

# PL/SQL 변경영향 분석기 요약 해석 디자인

김용기, 이동권, 김종권

2015년 12월 14일

## 제 1 절 출발 언어: PL/SQL 핵심 언어

PL/SQL 언어 명세로부터 현장에서 사용하는 부분 만을 요약한 PL/SQL 핵심 언어(Core Language)를 출발 언어로 정의하여 코드를 파싱한다. PL/SQL 언어는 등장한지 25년이 넘는 역사 동안 수많은 기능이 추가되어 언어 명세가 방대하기 때문에[?][?] 실제 사용하는 기능만을 정의하여 파서와 분석기를 제작하는 것이 필수적이다.

그림 1: PL/SQL 예제 코드

```
1 CREATE PACKAGE BODY pkg1 AS
2   CREATE PROCEDURE main(in_name    IN  VARCHAR2(10),
3                         out_score OUT NUMBER)    IS
4     v_id VARCHAR2(100) := '-';
5   BEGIN
6     SELECT id, score INTO v_id, out_score FROM student
7     WHERE name=in_name AND rownum=1;
8
9     FOR rec IN (SELECT name FROM student) LOOP
10       DBMS_OUTPUT.put_line('name: ' || rec.name);
11     END LOOP;
12   END main;
13 END pkg1;
```

그림 1은 Oracle 공식 언어 명세[?]와 각종 참고 자료를 참고하여 실제 반도체에서 사용되는 핵심 언어를 정의한 것이다. 트랜잭션이나 대량 데이터 처리와 관련된 SQL 문법은 대상 시스템에서 사용하지 않기 때문에 핵심 언어에 포함되지 않았다.

PL/SQL의 가장 큰 특징은 SQL 문장을 언어 수준에서 지원하는 것이다. 그림 1의 예제를 통해 PL/SQL 핵심 언어의 특징을 간략히 정리할 수 있다.

- 여러 함수/프로시저를 패키지라는 이름 공간으로 묶는다.
- 프로시저는 입력용 파라메터와 출력용 파라메터를 구분하여 받는다.
- 블록은 BEGIN ... END 로 선언된다.
- 변수 선언(초기화)는 블록 앞 부분에서만 가능하다.
- 문자열은 작은 따옴표로 묶는다.
- SQL 문장을 언어 레벨에서 지원한다.

그림 2: PL/SQL 핵심 언어

```

package → var external_decl*
external_decl → FUNCTION type var (type var)* block
               | PROCEDURE var (type var)* block
block → decl* stmt* exception_handler*
exception_handler → var stmt*
                   | OTHERS stmt*
decl → type var exp?                                (w/ initialization)
       | CURSOR var(exp*) select      (parameterized cursor)
       | typedef
typedef → TABLE type type var
           | RECORD (type var exp?)+ var      (initialized fields)
           | VARRAY type var
type → native_type
       | id                                     (type name)
stmt → label stmt_unit
stmt_unit → lexp := exp
           | IF exp stmt* (ELSE stmt*)?
           | LOOP stmt* | WHILE exp stmt* | FOR id exp stmt*
           | GOTO label | EXIT label? exp? | CONTINUE label? exp?
           | id(exp*)                               (proc call)
           | RETURN exp?
           | sql
           | OPEN var(exp*)
           | CLOSE var
           | FETCH var id+                      (from, to)
           | block
           | RAISE var
sql → select id+                                (from, to)
       | DELETE table exp                         (cond)
       | UPDATE table (id exp)+ exp        ((field, value)s, cond)
       | INSERT table (id+)? exp+         (fields, values)
select → SELECT table+ id+ exp?            (fields, cond)
         | select sop select
sop → UNION | MINUS | INTERSECT | UNIONALL
table → id
       | select
exp → lexp                                         (incl. func call)
       | const | exp bop exp | uop exp | (exp+)
id → var+
lexp → (var(exp*))*                           (incl. collection ref)
var → string
       | string e*                           (Id call)

```

PL/SQL 언어의 본질적인 모습은 다음 절의 중간 언어 변환 과정을 통해 더욱 잘 들어나게 된다.

## 제 2 절 중간 언어 변환

중간 언어 변환은 PL/SQL 핵심 언어를 설탕 구조(Syntactic Sugar)가 적절히 녹이고 분석에 적합한 형태로 프로그램을 변환하여 출발 언어에 중립적인 중간 언어로 바꾸는 작업이다. 예를 들어 PL/SQL에는 3가지 종류의 루프가 있는데 중간 언어에는 `while` 문법만 있기 때문에 변환 과정에서 하나로 녹여진다. 분석에 필요한 추가 정보도 만드는데, 파싱된 식별자에는 타입이 없기 때문에 식별자와 선언된 타입을 심볼테이블로 매핑하고, 같은 이름으로 쓰인 식별자들을 구별하는 알파 변환을 수행한다.

각 문법 요소 별 변환 규칙은 다음과 같다.

**타입 토큰** `TYPE`으로 선언되는 타입 식별자는 심볼 테이블에 등록 된다.

<pre>CREATE PACKAGE t AS   TYPE tr IS RECORD(f VARCHAR2);   TYPE tt IS TABLE OF INTEGER     INDEX BY PLS_INTEGER; END t;</pre>	심볼테이블 : { <code>tr: record(STRING, f),</code> <code>tt: table(INT, INT)</code> }
--	--

**패키지** `pkg`로 변환된다. `pkg`는 패키지에 선언되어있는 함수나 프로시저 이름과 본문을 함수 테이블로 저장하고 패키지 안에 사용되는 각종 식별자의 타입을 심볼테이블로 저장한다.

<pre>CREATE PACKAGE BODY p AS   PROCEDURE f (x IN NUMBER,                y OUT VARCHAR2)   IS     z NUMBER;   BEGIN ... END f;    FUNCTION g (x NUMBER)     RETURN NUMBER IS     BEGIN ... END g; END p;</pre>	<pre>p = pkg {   함수테이블: {     f: [ /* 함수 본문*/ ... ],     g: [ /* 함수 본문*/ ... ]   },   심볼테이블: {     f: type_f, g: type_g,     x1: INT, y2: STRING, z3: INT,     ...   } }</pre>
--	--

**프로시저/함수** (리턴타입, 함수인자 리스트) 모양으로 변환된다. 프로시저의 경우에는 리턴타입이 `None`으로 표기된다. 함수인자들은 (인자 타입, `IN/OUT`, 인자 이름)으로 표현되어 있다.

<pre>PROCEDURE f (   x IN NUMBER,   y OUT VARCHAR2 );  FUNCTION g (x NUMBER)   RETURN NUMBER;</pre>	<pre>type_f = fun(None, [   (INT, IN, x1),   (STRING, OUT, y2) ]);  type_g = fun(INT, [   (INT, IN, x5) ]);</pre>
---	---

