo-ization [27]. Given a top-level query "?- B", OLDT resolution initializes the computation by generating a tree consisting of a single root node labelled with B, called *initial OLDT tree*. In general, each node of an OLDT tree is labelled with a sequence of atoms, and the atoms are processed from left to right to generate the labels of child nodes. Once the leftmost atom of a node is solved (and an instance of the atom sequence except the leftmost atom appears at the leftmost of a descendant node), the solution is memo-ed in a table. At each step, OLDT resolution extends the OLDT tree at all the possible nodes

- either in the same way as the usual top-down interpreter (OLD resolution) as far as the leftmost atom of the label has not appeared before,
- or by utilizing the already obtained solutions in the table to solve the leftmost atom of the same form.

One-to-one correspondence between the behaviors of AT and OLDT resolution was established by Seki [17],[18].

Theorem (AT and OLDT)

Let P be a program, Q be a query, and $(\tilde{P}, \tilde{Q})$ be the result of "$transform1(P, Q)$."

- An atom $call\_A\sigma$ is generated at the k-th repetetion in $evaluate1(\tilde{P}, \tilde{Q})$ if and only if a node with leftmost atom $A\sigma$ is generated at the k-th extension of the initial OLDT tree of Q using P.

- An atom $sol\_A\tau$ is generated at the k-th repetetion in $evaluate1(\tilde{P}, \tilde{Q})$ if and only if a node with solution $A\tau$ is generated at the k-th extension of the initial OLDT tree of Q using P.

Proof. Seki's original AT [17],[18] does not generate an atom when it is an instance of an already generated atom in the bottom-up interpretation phase. In addition, Seki's original proof [17],[18] has shown the correspondence between his AT and slightly modified SLD-AL resolution [20]. The proof, however, goes in the same way for the correspondence between our modified AT and OLDT resolution. See Seki [17],[18] for the details.

The OLDT-based abstract interpretation executes a given query in the abstract domain using the OLDT resolution [7],[10]. The correspondence above immediately implies the one-to-one correspondence between AT-based abstract interpretation and OLDT-based abstract interpretation.

Theorem (AT-based and OLDT-based Abstract Interpretations)
Let P be a program, $Q\lambda$ be a mode-abstracted query, and $(\tilde{P}, \tilde{Q}\lambda)$ be the result of "$transform1(P, Q\lambda)$."

- A mode-abstracted atom $call\_A\mu$ is generated at the k-th repetetion in $evaluate1(\tilde{P}, \tilde{Q}\lambda)$ if and only if a node with leftmost atom $A\mu$ is generated at the k-th extension of the initial OLDT tree of $Q\lambda$ using P.
- A mode-abstracted atom $sol\_A\nu$ is generated at the k-th repetetion in $evaluate1(\tilde{P}, \tilde{Q}\lambda)$ if and only if a node with solution $A\nu$ is generated at the k-th extension of the initial OLDT tree of $Q\lambda$ using P.

Proof. Immediate from the theorem above.

5. Conclusions

We have presented a framework for logic program analysis based on "Alexander Templates" with its applications to mode inference, and shown the relation to Mellish's abstract interpretation.

Acknowledgements

This research was done as a part of the Fifth Generation Computer Systems project of Japan. We would like to thank Dr. K. Fuchi (Director of ICOT) for the opportunity of doing this research, and Dr. K. Furukawa (Deputy Director of ICOT) and Dr. R. Hasegawa (Chief of ICOT 1st Laboratory) for their advice and encouragement. We would also like thank Mr. H. Seki (Mitsubishi Electric Corporation) for his valuable suggestions.

References

[1] Bancilhon, F. and Ramakrishnan, R., "An Amateur's Introduction to Recursive Query Processing Strategies," *Proc. of the ACM-SIGMOD Conference* :16–52, Washington DC., 1986.

[2] Bruynooghe, M., A Practical Framework for the Abstract Interpretation of Logic Programs, to appear *the Journal of Logic Programming*, 1989.

