# 오퍼레이셔널 의미에 기반한 일차 함수형 언어의 정적 분할

## (Static Slicing of First-Order Functional Language based on Operational Semantics)

안 준 선 † 한 태 숙 ††
(Joonseon Ahn) (Taisook Han)

**요 약** 정적 분할이란 가능한 모든 입력값을 고려할 때 프로그램 내의 한 지점에서 계산되는 값에 영향을 줄 수 있는 프로그램 부분들의 집합을 말한다. 본 논문에서는 일차 함수형 언어의 오퍼레이셔널 의미 정의에 기반하여 정적 분할을 정형적으로 정의하고 요약 해석을 통한 정적 분할의 분석 방법을 제시하였으며, 제시된 분석 방법이 건전함을 증명하였다.

***Abstract*** Static slice means a set of parts of a program that potentially affect the values computed at a slicing criterion considering all the possible input values. In this paper, we have formally defined static slice of a first-order functional language based on operational semantics. And, we have presented a sound method to analyze static slice using abstract interpretation.

## 1. 서 론

프로그램 분할(program slice/slicing)이란 기준이 되는 프로그램 내의 한 지점(slicing criterion)에서 계산되는 값에 영향을 주는 프로그램의 모든 부분들의 집합 또는 이러한 부분들을 찾아내는 분석을 말한다[1]. 프로그램 분할은 프로그램의 디버깅(debugging)시에 오류 검색의 범위를 줄이는데 사용되며, 이외에도 쓰이지 않는 코드의 제거(dead code elimination), 처리되지 않는 예외 상황의 검출(exception analysis), 프로그램의 병렬화(parallelization) 등을 위한 분석에 사용될 수 있다.

프로그램 분할은 그 분석 시점에 따라 동적 분할(dynamic slice/slicing)과 정적 분할(static slice/slicing)로 분류된다[2]. 동적 분할이란 어떤 특정한 입력 값을 사용한 프로그램 수행에 대하여, 분할 기준에서 생성된 값들에 영향을 주었던 모든 프로그램의 부분들을 찾아내는 것이다. 반대로, 정적 분할이란 가능한 모든 입력 값을 고려하여, 분할 기준에서 계산되는 값에 영향을 줄 가능성이 있는 모든 프로그램의 부분들을 컴파일 시간(compile time)에 찾아내는 것을 말한다. 따라서, 동적 분할은 수행시의 부담(overhead)을 줄이면서 완벽한(sound and complete) 분석을 수행하는 것이 중요한 문제가 되며, 정적 분할은 컴파일 시간에 완벽한 분석이 불가능하므로 분석 시간을 최소로 하면서 건전(sound)하고 의미가 있는 결과를 내는 것이 주된 관심사가 된다.

Weiser가 프로그램 분할의 개념을 처음으로 제시한 이후, 명령형 언어(imperative languages)에 대한 정적 및 동적 분할에 관한 많은 연구가 진행되었다[1,2,3,4,5]. 그리고 함수형 언어(functional languages)의 프로그램 분할에 관련해서는, Biswas가 집합 기반 분석(set-based analysis) 방법론을 사용한 동적 분할 방법을 제시한 바 있다[6]. 함수형 언어의 정적 분할에 관한 연구는 발표된 바가 없으나, 프로그램 내의 두 부분식(subexpression) 간의 정의-사용 관계(def-use relation)의 조사가 주된 관심이 된다는 점에서 관련된 연구로서 함수형 프로그램의 자료 흐름 분석(data-flow

· 본 연구는 첨단정보기술연구센터를 통하여 한국과학재단의 지원을 받았음.
† 비 회 원 : 한국과학기술원 전자전산학과 전산학전공
jsahn@pllab.kaist.ac.kr
†† 종신회원 : 한국과학기술원 전자전산학과 전산학전공 교수
han@cs.kaist.ac.kr
논문접수 : 2000년 1월 7일
심사완료 : 2000년 5월 3일

analysis)을 들 수 있다[7,8]. 그러나, 이러한 자료 흐름 분석에 관한 연구들은 정적 분할이 아닌 범용적인 방법론(framework)을 제시하며, 리커젼(recursion)을 쉽게 표현하기 위하여 디노테이셔널 의미론(denotational semantics)에 기반하기 때문에 실제적인 구현이 복잡한 단점을 가진다.

본 연구는 일차 함수형 언어의 정적 분할을 목적으로 한다. 이를 위하여, 입력 함수형 언어의 실제적인 의미(concrete semantics)를 오퍼레이셔널 의미론(operational semantics)으로 정의하고, 이에 기반하여 정적 분할의 정형적(formal)인 의미를 정의하였으며, 요약 해석(abstract interpretation)[9]에 기반한 정적 분할 알고리즘을 제시하였다. 본 연구는 다음과 같은 특징을 가진다.