**블록(Body Block)** PL/SQL에서 블록 안에 `BEGIN`을 사용하여 블록을 삽입 하여 변수의 선언과 예외 처리를 할 수 있는데, 이 경우 변수의 이름 변경 후에 `try`문으로 변환한다.

BEGIN z := x.f; y := z; DECLARE x NUMBER(6); BEGIN y := x; EXCEPTION WHEN OTHERS THEN y := 0; END; END f;	[ z3 := x1.f, y2 := z3; try ( [ y2 := x4 ], { OTHERS: [ y2 := 0 ] }) ]
--	--

**함수와 배열** PL/SQL은 arr(i)과 같이 배열 인덱스에 접근한다. 따라서 파싱된 모양만으로는 함수 호출과 구분 되지 않는다. 중간 언어 변환 과정에서 타입 정보를 이용하여 이를 구별한다. 다음 예제에서 y := f(1)은 f(1)로 변환되고 z := x(1)은 x[1]로 변환 된것을 볼 수 있다.

PROCEDURE g IS TYPE t IS VARRAY(10) OF NUMBER; x t; y x%TYPE; z x%TYPE; -- typeof(x) BEGIN y := f(1); -- f(1); z := x(1); -- x[1]; END;	함수테이블: { g: [ y2 := f(1), z3 := x1[1] ] }
--	--

**각종 반복문** 반복문의 조건문 형식에 맞는 while문으로 변환된다.

LOOP ... END LOOP;	while (true, [ ... ])
FOR i IN 1..10 LOOP ... END LOOP;	while (i IN (1,10), [ ... ])
WHILE x <= 10 LOOP ... END LOOP;	while (x <= 10, [ ... ])

**GOTO와 EXIT** 모두 goto로 변환된다. 단, 루프 안에서 탈출을 위해 EXIT가 사용되었을 경우 명시적으로 레이블을 만든다. 아래 예제에서 label "out"을 만든 것을 볼 수 있다.

<<start>> ... GOTO start; ... GOTO end; ... <<end>>	label "start"; ... goto "start"; ... goto "end"; ... label "end";
LOOP ... EXIT WHEN x > 10; END LOOP;	while(true, [ ...; if (x > 10, goto "out") ]) label "out"

**SELECT문 INTO**에 지정된 변수에 대한 할당문의 형태로 변환된다. SELECT는 다음과같이 (테이블, Column 리스트, 조건문)으로 변환된다.

SELECT f,g INTO x,y FROM t WHERE cond	(x,y) := select(t,(f,g),cond);
SELECT * INTO x,y FROM t WHERE cond	(x,y) := select(t,[],cond);

**UPDATE문과 INSERT문** SELECT문과 거의 똑같은 형태로 다음과 같이 변환된다.

UPDATE t SET f = exp WHERE cond	$t := \text{update}(t, (f, \text{exp}), \text{cond})$
INSERT INTO t VALUES exp	$t := \text{insert}(t, \text{exp})$

중간 언어의 완전한 정의는 그림 3과 같다.

그림 3: 중간 언어 정의

<i>Packages</i>	$pkg \rightarrow x \ proc^* (x \ \tau)^*$	<i>(pkg id, procedure decls, typemap)</i>
<i>Procedures</i>	$proc \rightarrow x \ stmt$	
<i>Types</i>	$\tau \rightarrow \text{int}   \text{real}   \text{string}   \text{unit}   \text{bool}$ $\quad \quad \quad \mid \text{exception}$ $\quad \quad \quad \mid \text{table } \tau \ \tau$ $\quad \quad \quad \mid \text{array } \tau \ e$ $\quad \quad \quad \mid \text{record } (\tau \ x)^*$ $\quad \quad \quad \mid \text{fun } \tau \ (\tau \ (\text{in} \text{inout}) \ x)^*$ $\quad \quad \quad \mid \text{cursor } e_{\text{sql}} \ (\tau \ x)^*$ $\quad \quad \quad \mid \text{pkg } x$	<i>(primitive types)</i> $\quad \quad \quad \mid \text{(key type, value type)}$ $\quad \quad \quad \mid \text{(value type, size)}$ $\quad \quad \quad \mid \text{(field items)}$ $\quad \quad \quad \mid \text{(return type, parameters)}$ $\quad \quad \quad \mid \text{(sql expr, parameters)}$ $\quad \quad \quad \mid \text{(package type)}$
<i>Statements</i>	$stmt \rightarrow le := e$ $\quad \quad \quad \mid \text{if } e \ stmt \ stmt$ $\quad \quad \quad \mid \text{while } e \ stmt$ $\quad \quad \quad \mid \text{goto } label$ $\quad \quad \quad \mid \text{label } label$ $\quad \quad \quad \mid \text{call } le \ e^*$ $\quad \quad \quad \mid \text{return } e$ $\quad \quad \quad \mid \text{try } stmt \ (x \ stmt)^*$ $\quad \quad \quad \mid \text{stmt; stmt}$	<i>(Assign)</i>
<i>L-expressions</i>	$le \rightarrow x$ $\quad \quad \quad \mid le.x$ $\quad \quad \quad \mid le[e]$	<i>(record id.field)</i> $\quad \quad \quad \mid \text{(array id}[idx]\text{])}$
<i>Expressions</i>	$e \rightarrow le$ $\quad \quad \quad \mid const$ $\quad \quad \quad \mid le(e^*)$ $\quad \quad \quad \mid e \ bop \ e$ $\quad \quad \quad \mid uop \ e$ $\quad \quad \quad \mid cop \ e$ $\quad \quad \quad \mid sql$	<i>function call</i>
<i>Constants</i>	$const \in \{\text{null}, \text{true}, \text{false}\} \cup \mathbb{Z} \cup \mathbb{R} \cup \text{strings}$	
<i>SQLs</i>	$sql \rightarrow \text{select } e^+ \ e^+ \ e$ $\quad \quad \quad \mid \text{insert } le \ e^+ \ e^+$ $\quad \quad \quad \mid \text{update } le \ (le \ e)^+ \ e$ $\quad \quad \quad \mid \text{delete } le \ e$ $\quad \quad \quad \mid \text{sql sop sql}$	<i>(transfer, table, predicate)</i> $\quad \quad \quad \mid \text{(table, record value)}$ $\quad \quad \quad \mid \text{(table, (field, value)s, exp(cond))}$ $\quad \quad \quad \mid \text{(table, exp(cond))}$
<i>Variables</i>	$x$	
<i>Labels</i>	$label$	
<i>BinaryOps</i>	$bop \rightarrow aop \mid rop \mid lop$	
<i>UnaryOps</i>	$uop$	
<i>CursorOps</i>	$cop \rightarrow \text{notfound} \mid \text{rowid}$	
<i>ArithmeticOps</i>	$aop$	
<i>RelationalOps</i>	$rop$	
<i>LogicalOps</i>	$lop$	
<i>SQLqueryOps</i>	$sop$	

## 제 3 절 대상 언어: CFG 언어

중간 언어로 변환된 PL/SQL 코드는 CFG(제어 흐름 그래프) 언어로 최종 변환되어 분석기에 전달된다.