[3] Cousot,P. and Cousot,R., 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 :238–252, Los Angeles, 1977.

[4] Debray,S.K. and Warren,D.S., Automatic Mode Inference for Prolog Programs, *Proc. of 1986 Symposium on Logic Programming* :78–88, Salt Lake City, September 1986.

[5] Dietrich,S.W., Extension Tables: Memo Relations in Logic Programming, *Proc. of 1987 Symposium on Logic Programming* :264–272, San Francisco, August 1987.

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

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

[8] Kanamori,T., Horiuchi,K. and Kawamura,T., Detecting Functionality of Logic Programs based on Abstract Hybrid Interpretation, ICOT Technical Report TR-331, Tokyo, December 1987.

[9] Kanamori,T., Kawamura,T. and Horiuchi,K., Detecting Termination of Logic Programs based on Abstract Hybrid Interpretation, ICOT Technical Report TR-398, Tokyo, December 1987.

[10] Kanamori,T. and Kawamura,T., Logic Program Analysis by Abstract Hybrid Interpretation, ICOT Technical Report TR-485, Tokyo, July 1989.

[11] Mannila,H. and Ukkonen,E., Flow Analysis of Prolog Programs, *Proc. of 1987 Symposium on Logic Programming* :205–214, San Francisco, August 1987.

[12] Marriott, K. and Søndergaard,H., Bottom-up Abstract Interpretation of Logic Programs, *Proc. of Fifth International Conference and Symposium on Logic Programming* :733–748, Seattle, August 1988.

[13] Mellish,C.S., Some Global Optimizations for A Prolog Compiler, *J. Logic Programming* 2, 1 :43–66 (1985).

[14] Mellish,C.S., Abstract Interpretation of Prolog Programs, *Proc. of 3rd International Conference on Logic Programming* :463–474, London, July 1986.

[15] Ramakrishnan,R., Magic Templates: A Spellblinding Approach to Logic Programs, *Proc. of 5th International Conference and Symposium on Logic Programming* :140–159, Seattle, August 1989.

[16] Rohmer,R., Lescouer,R. and Kerisit,J.-M., The Alexander Method — A Technique for the Recursive Axioms in Deductive Databases, *New Generation Computing* 4,3 :273–285 (1986).

[17] Seki,H., On the Power of Alexander Templates, *Proc. of 8th ACM Symposium on Principles of Database Systems* :150–159, Pliladelphia, March 1989.

[18] Seki,H., On the Power of Alexander Templates, ICOT Technical Report TR-5??, ICOT, Tokyo, December 1989.

[19] Tamaki,H. and Sato,T., OLD Resolution with Tabulation, *Proc. of 3rd International Conference on Logic Programming* :84–98, London, July 1986.

[20] Vieille,L., A Database-complete Proof Procedure based on SLD Resolution, *Proc. of 4th International Conference on Logic Programming* :74–103, Melbourne, May 1987.

Appendix Proof of the Correctness of the Mode Inference based on AT

(1) Definitions for OLD Resolution

Let us first formalize the top-down interpretation. In the following, a *goal* is a (possibly empty) sequence of atoms. Goals are denoted by G, H, and the empty goal is denoted by $\Box$.

Definition OLD Tree

An *OLD tree* is a tree such that each node is labelled with a goal, and each edge is labelled with a substitution. An *OLD tree of atom* A is an OLD tree whose root node is labelled with a goal consisting of only one atom A. When a node in an OLD tree is labelled with "$A_1, A_2, \ldots, A_n$," the atom A_1 is called the *head atom of* the node.

Definition OLD Resolution

A terminal node of OLD tree T labelled with "$A, A_2, \ldots, A_n$" is said to be *OLD resolvable* using program P when there is some definite clause "B :- $B_1, B_2, \ldots, B_m$" ($m \geq 0$) in P such that A and B are unifiable, say by an m.g.u. θ. The (possibly empty) goal "$(B_1, B_2, \ldots, B_m, A_2, \ldots, A_n)\theta$" is called the *OLD resolvent*, and the substitution θ is called the *substitution of the OLD resolution*.