- 일반적으로, 요약 해석을 통한 함수형 프로그램의 분석은 디노테이셔널 의미론을 사용한다[7,8]. 그런데, 디노테이셔널 의미론에서는 재귀 함수(recursive function)의 값을 fixed-point 함수를 사용하여 직접 정의하게 되므로, 이에 기반한 언어의 의미 정의는 직접적인 구현에 어려움이 따른다. 본 연구에서는, 오퍼레이셔널 의미론에 기반하여 대상 언어의 의미(semantics)와 정적 분할의 분석을 정의함으로써, 직접적인 구현이 용이하도록 하였다.
- 입력 언어의 요약(abstraction)된 수행을 위한 값 공간(value space)과 요약된 의미 정의를 설계함에 있어, 튜플(tuple)값과 자료 생성자(data constructor)값의 구조를 유지함으로써, 자료 흐름의 세밀한 분석이 가능하도록 하였다. 예를 들어 *(x,y) = (e1,e2)*와 같은 지정문이 수행되는 경우에 *e1*은 *x*에만 지정되므로 *y*에는 영향을 미치지 않는데, 본 연구의 요약 해석에서는 이러한 정보의 분석이 가능하다.
- 분석에 사용되는 요약된 값 공간의 크기를 값의 최대 중첩 깊이를 사용하여 조절 할 수 있도록 함으로써, 분석의 정밀도를 상황에 따라 쉽게 조절할 수 있다.
- 유한 시간에 종결(terminate)되는 정적 분할의 분석 알고리즘을 정형적으로 정의하였으며, 정의된 분석 알고리즘이 정적 분할의 정의에 비추어 건전한 결과를 생성함을 증명하였다.

2절에서는 입력 함수형 언어의 문법과 오퍼레이셔널 의미를 제시하고, 이에 기반하여 정적 분할을 정형적으로 정의한다. 3절에서는 함수형 언어의 요약된 오퍼레이셔널 의미를 설계하고, 요약된 의미에 의한 프로그램의 수행 결과가 실제적인 수행 결과를 건전하게 반영함을 증명한다. 4절에서는 요약된 의미에 기반하여 정적 분할의 분석 방법을 제시하고, 5장에서 결론으로 맺는다. 본 논문에서 제시된 모든 성질과 보조 정리 및 정리에 대한 증명은 관련된 다른 논문에 기술하였다[10].

## 2. 병렬화를 위한 재귀 함수

### 2.1 입력 언어

정적 분할 분석을 위한 대상 언어는 그림 1과 같은 일반적인 일차 함수형 언어의 형태를 가진다. 본 연구에서는, 정의와 증명을 간단하게 하기 위하여, 함수에는 자유 변수(free variable)가 없으며 상호 재귀 호출(mutual recursion)은 사용하지 않는 것으로 가정하였다. 그러나 이러한 제한은 별다른 어려움 없이 완화될 수 있다.

$$
\begin{array}{lll}
p & ::= & c \mid x{:}l \mid \_ \mid (p_1, p_2) \mid x\ p \\
e & ::= & c{:}l \mid x{:}l \mid (e_1, e_2){:}l \mid x\ e{:}l \mid e_1 + e_2{:}l \\
 & & \mid f\ e{:}l \mid \text{let } p = e_1 \text{ in } e_2 \text{ end}{:}l \\
fdef & ::= & f\ p_1 = e_1 \mid \ldots \mid p_n = e_n
\end{array}
$$

그림 1 입력 언어

그림 1에서 *p*는 함수의 지정문의 왼편이나 함수의 형식 인자(formal parameter)에 쓰이는 패턴(pattern)을 나타내며 *e*는 표현식(expression), *fdef*는 함수 정의를 나타낸다. 그리고, *c*는 상수 패턴 또는 상수를 나타내며 *x*는 변수 패턴 또는 변수, *f*는 함수 이름, *x*는 자료 생성자를 나타낸다. 정의된 언어의 프로그램은 함수 정의의 형태를 가지며, 함수 정의는 함수 이름과 형식 인자와 함수 몸체(function body)쌍의 리스트로 이루어진다. 형식 인자는 상수 패턴, 튜플 패턴, 자료 생성자 패턴, 임의 패턴(wild pattern) 등과 같은 패턴이며, 함수 몸체는 상수, 변수, 튜플식, 자료 생성자의 적용, 기본 함수(primitive function) 적용, 재귀 호출, let문과 같은 표현식으로 이루어진다. 또한 정적 분할을 정의하기 위하여, 함수 몸체 내의 모든 변수 패턴들과 부분식들에는 각각을 나타내는 레이블(label) *l*을 덧붙였다.