CFG로 변환하는 이유는 프로그램 구조를 단순화하여 분석을 더욱 쉽게 만들기 위해서다. 변환되는 과정에서 함수 인라이닝, 함수 평탄화, 예외 처리 평탄화 작업을 통해 제어 흐름을 단순화 시켜 호출스택이나 예외처리 스택 없이 실행 의미를 정의할 수 있도록 돋는다. 최소한의 부품들로 프로그램의 의미를 정의할 수 있어 분석기 설계, 검증, 구현이 훨씬 수월해진다.

그림 4: CFG 언어 정의

$$\begin{aligned}
 \text{CFG} &= (\text{Node}, \rightarrow) \\
 n \in \text{Node} &= N_{cmd} \quad (\text{statement node}) \\
 &\triangleq N_{cond}(e) \quad (\text{condition node}) \\
 &\triangleq N_{skip} \quad (\text{skip node}) \\
 (\rightarrow) &= \text{Node} \times \text{Node} \quad (\text{normal edge}) \\
 (\xrightarrow{b}) &\mid \text{Node} \times \text{Node} \quad (\text{cond.edge}) \\
 &\quad (b \in \{\text{true}, \text{false}\}) \\
 (\rightsquigarrow) &\mid \text{Node} \times \text{Node} \quad (\text{handleredge}) \\
 cmd &\rightarrow le := e \\
 le &\rightarrow x \mid le.x \mid le[e] \\
 e &\rightarrow le \mid const \mid le(e^*) \mid e \text{ bop } e \mid uop e \mid cop e \mid sql
 \end{aligned}$$

CFG 언어의 정의는 그림 4와 같다. 언어를 이루는 부품이 엄청나게 간단해 졌음을 확인할 수 있다. 우선 CFG는 그래프이기 때문에 노드와 엣지의 집합으로 정의된다. 노드의 종류에는 할당 노드( $n : \text{cmd}$ ), 조건 노드( $N_{cond}(e)$ ), 그리고 스kip 노드( $N_{skip}$ )가 있다. 또한, 엣지의 종류는 일반 엣지( $\rightarrow$ ), 조건 엣지( $\xrightarrow{b}$ ), 핸들러 엣지( $\rightsquigarrow$ )의 3가지가 있다. 명령(cmd)에는 오직 할당문만 있다. 함수 호출, 예외 처리와, 루프 등과 같은 모든 제어 흐름이 할당문과 3가지 노드와 엣지로 전부 녹아들었음을 확인할 수 있다.

## 제 4 절 실제 실행 의미

프로그램  $P$ 의 실행의미는  $\llbracket P \rrbracket$ 은 프로그램이 실행 중에 만들 수 있는 모든 기계상태( $2^{Trace}$ )의 부분 집합이고, 다음 함수  $F \in 2^{Trace} \rightarrow 2^{Trace}$ 의 고정점으로 정의된다.( $T_0$ 는 초기의 기계상태 집합)

$$fix(F \stackrel{\text{let}}{=} \lambda T. T_0 \cup Next T)$$

### 4.1 실제 도메인

그림 5은 PL/SQL 핵심 언어 실행 의미 정의에 사용된 도메인이다. 각 도메인 객체의 역할은 다음과 같다.

- $Mem$  : 전역 메모리. 식별자( $Id$ )와 값( $Val$ )을 매핑하는 테이블이다.
- $Val$  : 스칼라 값, 테이블, 커서를 원소로 할 수 있는 값이다.

- *Taint* : 오염 기록을 관리하는 리스트. 어떤 노드에서 어떤 식별자로 인해 오염이 발생 했는지 기록한다.
- *Table* : DB 테이블을 행 번호에 매핑된 레코드로 모델링한 값이다.

## 4.2 실제 의미 구조

### 4.2.1 실제 실행 상태 전이 규칙

그림 6은 실제 기계 상태가 전이되는 규칙을 정의한다. 대부분의 명령이 다른 언어에도 존재하는 상식적인 구문이며 실행 의미도 특별하지 않다. 단, 오염 전파 규칙은 2장에서 정의한 변경 영향 전파 규칙을 따른다.

그림 6의 상태 전이 규칙에 사용된 셋팅, 표기법, 계산 함수는 다음과 같다.

### 4.2.2 셋팅

다음 함수가 제공된다.

- $R \in Node \xrightarrow{\text{fin}} Node \cup \{\perp\}$  : 노드  $n$ 이 조건문에 의해 생성되었을 경우(THEN절 또는 ELSE 절),  $n$ 과 그 노드를 만든 조건식 노드  $n_c$ 를 매핑한다.
- $P \in ProcId \xrightarrow{\text{fin}} (VarId \times \{\text{in}, \text{inout}\})^*$  : 프로시저  $Id$ 에 정의된 파라미터 정보를 리턴 한다.
- $\exists(T, n)$  : 테인트 정보  $T$ 에 노드  $n$ 이 포함되었는지 검사한다.

특별한 표기법은 다음과 같다.

- $\hat{m}\{x_i \mapsto v_i\} (1 \leq i \leq n)$  : 모든  $i$ 에 대한  $x_i \mapsto v_i$  쌍들을 업데이트 한다.