Definition Initial OLD Tree

The *initial OLD tree* of atom A is the OLD tree T_0 consisting of only the root node labelled with A.

Definition Extension of OLD Tree

An *immediate extension* of OLD tree T using program P is the result of the following operations, when a node v of OLD tree T is OLD resolvable using P.

- Let $C_1, C_2, \ldots, C_k$ ($k \geq 1$) be all the clauses with which the node v is OLD resolvable, and $G_1, G_2, \ldots, G_k$ be the respective OLD resolvents. Then add k child nodes 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 θ_i, where θ_i is the substitution of the OLD resolution with C_i.

An OLD tree T_{ext} is an *extension* of OLD tree T using program P if T_{ext} is obtained from T through successive application of immediate extensions using P.

Definition OLD Subrefutation and OLD Partial Subrefutation

An *OLD subrefutation of* an atom and an *OLD subrefutation of* a goal are paths in an OLD tree (not necessarily starting from the root node) which are simultaneously defined inductively as follows:

(a) A path with length more than 0 starting from a node is an *OLD subrefutation of* an atom $A\sigma$ with solution $A\tau$ when
- the initial node is labelled with a goal of the form "$A\sigma, G\sigma$," the initial edge with a substitution θ, and the last node with a goal of the form "$G\tau$," and
- the node next to the initial node is labelled with a goal of the form "$(A_1, A_2, \ldots, A_n)\theta$, $G\sigma\theta$," and the path except the initial node and the initial edge is a subrefutation of "$(A_1, A_2, \ldots, A_n)\theta$" with solution "$(A_1, A_2, \ldots, A_n)\eta$ ($n \geq 0$)," and
- τ is $\sigma\eta$.

(b1) A path with length 0, i.e., a path consisting of only one node, is an *OLDT subrefutation of* "$\square$" with solution "$\square$."

(b2) A path with a length more than 0 is an *OLD subrefutation of* a goal "$(A_1, A_2, \ldots, A_n)\sigma$" with solution "$(A_1, A_2, \ldots, A_n)\tau$" ($n > 0$) when
- the initial node is labelled with a goal of the form "$(A_1, A_2, \ldots, A_n)\sigma, H\sigma$," and the last node with a goal of the form "$H\tau$,"
- the path is the concatenation of a subrefutation of $A_1\sigma$ with solution $A_1\sigma\eta_1$, a subrefutation of $A_2\sigma\eta_1$ with solution $A_2\sigma\eta_1\eta_2, \ldots$, a subrefutation of $A_n\sigma\eta_1\eta_2\cdots\eta_{n-1}$ with solution $A_n\sigma\eta_1\eta_2\cdots\eta_{n-1}\eta_n$, and
- τ is $\sigma\eta_1\eta_2\cdots\eta_{n-1}\eta_n$.

In particular, a subrefutation of A is called a *unit subrefutation of* A.

A path in an OLD tree starting from a node with head atom A is called a *partial subrefutation of* A when it does not contain any subrefutation of A as its prefix.

(2) Definitions for Alexander Templates

As for the notions of "Alexander Templates," some of the following definitions overlap with the contents of Section 2. We have repeated them to make clear the correspondence between the notions of OLD resolution and those of AT. Hereafter, $\tilde{P}$ and $\tilde{Q}$ denote the result of "$translate0(P, Q)$."

Definition Atom Set
A set of atoms is called an *atom set* when it consists of call-atoms or sol-atoms.

Definition Generated Atom
Let $\tilde{C}$ be a clause in $\tilde{P}$, and Γ be a set of atoms. Then, atom $B\theta$ is said to be *generated from* Γ *using* $\tilde{C}$ when

- B is the head atom of $\tilde{C}$, and
- there exists a sequence of atoms in Γ unifiable with the sequence of the body atoms of $\tilde{C}$, and θ is an m.g.u.