입력 언어의 오퍼레이셔널 의미 정의에서 사용되는 값(semantic values)들의 집합은 다음과 같이 정의된다.

$$
\begin{array}{lll}
v & \in & Val ::= \ c \mid (v_1, v_2) \mid x\ v \mid v_\perp \\
F & \in & Collection : Label \rightarrow 2^{Val} \\
\rho & \in & Env ::= Id \rightarrow (Val \cup Fcl)
\end{array}
$$

*Val*은 표현식들이 계산된 결과값들의 집합으로서, 상수값 *c*, 튜플값 $(v_1, v_2)$, 자료 생성자가 적용된 결과값 *x v*, 그리고 차단값(masked value) $v_\perp$으로 이루어진다.

차단값은 어떤 값의 오른쪽 밑에 $\perp$을 붙인 것으로서, 관심이 있는 특정 부분식으로부터 계산된 값임을 나타내기 위하여 사용된다. *Collection*은 어떤 표현식을 계산하여 결과가 생성되었을 경우에, 표현식 내의 모든 변수 패턴이나 부분식들에 대하여 각각 지정(assign)되었거나 계산되었던 모든 값들을 저장하는데 사용된다. 따라서 *Collection*은 변수 패턴이나 부분식을 지칭하는 프로그램 레이블들의 집합 *Label*의 원소에 대하여, 값들의 집합을 돌려주는 함수의 형태로 나타낸다. *Env*는 식별자(identifier)에 대한 환경으로서 변수 및 함수 이름에 대하여 해당하는 값이나 함수 정의를 저장하기 위하여 사용된다. 본 논문에서는 *Val*, *Collection*, *Env*의 원소에 대하여 각각 $v$, $F$, $\rho$를 사용하였다.

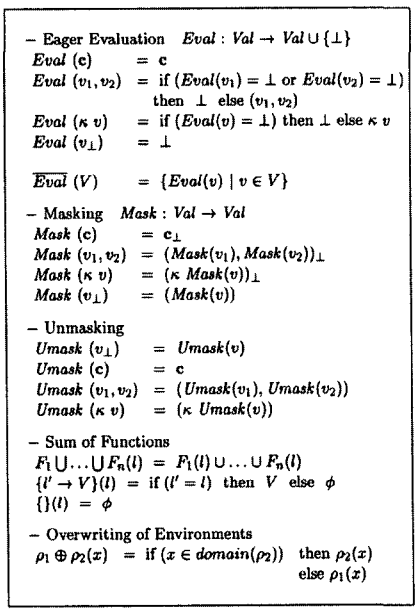


그림 2 의미 정의에서 사용되는 부가 함수들

그림 2는 언어의 오퍼레이셔널 의미론에서 사용되는 부가적인 함수들이다. *Eval*은 어떤 값이 차단값을 포함하는지 조사하는 함수로서, 차단값을 포함하고 있으면 $\perp$을 결과로 돌려준다. *Mask*는 주어진 값의 모든 부분에 $\perp$을 붙이는 함수이며 *Umask*는 주어진 값의 모든 $\perp$을 없애는 함수이다. $\cup$은 함수의 합을 나타내는데, 함수의 합은 주어진 인자에 각각의 함수들을 적용시킨 결과들의 합집합을 결과로 생성한다. $\oplus$는 새로운 변수 환경의 추가를 나타내며, 이 때 왼쪽 변수 환경에 오른

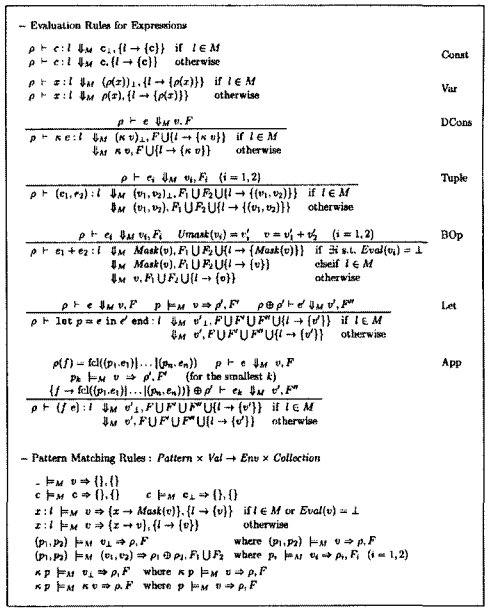


그림 3 입력 언어의 오퍼레이셔널 의미 정의

쪽 변수 환경이 덮어써지게 된다.