- $T \cup_b T' = \begin{cases} T \cup T' & (b = \text{true}) \\ T & (\text{otherwise}) \end{cases} : b가 오염되었을 경우만 T와 T'을 합친다.$

그림 5: 실제 실행 의미 도메인

$$\begin{aligned}
 Trace &= State^* \\
 \sigma \in &State &= Node \times Mem \times Taint \\
 m \in &Mem &= Loc \xrightarrow{\text{fin}} Val \\
 v \in &Val &= (ScalarVal + Table + Cursor) \\
 &ScalarVal &= \{\mathbb{Z} + String + \mathbb{B} + \{NULL\}\} \times \mathbb{T} \\
 &\mathbb{T} &= \{true, false\} &(\text{Taint여부}) \\
 &Cursor &= Table \times \mathbb{Z}_+ \\
 v_t \in &Table &= RowId \xrightarrow{\text{fin}} Record \\
 v_r \in &Record &= ColId \xrightarrow{\text{fin}} ScalarVal \\
 &Loc &= Id \\
 &ColId, Id &= String \\
 i \in &RowId &= \mathbb{Z}_+ \\
 T \in &Taint &= Node \xrightarrow{\text{fin}} \mathbb{T}
 \end{aligned}$$

### 4.2.3 실제 값 계산 함수

$$\boxed{\mathcal{V} \in (Mem \times Exp) \rightarrow (Val \times \mathbb{T})} \quad (\text{표현식에서 값 계산})$$

$$\begin{aligned} \mathcal{V}(m, const) &= \langle const, false \rangle \\ \frac{\mathcal{L}(m, le) = \langle x, b_1 \rangle \quad m(x) = \langle v, b_2 \rangle}{\mathcal{V}(m, le) = \langle v, b_1 \vee b_2 \rangle} \\ \frac{\mathcal{V}(m, e_i) = \langle v_i, b_i \rangle}{\mathcal{V}(m, e_1 \ bop \ e_2) = \langle v_1 \ bop \ v_2, b_1 \vee b_2 \rangle} \\ \frac{\mathcal{V}(m, e) = \langle v, b \rangle}{\mathcal{V}(m, uop \ e) = \langle uop \ v, b \rangle} \end{aligned}$$

$$\boxed{\mathcal{L} \in (Mem \times Exp) \rightarrow (Loc \times \mathbb{T})} \quad (\text{L-표현식에서 주소값 계산})$$

$$\begin{aligned} \frac{m(x) = \langle v, b \rangle}{\mathcal{L}(m, x) = \langle x, b \rangle} \\ \frac{\mathcal{L}(m, le) = \langle y, b_1 \rangle \quad m(y.x) = \langle v_2, b_2 \rangle}{\mathcal{L}(m, le.x) = \langle y.x, b_1 \vee b_2 \rangle} \\ \frac{\mathcal{L}(m, le) = \langle x, b_1 \rangle \quad \mathcal{V}(m, e) = \langle v, b_2 \rangle}{\mathcal{L}(m, le[e]) = \langle x.v, b_1 \vee b_2 \rangle} \end{aligned}$$

그림 6: 실제 실행 의미의 상태 전이 규칙

$$\begin{aligned} \boxed{next : \langle n, m, T \rangle \rightarrow \langle n', m', T' \rangle} \\ \frac{\mathcal{V}(m, e) = \langle v, b_1 \rangle \quad \mathcal{L}(m, le) = \langle x, b_2 \rangle \quad T(R(n)) = b_r \quad n \rightarrow n'}{\langle n : le := e, m, T \rangle \rightarrow \langle n', m \{x \mapsto (v, b_1)\}, T \{n \mapsto b_1 \vee b_2 \vee b_r\} \rangle} \\ \frac{\mathcal{S}(m, e_{sql}) = v_t \quad \mathcal{L}(m, le) = \langle x, b \rangle \quad T(R(n)) = b_r \quad n \rightarrow n'}{\langle n : le := e_{sql}, m, T \rangle \rightarrow \langle n', m \{x \mapsto v_t\}, m, T \{b \vee b_r \mapsto\} \rangle} \\ \frac{\mathcal{V}(m, e) = \langle v_c, b \rangle \quad T(R(n)) = b_r \quad n \xrightarrow{v_c} n'}{\langle n : N_{cond}(e), m, T \rangle \rightarrow \langle n', m, T \{n \mapsto b \vee b_r\} \rangle} \quad (\cdot \text{는 더미 주소}) \\ \frac{\mathcal{S}(m, e_{sql}) = v_t \quad tainted(v_t) = b \quad n \rightarrow n'}{\langle n : x := e_{sql}, m, T \rangle \rightarrow \langle n', m \{x \mapsto \langle v_t, 1 \rangle\}, T \{n \mapsto b\} \rangle} \\ \frac{m(x_c) = \langle v_t, k \rangle \quad v_t(k)(col_i) = \langle v_i, b_i \rangle \quad T' = T \{n \mapsto b_i\} \quad n \rightarrow n'}{\langle n : x_r := x_c, m, T \rangle \rightarrow \langle n', m \{(x_r.col_i \mapsto \langle v_i, b_i \rangle, x_c \mapsto \langle v_t, k+1 \rangle), T'\} \rangle} \\ \frac{\mathcal{V}(m, e_2) = \langle 0, b \rangle \quad n \xrightarrow{\text{OTHERS}} n'}{\langle n : le := e_1/e_2, m, T \rangle \Rightarrow \langle n', m, T \rangle} \\ \frac{\mathcal{V}(m, e) = \langle v, b \rangle \quad v = \langle v_t, i \rangle \in Cursor \quad i > |\text{dom}(v_t)| \quad n \xrightarrow{\text{NOT FOUND}} n'}{\langle n : le := e, m, T \rangle \Rightarrow \langle n', m, T \rangle} \\ \frac{n \rightarrow n'}{\langle N_{skip}, m, T \rangle \rightarrow \langle n', m, T \rangle} \end{aligned}$$

$$\boxed{\mathcal{R} \in (Mem \times Record \times Exp) \rightarrow ScalarVal} \quad (\text{레코드, 메모리에서 표현식 계산})$$

$$\begin{aligned} & \overline{\mathcal{R}(m, v_{row}, const) = \langle const, false \rangle} \\ & \frac{\mathcal{L}(m, le) = \langle x, b_1 \rangle \quad m(x) = \langle v, b_2 \rangle}{\mathcal{R}(m, v_{row}, le) = \langle v, b_1 \vee b_2 \rangle} \quad (x \in \text{dom}(M)) \\ & \frac{\mathcal{L}(m, le) = \langle x, b_1 \rangle \quad v_{row}(x) = \langle v, b_2 \rangle}{\mathcal{R}(m, v_{row}, le) = \langle v, b_1 \vee b_2 \rangle} \quad (\text{otherwise}) \\ & \frac{\mathcal{R}(m, v_{row}, e_i) = \langle v_i, b_i \rangle}{\mathcal{R}(m, v_{row}, e_1 \text{ bop } e_2) = \langle v_1 \text{ bop } v_2, b_1 \vee b_2 \rangle} \\ & \frac{\mathcal{R}(m, v_{row}, e) = \langle v, b \rangle}{\mathcal{R}(m, v_{row}, uop e) = \langle uop v, b \rangle} \end{aligned}$$