Definition Initial Atom Set
The *initial atom set of* "$call\_B$" is the set of atoms $\{call\_B\}$.

Definition Extension of Atom Set
An *immediate extension* of atom set Γ in $\tilde{P}$ is
$\Gamma' \cup \Gamma_0$,
where Γ' is the set of all the atoms generated from Γ using some clause in $\tilde{P}$, and Γ_0 is an initial atom set. An atom set Γ_{ext} is an *extension of* atom set Γ in $\tilde{P}$ if Γ_{ext} is obtained from Γ through successive application of immediate extensions.

The notions for the mode inference based on AT are defined similarly.

(3) Proof of the Correctness

The following Lemma A1 reduces the correctness of the mode inference based on the refined "Alexander Templates" to that on the naive "Alexander Templates," which is in turn guaranteed by the following Lemma A2. Let P be a program, Q be a query, and $(\tilde{P}, \tilde{Q})$ be the result of "$transform0(P, Q)$."

Lemma A1
Let Γ_∞ and Δ_∞ be the set of all the mode-abstracted call- and sol-atoms generated by the naive "Alexander Templates" and the refined "Alexander Templates," respectively. Then, Γ_∞ and Δ_∞ are identical.

Proof. Obvious by induction on the number of steps required to generate the atoms.

The theorem in Section 3.3 is restated as follows:

Theorem (Correctness of the Mode Inference)
Let $B\lambda$ be a mode-abstracted atom, T_0 be the initial OLD tree of an atom in $B\lambda$, and Δ_0 be the initial mode-abstracted atom set of $call\_B\lambda$.

(a) If some extension of T_0 contains a node with head atom $A\sigma$, then some extension of Δ_0 contains a mode-abstracted atom $call\_A\mu$ such that $A\sigma$ is in $A\mu$. (Correctness for Calling Patterns)

(b) If some extension of T_0 contains a subrefutation with solution $A\tau$, then some extension of Δ_0 contains $sol\_A\nu$ such that $A\tau$ is in $A\nu$. (Correctness for Exiting Patterns)

The theorem is an immediate consequence of the following lemma:

Lemma A2

Let T be an extension of an initial OLD tree, and Γ be an extension of an initial mode-abstracted atom set.

(a) If T contains a partial OLD subrefutation of $A\sigma$ whose last node has leftmost atom $B\tau$, and Γ contains $call\_A\mu$ such that $A\sigma$ is in $A\mu$, then some extension of Γ contains $call\_B\nu$ such that $B\tau$ is in $B\nu$.

(b) If T contains an OLD subrefutation of $A\sigma$ with solution $A\tau$, and Γ contains $call\_A\mu$ such that $A\sigma$ is in $A\mu$, then some extension of Γ contains $sol\_A\nu$ such that $A\tau$ is in $A\nu$.

Proof. The proof is by simultaneous induction on the length of (partial) subrefutations.

Proof of Part (a):

Let r be a partial subrefutation starting from node u and ending with node v.

Base Case: If the length of r is 1, then "$B\tau$" is identical to "$A\sigma$," hence, from the assumption, "$call\_A\mu$" is in Γ.

Induction Step: If the length of r is greater than 1, there exists a clause, say of the form

"A_0 :- $A_1, A_2, \ldots, A_m$"

with which u is resolvable. Let u_0 be the immediate child node of u labelled with resolvent

$(A_1, A_2, \ldots, A_m)\theta_0, \ldots$

Let the path from u_0 to v be divided into

r_1 : subrefutation of "$A_1\theta_0$" with solution "$A_1\theta_0\theta_1$,"

r_2 : subrefutation of "$A_2\theta_0\theta_1$" with solution "$A_2\theta_0\theta_1\theta_2$,"

$\vdots$

r_{i-1} : subrefutation of "$A_{i-1}\theta_0\theta_1\theta_2\cdots\theta_{i-2}$" with solution "$A_{i-1}\theta_0\theta_1\theta_2\cdots\theta_{i-1}$,"

r_i : partial subrefutation of "$A_i\theta_0\theta_1\theta_2\cdots\theta_{i_1}$" with length shorter than r.