정의된 언어의 오퍼레이셔널 의미 정의는 그림 3과 같이 표현식의 계산 규칙과 패턴 매칭 규칙으로 이루어진다. 정적 분할을 정의하기 위하여 값이 차단되는 표현식의 집합인 $M$이 추가된 것을 제외하면, 제시된 의미 정의는 함수형 프로그램의 일반적인 오퍼레이셔널 의미 정의와 같은 형태를 가진다. 표현식의 계산 규칙은 다음과 같은 형태를 갖는다.

$$\rho \vdash e \Downarrow_M v, F$$

$\rho$는 변수에 대한 환경이며 $e$는 계산되는 표현식이다. $\Downarrow_M$은 계산 규칙이 적용되는 것을 나타내며, 결과로서 표현식 $e$의 값 $v$와 모든 부분식들에 대하여 수행 중에 계산되었던 모든 값들을 저장한 콜렉션(collection) $F$가 생성된다. 표현식의 계산 규칙은 표현식의 구조에 따라서 각각 정의되는데, 2단으로 이루어진 규칙에서 상단은 각각의 부분식이나 환경에 대한 조건을 나타내며 하단은 이러한 조건하에서 전체식이 계산된 결과를 나타낸다.

패턴 매칭 규칙은 let 표현식이나 함수 적용의 계산에서 어떤 패턴에 값이 지정(assign)되어 변수 환경과 콜렉션이 생성되는 방법을 나타낸다. 이때 변수 환경은

패턴내의 각각의 변수에 지정되는 값들을 저장하며, 콜렉션은 패턴 내의 변수 패턴의 레이블에 대하여 해당 변수에 지정되었던 모든 값들의 집합을 저장한다.

$M$은 레이블의 집합으로서, 프로그램의 수행 중에 이 레이블들이 가리키는 부분식의 결과값이나 변수 패턴에 매치되는 값들에는 $\perp$을 붙여준다. 그리고, $\perp$을 포함한 값을 사용한 계산의 결과는 언제나 $\perp$을 포함하도록 계산 규칙을 정의함으로써, $M$에 속한 부분식에서 계산된 값을 분할 기준에서 사용하는지를 검사할 수 있다.

원시 함수의 계산과 변수 패턴에 대한 패턴 매칭에서 사용된 *Mask*함수는 해당 계산이 부분적으로 수행될 수 없고 모든 필요한 값들이 있어야 수행이 일어남을 나타낸다. 예를 들어, $(x,y)=(e_1,e_2)$와 같은 지정문의 경우에 $x$와 $y$에 대한 지정은 따로 수행이 가능하다. 그러나, $x=(e_1,e_2)$와 같은 경우에는 오른쪽 표현식이 전부 계산되어야 지정이 가능하므로 결과적으로 $x$에 지정되는 $(e_1,e_2)$의 계산 결과인 $(v_1,v_2)$의 모든 부분은 $e_1$과 $e_2$의 영향을 모두 받는다고 할 수 있다. 따라서 $v_1$ 또는 $v_2$가 $\perp$을 포함하고 있다면 환경에 저장되는 x값의 모든 부분에는 $\perp$이 붙여지게 된다. 또, 원시 함수의 계산에 있어서도, 어떤 원시 함수 $\otimes$가 $1\otimes 2=(1,3)$과 같이 계산된다고 할 때 결과값이 생성되기 위해서는 $1\perp$ 와 $2$가 모두 필요하므로 $1_\perp\otimes 2$, $1\otimes 2_\perp$, $1_\perp\otimes 2_\perp$의 결과는 모두 $Mask((1,3))=(1_\perp,3_\perp)_\perp$이 된다.

다음은 x의 값을 1로 저장한 환경에서 표현식 let $y:l_1=x:l_2$ in $y:l_3$ end$:l_4$가 계산되는 예이다. (이때 편의상 $l_1$, $l_2$, $l_3$는 $M$에 포함되지 않는다고 가정하였다.)

$$\frac{\begin{array}{c}\{x\to 1\}\vdash x:l_2 \Downarrow_M 1,\ \{l_2\to\{1\}\}\\ y:l_1 \models 1 \Rightarrow \{y\to 1\},\ \{l_1\to\{1\}\}\\ \{x\to 1\}\oplus\{y\to 1\}\vdash y:l_3 \Downarrow_M 1,\ \{l_3\to\{1\}\}\end{array}}{\begin{array}{l}\{x\to 1\}\vdash \text{let } y:l_1=x:l_2 \text{ in } y:l_3 \text{ end}:l_4\\ \quad\Downarrow_M 1_\perp,\ \{l_1\to\{1\}\}\cup\{l_2\to\{1\}\}\cup\{l_3\to\{1\}\}\cup\{l_4\to\{1\}\} \quad \text{if } l_4\in M\\ \quad\Downarrow_M 1,\ \{l_1\to\{1\}\}\cup\{l_2\to\{1\}\}\cup\{l_3\to\{1\}\}\cup\{l_4\to\{1\}\} \quad \text{if } l_4\notin M\end{array}}$$