$$\boxed{\mathcal{S} \in (Mem \times Exp) \rightarrow Table} \quad (\text{SQL 표현식에서 테이블 값 계산})$$

$$\begin{aligned} & \frac{\mathcal{R}(m, v_t(i), e_{cond}) = \langle v_i, b_i \rangle \quad v_t(i)(j) = \langle v_{ij}, b_{ij} \rangle \quad b_{cond} = \vee b_i}{filter(v_t, e_{cond}) = \{i \mapsto v_t(i)\{j \mapsto (v_{ij}, b_{ij} \vee b_{cond})\}\}_{v_i=true}} \\ & \frac{\mathcal{R}(m, v_t(i), e_t) = \langle v_i, b_i \rangle}{trans(v_t, \langle e_t, id \rangle) = \{i \mapsto \{id \mapsto \langle v_i, b_i \rangle\}\}} \\ & \frac{m(x) = v_t}{\mathcal{S}(m, x) = v_t} \\ & \frac{\mathcal{V}(m, e_{table}) = v_t \quad renum(filter(v_t, e_{cond})) = v'_t}{\mathcal{S}(m, (\text{SELECT } \langle e_t, id_c \rangle e_{table} e_{cond})) = trans(v'_{table}, \langle e_t, id_c \rangle)} \\ & \frac{\mathcal{S}(m, t) = v_t \quad renum(filter(v_t, not e_{cond})) = v'_t}{\mathcal{S}(m, (\text{DELETE } t e_c)) = v'_t} \\ & \frac{\mathcal{S}(m, t) = v_t \quad \mathcal{V}(m, e) = (v, b) \quad r = newrow(v_t, id, \langle v, b \rangle)}{\mathcal{S}(m, (\text{INSERT } t \langle id, e \rangle)) = v_t\{(|\text{dom}(v_t)| + 1) \mapsto r\}} \\ & \frac{\mathcal{S}(m, t) = v_t \quad \mathcal{V}(m, e) = \langle v, b \rangle \quad RV(i) = \langle v_i, b_i \rangle \quad v'_t = v_t\{i \mapsto \{j \mapsto \langle v_{ij}, b_{ij} \vee b_i \rangle\}\}}{\mathcal{S}(m, (\text{UPDATE } t (id, e) e_{cond})) = v'_t\{i \mapsto setrow(v'_t(n), id, \langle v, b'_{ij} \vee b \rangle)\}_{v_i=true}} \\ & (RV(i) = \mathcal{R}(m, v_t(i), e_{cond}) \quad v_t(i)(j) = \langle v_{ij}, b_{ij} \rangle \quad v'_t(i)(j) = \langle v'_{ij}, b'_{ij} \rangle) \\ & \frac{\mathcal{S}(m, e_1) = v_t \quad Seval(e_2) = v'_t}{\mathcal{S}(m, e_1 \text{ s o p } e_2) = (v_t \text{ s o p } v'_t)} \end{aligned}$$

$filter(v_t, e_{cond}) = v_t$ 의 Row 중  $e_{cond}$ 를 만족하는 것만을 모은 새로운 Table

$renum(v_t) = v_t$ 의 행번호를 1부터 새로이 붙인 Table

$trans(v_t, \langle e_t, id \rangle) = v_t$ 을  $e_t$ 에 맞게 재구성한 Col에  $id$ 라는 이름을 붙여 만든 Table

$newrow(v_t, id, v) = v_t$ 의 ColId 중  $id$ 인 것에는  $v$ 를, 나머지에는  $\perp$ 을 갖고 있는 Row

$setrow(v_{row}, id, v) = v_{row}$ 의 ColId 중  $id$ 에 해당하는 값을  $v$ 로 바꾼 Row

## 제 5 절    요약 실행 의미

프로그램  $P$ 의 요약해석 결과  $\llbracket \hat{P} \rrbracket$ 는 모든 기계상태들의 모음을 몇 개의 조각으로 분할해서 요약하는 것이다.

$$\llbracket \hat{P} \rrbracket \in \Delta \rightarrow \hat{\text{State}}$$

이 분석기는 분석 대상 언어인 CFG에 맞게 노드를 분할 기준으로 삼는다.

### 5.1    요약 도메인

그림 7: 요약 실행 의미 도메인

$$\begin{aligned}
 \hat{\text{Trace}} &= \Delta \rightarrow \hat{\text{State}} \\
 \hat{\sigma} \in \hat{\text{State}} &= \text{Node} \times \hat{\text{Mem}} \times \hat{\text{Taint}} \\
 \Delta = \text{Node} & \\
 \hat{m} \in \hat{\text{Mem}} &= \hat{\text{Loc}} \xrightarrow{\text{fin}} \hat{\text{Val}} \\
 \hat{l} \in \hat{\text{Loc}} &= \text{Id} \\
 \hat{v} \in \hat{\text{Val}} &= (\hat{\text{ScalarVal}} \times \hat{\text{Table}} \times \hat{\text{Cursor}}) \\
 \hat{v}_s \in \hat{\text{ScalarVal}} &= \hat{\mathbb{T}} \times \hat{\mathbb{Z}} \\
 \hat{\mathbb{T}} &= \{\perp, \top\} \\
 \hat{z} \in \hat{\mathbb{Z}} &= \mathbb{Z}_\perp \cup \{\top\} \quad (\text{올려붙인 } \mathbb{Z}) \\
 \hat{v}_t \in \hat{\text{Table}} &= \text{ColId} \xrightarrow{\text{fin}} \hat{\text{ScalarVal}} \text{ (모든 행 뭉침)} \\
 \text{ColId} &= \text{Id} \\
 \hat{\text{Cursor}} &= \hat{\text{Table}} \quad (\text{커서 인덱스 제거}) \\
 \hat{T} \in \hat{\text{Taint}} &= \text{Node} \xrightarrow{\text{fin}} \hat{\mathbb{T}}
 \end{aligned}$$

요약 실행 의미의 도메인은 그림 5.1과 같다.