Because the clause

$call\_A_1$:- $call\_A_0$

is in $\tilde{P}$, the immediate extension of Γ includes $call\_A_1\mu_1$ containing $call\_A_1\theta_0$ due to the property of the mode propagation. From the induction hypothesis for part (b), some extension of Γ includes $sol\_A_1\mu_2$ containing $sol\_A_1\theta_0\theta_1$. Similarly, some extension of Γ includes

"$call\_A_2\mu_2$" containing "$call\_A_2\theta_0\theta_1$,"

"$sol\_A_2\mu_3$" containing "$sol\_A_2\theta_0\theta_1\theta_2$,"

$\vdots$

"$call\_A_{i-1}\mu_{i-1}$" containg "$call\_A_{i-1}\theta_0\theta_1\theta_2\cdots\theta_{i-2}$,"

"$sol\_A_{i-1}\mu_i$" containing "$sol\_A_{i-1}\theta_0\theta_1\theta_2\cdots\theta_{i-1}$,"

"$call\_A_i\mu_i$" containing "$call\_A_i\theta_0\theta_1\theta_2\cdots\theta_{i-1}$."

Then, from the induction hypothesis for part (a), some extension of Γ includes "$call\_B\nu$" containing "$call\_B\tau$."

Proof of Part (b):

Let r be a subrefutation starting from node u and ending with node v.

Base Case: If the length of r is 1, there exists a unit clause, say of the form

"A_0"

with which u is resolvable. Becaues a clause

sol_A_0 :- $call\_A_0$

is in $\tilde{P}$, the immediate extension of Γ includes "$sol\_A_0\nu$" containing "$sol\_A\tau$" due to the property of the mode propagation.

Induction Step: If the length of r is greater than 1, there exists a clause, say of the form

"A_0 :- $A_1, A_2, \ldots, A_m$"

with which u is resolvable. Let u_0 be the immediate child node of u labelled with resolvent

$(A_1, A_2, \ldots, A_m)\theta_0, \ldots$

Let the path from u_0 to v be divided into

r_1 : subrefutation of "$A_1\theta_0$" with solution "$A_1\theta_0\theta_1$,"

r_2 : subrefutation of "$A_2\theta_0\theta_1$" with solution "$A_2\theta_0\theta_1\theta_2$,"

$\vdots$

r_m : subrefutation of "$A_m\theta_0\theta_1\theta_2\cdots\theta_{m-1}$" with solution "$A_m\theta_0\theta_1\theta_2\cdots\theta_m$."

Because the clause

$call\_A_1$:- $call\_A_0$

is in $\tilde{P}$, the immediate extension of Γ includes "$call\_A_1\nu_0$" containing "$call\_A_1\theta_0$." From the induction hypothesis for part (b), some extension of Γ includes "$sol\_A_1\nu_1$" containing "$sol\_A_1\theta_0\theta_1$" due to the property of the mode propagation. Similarly, some extension of Γ includes

"$call\_A_2\nu_1$" containing "$call\_A_2\theta_0\theta_1$,"

"$sol\_A_2\nu_2$" containing "$sol\_A_2\theta_0\theta_1\theta_2$,"

$\vdots$

"$call\_A_m\nu_{m-1}$" containing "$call\_A_m\theta_0\theta_1\theta_2\cdots\theta_{m-1}$,"

"$sol\_A_m\nu_m$" containing "$sol\_A_m\theta_0\theta_1\theta_2\cdots\theta_m$."

Then, because the clause

sol_A_0 :- $call\_A_0, sol\_A_1, \ldots, sol\_A_m$

is in $\tilde{P}$, some extension of Γ includes "$sol\_A\nu$" containing "$sol\_A\theta_0\theta_1\theta_2\cdots\theta_m$," i.e., "$sol\_A\tau$" due to the property of the mode propagation.