전체 let 표현식의 계산을 위하여 먼저 $y$에 지정되는 표현식 $x$의 값이 변수의 계산 규칙에 의하여 1로 계산되고, 이 값이 패턴 $y$와 매치되므로써 새로운 환경이 생성된다. 그리고, 이전의 환경과 새로운 환경이 합쳐진 환경 하에서 let 표현식의 결과 부분인 $y:l_3$가 계산되므로써 let 표현식의 값이 계산된다. 이 때 만약 계산되는 let 표현식의 레이블 $l_4$가 $M$에 포함되면 결과값에 $\perp$을 붙여 결과값이 $M$에 포함된 식에서 계산된 값임을 표시하게 된다. 그리고, 위의 계산 과정에서 각각의 부분식에서 계산되었던 값들과 변수에 매치되었던 값들이 콜렉션에 저장되어 결과 생성 시에 모두 합쳐짐으로써, 전체 계산에 대한 콜렉션을 생성하게 된다. 생성된 콜렉션은, 위 let문의 계산중에 부분식들인 $l_2$, $l_3$, $l_4$가 1로 계산되었고 변수 패턴 $l_1$에 1이 매치되었음을 기록한다.

### 2.2 정적 분할

이전 절에서 정의된 함수형 언어의 오퍼레이셔널 의미 정의를 기반으로 정적 분할을 다음과 같이 정형적으로 정의할 수 있다.

**정의1 정적 분할**

함수 정의 $f\ p_1=e_1 \mid \ldots \mid p_n=e_n$에 대하여, $e_1,\ldots,e_n$에 있는 모든 레이블의 집합을 $L_f$라고 할 때, 분할 기준 $l$에 대한 정적 분할 $SS_f(l)$은 다음과 같이 정의된다.

$$\begin{array}{l}SS_f(l)=\{l'\in L_f \mid \exists e \text{ s.t.}\\ \quad \{\ \}\vdash e\Downarrow_{\{l'\}} v, F,\\ \quad Eval(v)\neq\perp,\\ \quad \{f\to \overline{\text{fcl}}((p_1,e_1)|\ldots|(p_n,e_n))\}\vdash f\ e:l_0 \Downarrow_{\{l'\}} v', F',\\ \quad \perp\in \overline{Eval}(F'(l))\ \}.\end{array}$$

■

$\perp$을 포함하지 않는 어떤 실인자값 $v$에 대한 함수 $f$의 적용을 계산했을 때에 만약 정적 분할 기준 $l$에서 계산되었던 값들 가운데 $\perp$을 포함하는 것이 있다면, 이는 부분식 $l$이 $l'$에서 계산된 값을 사용했다는 뜻이 된다. 따라서 이때 $l'$은 $l$의 정적 분할에 포함된다.

## 3. 요약된 오퍼레이셔널 의미(abstract operational semantics) 정의

정적 분할은 함수에 적용 가능한 모든 실인자를 고려하여 정의되었다. 따라서 위의 정의에 의하여 정적 분할을 찾는다면 무한한 모든 실인자들에 대하여 함수 적용을 계산해 보아야 한다. 본 절에서는, 정적 분할을 컴파일 시간에 분석하기 위하여 요약된 프로그램의 의미 정의를 제시한다.

### 3.1 수행을 위한 값집합의 요약

본 연구는 정적 분할의 분석을 목적으로 하므로, 요약된 수행을 위한 값집합은 정적 분할 분석의 정확성을 최대한 유지하면서, 최대한 간략하게 프로그램 의미를 나타내어야 한다. 다음은 요약된 오퍼레이셔널 의미 정의를 위한 값들의 집합이다.

$$\begin{array}{l}v \in Val' ::= \mathbf{1} \mid (v_1,v_2) \mid x\ v \mid v_\perp \mid \mathbf{v} \mid \perp \mid \top\\ F \in Collection' : Label\to 2^{Val'}\\ \rho \in Env' : Id\to(2^{Val'}\cup Fcl)\end{array}$$

$Val'$은 $Val$을 요약한 집합이다. 1은 실제적인 의미 정의를 위한 값 중에서 모든 상수값들을 나타내며

$(v_1, v_2)$는 튜플값들을 나타낸다. 그리고, $x\ v$는 자료 생성자 $x$의 적용에 의하여 생성된 값들을 나타내며, $v_\perp$은 어떤 값에 바깥쪽에 $\perp$이 붙여진 값들을 나타낸다. $\mathbf{v}$는 $\perp$을 전혀 포함하지 않은 값들에 대응되며, $\perp$은 모든 부분들에 $\perp$이 붙여진 값들을 나타낸다. 따라서 $\perp$과 $\mathbf{v}_\perp$이 나타내는 값들의 교집합은 $1_\perp$이 나타내는 값들이 된다. 또 $\top$은 모든 실제적인 값(concrete value)들의 집합 $Val$에 대응된다. $Collection'$과 $Env'$도 이러한 $Val$의 요약에 기준하여 정의될 수 있다.