- $\hat{\text{ScalarVal}}$  : 정수일 때만 올려붙인 도메인으로 상수를 보존하고 나머지는  $\top$ 으로 보낸다.  
오염 정보  $\mathbb{T}$ 는  $\hat{\mathbb{T}}$ 로 원래 모양대로 요약한다.
- $\hat{\text{Table}}$  : 모든 행을 하나로 뭉쳐 컬럼에 매핑된 값으로 요약한다.
- $\hat{\text{Cursor}}$  :  $\hat{\text{Table}}$ 과 똑같이 요약된다. 한 행으로 뭉쳐졌으므로 행번호 인덱스가 불필요하다.
- $\hat{\text{Taint}}$  : 오염 노드와 관련된 주소를 모으지 않고, 단순히 노드의 오염 여부만 요약한다.  
분석 후에 오염된 노드를 시간순으로 정렬하면 어떤 주소가 오염되었는지 쉽게 확인할 수 있기 때문이다.

이 도메인은 넓히기(Widening) 없이도 유한시간에 분석이 끝난다. 정수 도메인은 올려붙인 도메인으로 높이가 3으로 유한하고, 실수, 문자열 등 다른 스칼라 값들을  $\top$ 으로 보내기 때문이다. 다른 도메인 객체들도 프로그램에 등장하는 객체들을 조립하거나 면집합으로 구성되므로 프로그램 크기에 의해 유한함이 보장된다. 모든 도메인 객체의 높이가 유한하므로 넓히기 없이 분석이 고정점에 도달한다.

### 5.2    요약 의미 구조

요약 실행의 기계 상태 전이 규칙은 그림 8 같다. 대부분 실제 의미에서 자연스럽게 유도된다. 핵심 규칙을 정리하면 다음과 같다.

- 할당문에서 L-표현식 주소가 오염되었거나 값이 오염될 경우 오염이 전파된다.
- 할당문을 자식으로 갖는 조건 노드가 오염되었을 경우, 오염이 전파된다.
- 조건 노드는 조건식이 오염되었거나, 조건 노드를 자식으로 갖는 상위 조건 노드가 오염되었을 경우, 오염이 전파된다.

### 5.2.1 요약 실행 상태 전이 규칙

그림 8: 요약 실행의 상태 전이 규칙

$$\begin{array}{c}
 \frac{\hat{\mathcal{V}}(\hat{m}, e) = \langle \hat{b}_1, \hat{z}, \hat{t} \rangle \quad \hat{T}(R(n)) = \hat{b}_r \quad n \rightarrow n_1}{\hat{\mathcal{L}}(\hat{m}, le) = \langle x, \hat{b}_2 \rangle \quad \hat{b}' = \hat{b}_1 \sqcup \hat{b}_2 \sqcup \hat{b}_r \quad n \xrightarrow{*} N_h} \\
 \frac{\langle n : le := e, \hat{m}, \hat{T} \rangle \xrightarrow{*} \{ \langle n', \hat{m} \{ x \mapsto \langle \hat{b}', \hat{z}, \hat{t} \rangle \}, \hat{T} \{ n \mapsto \hat{b}' \} \} \mid n' \in (\{n_1\} \cup N_h) \}}{} \\
 \\ 
 \frac{\hat{\mathcal{S}}(\hat{m}, e_{sql}) = \langle \hat{b}_1, \hat{z}, \hat{t} \rangle \quad \hat{T}(R(n)) = \hat{b}_r \quad n \rightarrow n_1}{\hat{\mathcal{L}}(\hat{m}, le) = \langle x, \hat{b}_2 \rangle \quad \hat{b}' = \hat{b}_1 \sqcup \hat{b}_2 \sqcup \hat{b}_r \quad n \xrightarrow{*} N_h} \\
 \frac{\langle n : le := e_{sql}, \hat{m}, \hat{T} \rangle \xrightarrow{*} \{ \langle n', \hat{m} \{ x \mapsto \langle \hat{b}', \hat{z}, \hat{t} \rangle \}, \hat{m}, \hat{T} \{ n \mapsto \hat{b}' \} \} \mid n' \in (\{n_1\} \cup N_h) \}}{} \\
 \\ 
 \frac{\hat{\mathcal{V}}(\hat{m}, e) = \langle \hat{v}_c, \hat{b} \rangle \quad \hat{T}(R(n)) = \hat{b}_r \quad n \rightsquigarrow N'}{\langle n : N_{cond}(e), \hat{m}, \hat{T} \rangle \rightarrow \langle n', \hat{m}, \hat{T} \{ n \mapsto \hat{b} \sqcup \hat{b}_r \} \rangle \mid n' \in N'} \\
 \\ 
 \frac{\hat{\mathcal{V}}(\hat{m}, e) = \langle \hat{b}, \hat{z}, \hat{t} \rangle \quad \hat{t} = \left\{ \begin{array}{l} c_1 \mapsto v_1, \\ \vdots \\ c_n \mapsto v_n \end{array} \right\} \quad \hat{\mathcal{L}}(\hat{m}, le_1) = \langle x_1, \hat{b}_1 \rangle \quad \hat{T}(R(n)) = \hat{b}_r}{\hat{\mathcal{L}}(\hat{m}, le_n) = \langle x_n, \hat{b}_n \rangle \quad \hat{b}' = \hat{b} \sqcup \hat{b}_r \quad 1 \leq i \leq n} \\
 \frac{\langle n : (le_1, \dots, le_n) := e, \hat{m}, \hat{T} \rangle \xrightarrow{*} \{ \langle n', \hat{m} \{ x_i \mapsto v_i \sqcup \langle \hat{b}', \perp, \perp \rangle \}, \hat{T} \{ n \mapsto \hat{b}' \} \} \mid n' \in (\{n_1\} \cup N_h) \}}{} \\
 \\ 
 \frac{n \rightarrow n'}{\langle N_{skip}, \hat{m}, \hat{T} \rangle \xrightarrow{*} \langle n', \hat{m}, \hat{T} \rangle}
 \end{array}$$

### 5.2.2 요약 값 계산 함수

$$\hat{\mathcal{V}} \in (\hat{Mem} \times \hat{Exp}) \rightarrow \hat{Val}$$

$$\hat{\mathcal{V}}(\hat{m}, const) = \alpha T_{vs}(\{const\})$$

$$\frac{\hat{\mathcal{L}}(\hat{m}, le) = (x, \hat{b}) \quad \hat{m}(x) = \langle \hat{b}', \hat{z}', \hat{t}' \rangle}{\hat{\mathcal{V}}(\hat{m}, le) = \hat{v} \sqcup \langle \hat{b} \sqcup \hat{b}', \hat{z}', \hat{t}' \rangle}$$

$$\frac{\hat{\mathcal{V}}(\hat{m}, e_1) = \hat{v}_1 \quad \hat{\mathcal{V}}(\hat{m}, e_2) = \hat{v}_2}{\hat{\mathcal{V}}(\hat{m}, e_1 \ bop \ e_2) = \hat{v}_1 \ bop \hat{v}_2}$$

$$\frac{\hat{\mathcal{V}}(\hat{m}, e) = \hat{v}}{\hat{\mathcal{V}}(\hat{m}, uop \ e) = uop(\hat{v})}$$

$$\hat{\mathcal{L}} \in (\hat{Mem} \times \hat{Exp}) \rightarrow (\hat{Loc} \times \hat{\mathbb{T}})$$

$$\frac{\hat{m}(x) = \hat{v}}{\hat{\mathcal{L}}(\hat{m}, x) = (x, \perp)}$$

$$\frac{\hat{\mathcal{L}}(\hat{m}, le) = (y, \hat{b}) \quad \hat{m}(y.x) = \langle \hat{b}', \hat{z}', \hat{t}' \rangle}{\hat{\mathcal{L}}(\hat{m}, le.x) = (y.x, \hat{b} \sqcup \hat{b}')} \\ \frac{\hat{\mathcal{L}}(\hat{m}, le) = (x, \hat{b}) \quad \hat{\mathcal{V}}(\hat{m}, e) = \langle \hat{b}', \hat{z}', \hat{t}' \rangle}{\hat{\mathcal{L}}(\hat{m}, le[e]) = (x[v], (\hat{b} \sqcup \hat{b}'))}$$

$$\boxed{\hat{\mathcal{R}} \in (\hat{Mem} \times \hat{Table} \times \hat{Exp}) \rightarrow \hat{Val}}$$

$$\overline{\hat{\mathcal{R}}(\hat{m}, \hat{t}, const) = \alpha T_{vs}(\{const\})}$$

$$\frac{\hat{\mathcal{L}}(\hat{m}, le) = (x, \hat{b}) \quad \hat{t}(x) = \langle \hat{b}', \hat{z}', \perp \rangle \quad (x \in \text{dom}(\hat{t}))}{\hat{\mathcal{R}}(\hat{m}, \hat{t}, le) = \langle \hat{b} \sqcup \hat{b}', \hat{z}', \perp \rangle} \\ \frac{\hat{\mathcal{L}}(m, le) = (x, \hat{b}) \quad \hat{m}(x) = \langle \hat{b}', \hat{z}', \hat{t}' \rangle \quad (\text{otherwise})}{\hat{\mathcal{R}}(\hat{m}, \hat{t}, le) = \langle \hat{b} \sqcup \hat{b}', \hat{z}', \hat{t}' \rangle} \\ \frac{\hat{\mathcal{R}}(\hat{m}, \hat{t}, e_1) = \hat{v}_1 \quad \hat{\mathcal{R}}(\hat{m}, \hat{t}, e_2) = \hat{v}_2}{\hat{\mathcal{R}}(\hat{m}, \hat{t}, e_1 \text{ bop } e_2) = \hat{v}_1 \text{ bop } \hat{v}_2} \\ \frac{\hat{\mathcal{R}}(\hat{m}, \hat{t}, e) = \hat{v}}{\hat{\mathcal{R}}(\hat{m}, \hat{t}, uop e) = uop(\hat{v})}$$

$$\boxed{\hat{\mathcal{S}} \in (\hat{Mem} \times \hat{Exp}) \rightarrow \hat{Table}}$$

$$\frac{\hat{m}(x) = \hat{v}}{\hat{\mathcal{S}}(\hat{m}, x) = \hat{v}} \\ \frac{\hat{\mathcal{R}}(\hat{m}, \hat{t}, e_c) = \langle \hat{b}_c, \perp, \perp \rangle}{filter(\hat{m}, \hat{t}, e_c) = \{c \mapsto \langle \hat{b}_c \sqcup \hat{b}, \hat{z}, \perp \rangle \mid c \in \text{dom}(\hat{t}), \hat{t}(c) = \langle hb, \hat{z}, \perp \rangle\}} \\ \hat{\mathcal{R}}(\hat{m}, \hat{t}, e_1) = \hat{v}_1 \\ \vdots \\ \hat{\mathcal{R}}(\hat{m}, \hat{t}, e_n) = \hat{v}_n \\ \overline{trans(\hat{m}, \hat{t}, (\langle e_1, id_1 \rangle, \dots, \langle e_n, id_n \rangle)) = \{id_i \mapsto \hat{v}_i \mid 1 \leq i \leq n\}}$$

$$\frac{\hat{\mathcal{V}}(\hat{m}, e_t) = \langle \hat{b}, \_, \hat{t} \rangle \quad filter(\hat{m}, \hat{t}, e_c) = \hat{t}' \quad trans(\hat{m}, \hat{t}', (\langle e_i, id_i \rangle)^*) = \hat{t}''}{\hat{\mathcal{S}}(\hat{m}, \text{SELECT } (\langle e_i, id_i \rangle)^* e_t e_c) = \hat{t}''}$$

$$\frac{\hat{\mathcal{S}}(\hat{m}, e_t) = \hat{t} \quad filter(\hat{m}, \hat{t}, \text{not } e_c) = \hat{t}'}{\hat{\mathcal{S}}(\hat{m}, \text{DELETE } e_t e_c) = \hat{t}'}$$

$$\frac{\hat{\mathcal{S}}(\hat{m}, e_t) = \hat{t} \quad \{c \mapsto \hat{\mathcal{R}}(\hat{m}, \hat{t}, e) \mid c \in \text{dom}(\hat{t}), e = (e_i \text{ if } c = id_i \text{ else } \text{null})\} = \hat{t}'}{\hat{\mathcal{S}}(\hat{m}, \text{INSERT } e_t (\langle id_i, e_i \rangle)^*) = \hat{t} \sqcup \hat{t}'}$$

$$\frac{\hat{\mathcal{S}}(\hat{m}, e_t) = \hat{t} \quad filter(\hat{m}, \hat{t}, e_c) = \hat{t}' \quad \{c \mapsto v \mid c \in \text{dom}(\hat{t}'), v = \hat{\mathcal{R}}(\hat{m}, \hat{t}', e_i) \text{ if } c = id_i \text{ else } \hat{t}'(c)\}}{\hat{\mathcal{S}}(\hat{m}, (\text{UPDATE } e_t ((id_i, e_i))^* e_c)) = \hat{t} \sqcup \hat{t}'}$$

$$\frac{\hat{\mathcal{S}}(\hat{m}, e_1) = \hat{t}_1 \quad Seval(\hat{m}, e_2) = \hat{t}_2}{\hat{\mathcal{S}}(\hat{m}, e_1 \text{ sop } e_2) = \hat{t}_1 \text{ sop } \hat{t}_2}$$

## 제 6 절 변경 영향 정보 가공

요약된 오염 정보  $\hat{T}$ 에는 노드들의 오염 여부만 있기 때문에 어떤 과정으로 변경 영향이 전파되었는지 확인하기 위해 정보 가공이 필요하다. 다음 알고리즘으로 변경 영향 그래프를 생성할 수 있다.