$$
\begin{array}{l}
- \ Abs_{Val} : 2^{Val} \to 2^{Val'} \\
\quad Abs_{Val}(V) \ = \ \{Abs_v(v) \mid v \in V\} \\
\\
- \ Abs_v : Val \to Val' \\
\quad Abs_v(\mathbf{c}) \ = \ 1 \\
\quad Abs_v(v_\perp) \ = \ (Abs_v(v))_\perp \\
\quad Abs_v(\kappa\ v) \ = \ \kappa\ (Abs_v(v)) \\
\quad Abs_v((v_1, v_2)) \ = \ (Abs_v(v_1), Abs_v(v_2)) \\
\\
- \ Abs_{Col} : Collection \to Collection' \\
\quad Abs_{Col}(F)(l) \ = \ Abs_{Val}(F(l)) \\
\\
- \ Abs_{Env} : Env \to Env' \\
\quad (Abs_{Env}(\rho))(id) = \begin{array}{ll} Abs_v(\rho(id)) & \text{if } id \text{ is a variable} \\ \rho(id) & \text{if } id \text{ is a function name} \end{array}
\end{array}
$$

그림 4 값의 요약을 위한 함수들

$$
\begin{array}{l}
- \ Conc_{Val} : 2^{Val'} \to 2^{Val} \\
\quad Conc_{Val}(V) \ = \ \bigcup_{v \in V} Conc_v(v) \\
\\
- \ Conc_{Val} : 2^{Val'} \to 2^{Val} \\
\quad Conc_v(1) \ = \ \{\mathbf{c} \mid \mathbf{c} \in Val\} \\
\quad Conc_v(v_1, v_2) \ = \ \{(v_1', v_2') \mid v_i' \in Conc_v(v_i)\} \\
\quad Conc_v(\kappa\ v) \ = \ \{\kappa\ v' \mid v' \in Conc_v(v)\} \\
\quad Conc_v(v_\perp) \ = \ \{v_\perp' \mid v' \in Conc_v(v)\} \\
\quad Conc_v(\mathbf{v}) \ = \ \{v \in Val \mid Eval(v) \neq \perp\} \\
\quad Conc_v(\perp) \ = \ \{Mask(v) \mid v \in Val\} \\
\quad Conc_v(\top) \ = \ Val \\
\\
- \ Conc_{Col} : Collection' \to Collection \\
\quad Conc_{Col}(F)(l) \ = \ Conc_{Val}((F(l)) \\
\\
- \ Conc_{Env} : Env' \to (Id \to 2^{Val}) \\
\quad (Conc_{Env}(\rho))(id) = \begin{array}{ll} Conc_{Val}(\rho(id)) & \text{if } id \text{ is a variable} \\ \{\rho(id)\} & \text{if } id \text{ is a function name} \end{array}
\end{array}
$$

그림 5 값의 실제화를 위한 함수들

그림 4의 함수들은 실제적인 의미 정의를 위하여 사용되는 값들에 대응되는 요약된 값집합의 원소를 정의하고 있다. $Abs_v$는 $Val$의 원소를 요약하는 함수이며, $Abs_{Val}$은 이를 집합으로 확장한 것이다. 또 $Abs_{Col}$과 $Abs_{Env}$는 콜렉션과 환경을 요약하는 함수이다.

이러한 함수들과 반대로, 요약된 수행을 위한 값을 이것이 나타내는 실제 값들의 집합으로 바꾸는 것을 실제화(concretization)라고 한다. 그림 5는 실제화를 수행하는 함수들이다.

제시된 요약과 실제화 함수들의 성질을 기술하기 전에, 요약된 수행을 위한 값들 사이의 관계를 나타내는 표기법을 정의한다. 요약된 계산을 위한 값들의 관계는 이에 대응하는 실제 값들의 집합들 사이의 포함관계를 고려하여 표기법 2와 같이 정의된다.

**표기법 1**

- $f_1 \subset f_2$ *means that* $\forall x \in domain(f_1),\ f_1(x) \subset f_2(x)$.
- $f_1 \in f_2$ *means that* $\forall x \in domain(f_1),\ f_1(x) \in f_2(x)$.

**표기법 2**

- $\forall V_1, V_2 \subset Val',\ V_1 \widehat{=} V_2$ *means that* $Conc_{Val}(V_1) = Conc_{Val}(V_2)$.
- $\forall v_1, v_2 \in Val',\ v_1 \widehat{\subset} v_2$ *means that* $Conc_{Val}(v_1) \subset Conc_{Val}(v_2)$.
- $\forall V_1, V_2 \subset Val',\ V_1 \widehat{\subset} V_2$ *means that* $Conc_{Val}(V_1) \subset Conc_{Val}(V_2)$.
- $\forall F_1, F_2 \in Collection',\ F_1 \widehat{\subset} F_2$ *means that* $\forall l \in domain(F_1),\ F_1(l) \widehat{\subset} F_2(l)$.
- $\forall F_1, F_2 \in Collection',\ F_1 \widehat{=} F_2$ *means that* $F_1(l) \widehat{\subset} F_2(l)$ *and* $F_2(l) \widehat{\subset} F_1(l)$.
- $\rho_1, \rho_2 \in Env',\ \rho_1 \widehat{\subset} \rho_2$ *means that* $\forall\ id \in domain(\rho_1)$, *if* $id$ *is a function name,* $\rho_1(id) = \rho_2(id)$ *and if* $id$ *is a variable,* $\rho_1(id) \widehat{\subset} \rho_2(id)$.
- $\forall \rho_1, \rho_2 \in Env',\ \rho_1 \widehat{=} \rho_2$ *means that* $\rho_1 \widehat{\subset} \rho_2$ *and* $\rho_2 \widehat{\subset} \rho_1$.

정의된 요약 및 실제화 함수들은, 요약된 의미론에 기반한 프로그램 수행 결과가 실제 수행의 결과를 건전하게 나타냄을 증명하는데 유용한 다음과 같은 일련의 성질들을 만족한다.

**성질 1**

- $\forall v \in Val,\ v \in Conc_v(Abs_v(v))$
- $\forall V \subset Val,\ Val \subset Conc_{Val}(Abs_{Val}(V))$ ■

**성질 2**

- $\forall v \in Val',\ (Abs_{Val}(Conc_{Val}(\{v\}))) \widehat{=} \{v\}$.
- $\forall V \subset Val',\ (Abs_{Val}(Conc_{Val}(V))) \widehat{=} V$. ■

**성질 3**

$\forall v \in Val$ and $V \subset Val'$, if $v \in Conc_{Val}(V)$ *then* $\{Abs_v(v)\} \widehat{\subset} V$.

■

**성질** 1은 실제 수행을 위한 값을 요약했을 때, 이 값을 다시 실제값들의 집합으로 바꾸면 이 집합이 원래의 실제값을 포함함을 나타낸다. **성질** 2는 요약된 수행을 위한 값을 실제화 한 후에 다시 요약을 수행하면, 원래의 값과 $\cong$ 관계, 즉 같은 실제값을 나타내는 값으로 됨을 의미한다. **성질** 2는 값의 요약이 해당 값을 나타내는 요약 공간의 가장 작은 값을 결과로 내기 때문에 성립하며, 이는 **성질** 3에 기술되어 있다.

### 3.2 요약된 오퍼레이셔널 의미 정의

이전 절에서 정의된 요약 수행을 위한 값의 집합을 기반으로 하여 요약된 오퍼레이셔널 의미론을 정의할 수 있다. 요약된 의미론의 정의에서도 마찬가지로 그림 6과 같은 부가적인 함수가 필요하다.

- Eager Evaluation $Eval' : Val' \to Val'$

$$
\begin{aligned}
Eval'(\mathbf{1}) &= \mathbf{1}\\
Eval'(\mathbf{v}) &= \mathbf{v}\\
Eval'(\bot) &= \bot\\
Eval'(v_\bot) &= \bot\\
Eval'(\kappa\ v) &= \text{if } (Eval'(v) = \bot) \text{ then } \bot \text{ else } \kappa\ v\\
Eval'(v_1, v_2) &= \text{if } (Eval'(v_1) = \bot \text{ or } Eval'(v_2) = \bot)\\
&\quad \text{then } \bot \text{ else } (v_1, v_2)\\
Eval'(\top) &= \bot\\
\\
\overline{Eval'}(V) &= \{Eval'(v) \mid v \in V\}
\end{aligned}
$$

- Masking $Mask' : Val' \to Val'$

$$
\begin{aligned}
Mask'(\mathbf{1}) &= \mathbf{1}_\bot\\
Mask'(\mathbf{v}) &= \bot\\
Mask'(v_\bot) &= Mask'(v)\\
Mask'(v_1, v_2) &= (Mask'(v_1), Mask'(v_2))_\bot\\
Mask'(\kappa\ v) &= (\kappa\ Mask'(v))_\bot\\
Mask'(\bot) &= \bot\\
Mask'(\top) &= \bot
\end{aligned}
$$

그림 6 요약된 의미 정의에서 사용되는 부가 함수들

$Eval'$은 실제 수행에서의 $Eval$과 마찬가지로 값에 $\bot$이 포함되어 있는지를 검사한다. $Mask'$도 실제 수행에서 사용된 $Mask$처럼 주어진 값의 모든 부분에 $\bot$을 붙여주는 작업을 수행한다. 다음 성질들은 요약된 값에 대한 그림 6의 함수들이 원래 실제 수행시의 의미를 그대로 보존함을 나타낸다.