1. 오염된 노드를 목록으로 모은다.
2. 오염된 노드를 실행 역순으로 정렬한다.
3. 가장 마지막 실행된 오염 노드가 첫번째 노드가 되는데, 이를 현재 노드로 할당한다.
4. 현재 노드에서 오염된 값의 주소를 모은다.
5. 그 다음 노드에서 오염되지 않은 값의 주소를 모은다.
6. 두 주소의 교집합으로 현재 노드로 전이하면서 오염된 주소를 찾는다.
7. 이들간에 간선을 그어 의존 관계를 나타낸다.
8. 그 다음 노드를 현재 노드로 할당하고 더이상 다음 노드가 없을 때까지 같은 작업을 반복한다.

## 제 7 절 분석기 구현

본 연구에서는 개념 검증을 위해 설계 명세를 준수하는 가장 간단한 분석기를 개발하였다. 구현된  $\hat{V}$  함수는 정수의 사칙/단항 연산만을 계산 가능하고, 정수의 등식과 부등식만을 조건문으로 사용 가능하며  $\hat{S}$  함수에는 여러 테이블의 조인이나, 서브 쿼리를 구현하지 않았다. 요약 해석기 구현에는 실용적으로 사용되는 워크리스트 알고리즘을 사용했으며, 빠른 개념 검증을 목적으로 했으므로 최근 연구에서 고안된 스파스 분석, 선별적 문맥 구분 분석 등의 최적화 기술들은 적용하지 않았다. 단, 파서는 PL/SQL 핵심 문법을 모두 올바르게 파싱한다.

구현을 위해 선택한 언어는 스칼라(Scala)인데, 윈도와 유닉스 환경에서 주로 개발하는 반도체 현장에 적합하고 함수형 언어로 개발되오던 기존 코드를 참고하여 개발할 수 있기 때문이다. 스칼라는 각광받는 함수형 언어로서, 자바 가상 머신 위에서 동작하기 때문에 기존에 작성된 풍부한 자바 라이브러리를 그대로 쓸 수 있다. 뿐만 아니라 자바 가상머신이 포팅된 모든 환경에서 사용할 수 있다. F#, OCaml, Haskell 같은 다른 함수형 언어는 윈도나 유닉스 플랫폼에서 모두 사용하는데 제약이 있다.

### 7.1 요약 도메인 구현

기존에 연구된 OCaml용 요약 도메인 생성 라이브러리를 스칼라로 옮겨 요약 도메인을 구현했다. 같은 함수형 언어였기 때문에 적은 노력으로 라이브러리를 옮길 수 있었다. 그럼 9의 예제는 앞서 디자인했던 요약 도메인을 스칼라 코드로 자연스럽게 옮긴 사례이다. 자바 같은 객체지향 언어의 표현력으로는 이렇게 간단히 도메인을 구현하는 것이 불가능하다.

그림 9: Scala를 이용한 요약 도메인 구현 예제

```

1  /* Primitive Sets */
2  class IdSet    extends PrimitiveSet[Id]
3  class NameSet  extends PrimitiveSet[Name]
4  class NodeSet  extends PrimitiveSet[Node]
5  class StrSet   extends PrimitiveSet[String]
6  class IntSet   extends PrimitiveSet[Long]
7
8  /* Abstract Domain Types */
9  class DInt extends FlatDomain[IntSet] {
10    def bop(f:(Long,Long) => Long, x:elt, y:elt): elt =
11      (x, y) match { ... }
12    def uop(f:Long => Long, x: elt): elt = x match { ... }
13  }
14  type DTF      = BinaryDomain
15  type DTable   = TableDomain[IdSet, DTF]
16  type DVal     = ProductDomain3[DTF, DInt, DTable]
17  type DMem     = FuncDomain[IdSet, DVal]
18  type DState   = ProductDomain[DMem, DTF]
19  type DTrace   = FuncDomain[NodeSet, DState]

```

## 7.2 요약 해석 알고리즘 구현

요약 해석기를 사용한 워크리스트 알고리즘의 의사 코드와 이 코드를 바탕으로 스칼라로 알고리즘을 옮긴 코드가 그림 10이다. 워크리스트 알고리즘은 매번 모든 노드를 훑으며  $\hat{next}$ 를 계산할 필요 없이, 이전 반복에서 변화된 것들만( $T[i] \sqsubseteq T'[i]$ ) 다시 계산한다. 의사 코드에서 실제 코드가 자연스럽게 유도됨을 확인할 수 있다.

그림 10: 워크 리스트 알고리즘 구현 예제

```

 $T, T' : \Delta \rightarrow \hat{State}$ ;
 $W : 2^\Delta$ ; (* worklist *)
begin
     $T := T' := \alpha(T_0)$ ;  $W := \Delta$ ;
    repeat
         $T' := T$ ;
         $T := \alpha(T_0) \sqcup ((\wp \sqcup) \circ \hat{\pi})(\bigcup_{i \in W} \hat{next}(T[i]))$ ;
         $W := \{i \in \Delta \mid T[i] \not\subseteq T'[i]\}$ ;
    until  $W = \{\}$ ; (* no more increase *)
    return  $T'$ ;
end

```

```

1 def worklist(trace: HTrace.elt, todos: List[Node], remains: List[Node] = Nil)
2   : HTrace.elt = { // Trace를 받아 Trace를 리턴
3   // 실제 도메인과 요약도메인의 파티션이 같으므로  $(\wp \sqcup) \circ \hat{\pi}$  는 무시
4   //  $T = trace$ ,  $W = todos$ 
5   todos match {
6     case Nil => trace // no more works
7     case n :: todos1 => //  $n = i$ 
8       val state = HTrace.image(trace, n) // state =  $T[i]$ 
9       val (state1, nexts) = hnext(i, trace, state) //  $\hat{next}(n, state)$ 
10      val trace1 = nexts.foldLeft(trace) { (trace, n) =>
11        HTrace.weakupdate(trace, i, state1)
12      } //  $\bigcup_{i \in W} \hat{next}(n, state)$ 
13      val (todos2, remains1) =
14        nextWorks(trace, state1, nexts, todos1, remains)
15      worklist(trace1, todos2, remains1)
16    }
17  }

```