**성질 4**

- $\forall v \in Val,\ Eval(v) = \bot \ \textit{iff}\ Eval'(Abs_v(v)) = \bot$.
- $\forall V \subset Val,\ \bot \in \overline{Eval}(V) \ \textit{iff}\ \bot \in \overline{Eval'}(Abs_{Val}(V))$. ■

**성질 5**

- $\forall v \in Val',\ Eval'(v) = \bot \ \textit{iff}\ \bot \in \overline{Eval}(Conc_v(v))$.
- $\forall V \subset Val',\ \bot \in \overline{Eval'}(V) \ \textit{iff}\ \bot \in \overline{Eval}(Conc_{Val}(V))$. ■

**성질 6**

$\forall v \in Val',\ Conc_v(Mask'(v)) = \{Mask(v') \mid v' \in Conc_v(v)\}$ ■

요약된 오퍼레이셔널 의미 정의는 그림 7과 그림 8의 표현식의 계산 규칙과 패턴 매칭 규칙으로 이루어진다. 표현식의 요약된 계산 규칙은 실제 의미와 마찬가지로 각각의 프로그램 구조에 대하여 정의되며 표현식의 값과 콜렉션을 결과로 생성한다. 표현식의 값이 하나의 값이 아니라 값의 집합이 되는 이유는 요약된 수행의 경우 결과가 될 수 있는 모든 값을 결과로 생성하게 되기 때문이다. 또, 패턴 매칭 규칙은 요약 수행을 위한 값들의 집합을 패턴과 매칭하여 변수에 대한 환경과 콜렉션을 생성하게 된다. 이때 생성되는 환경도 각각의 변수에 대하여 지정될 가능성이 있는 모든 값들의 집합을 저장하게 된다.

$$
\frac{}{\rho \vdash c : l \Downarrow'_M \{1_\bot\}, \{l \to \{1\}\} \text{ if } l \in M \quad \rho \vdash c : l \Downarrow'_M \{1\}, \{l \to \{1\}\} \text{ otherwise}} \ \text{Const}
$$

$$
\frac{}{\rho \vdash x : l \Downarrow'_M \{v_\bot \mid v \in \rho(x)\}, \{l \to \rho(x)\} \text{ if } l \in M \quad \rho \vdash x : l \Downarrow'_M \rho(x), \{l \to \rho(x)\} \text{ otherwise}} \ \text{Var}
$$

$$
\frac{\rho \vdash e \Downarrow'_M V, F}{\rho \vdash \kappa\ e : l \Downarrow'_M V'', F \cup \{l \to V'\} \text{ if } l \in M \quad \Downarrow'_M V', F \cup \{l \to V'\} \text{ otherwise}} \ \text{DCons}
$$

where $V' = \{\kappa\ v \mid v \in V\}$ and $V'' = \{v_\bot \mid v \in V'\}$

$$
\frac{\rho \vdash e_i \Downarrow'_M V_i, F_i \quad (i = 1, 2)}{\rho \vdash (e_1, e_2) : l \Downarrow'_M V', F_1 \cup F_2 \cup \{l \to V\} \text{ if } l \in M \quad \Downarrow'_M V, F_1 \cup F_2 \cup \{l \to V\} \text{ otherwise}} \ \text{Tuple}
$$

where $V = \{(v_1, v_2) \mid v_1 \in V_1, v_2 \in V_2\}$ and $V' = \{v_\bot \mid v \in V\}$

$$
\frac{\rho \vdash e_i \Downarrow'_M V_i, F_i \quad (i = 1, 2)}{\rho \vdash e_1 + e_2 : l \Downarrow'_M \{\mathbf{v}, \bot\}, F_1 \cup F_2 \cup \{l \to \{\mathbf{v}, \bot\}\} \text{ if } \exists i \text{ s.t. } \bot \in \overline{Eval'}(V_i) \quad \Downarrow'_M \{\bot\}, F_1 \cup F_2 \cup \{l \to \{\mathbf{v}\}\} \text{ elseif } l \in M \quad \Downarrow'_M \{\mathbf{v}\}, F_1 \cup F_2 \cup \{l \to \{\mathbf{v}\}\} \text{ otherwise}} \ \text{BOp}
$$

$$
\frac{\rho \vdash e \Downarrow'_M V, F \quad p \Vdash'_M V \Rightarrow \rho', F' \quad \rho \oplus \rho' \vdash e' \Downarrow'_M V', F''}{\rho \vdash \text{let } p = e \text{ in } e' \text{ end} : l \Downarrow'_M V'', F \cup F' \cup F'' \cup \{l \to V'\} \text{ if } l \in M \quad \Downarrow'_M V', F \cup F' \cup F'' \cup \{l \to V'\} \text{ otherwise}} \ \text{Let}
$$

where $V'' = \{v_\bot \mid v \in V'\}$

$$
\frac{\rho(f) = \text{fcl}((p_1, e_1) | \ldots | (p_n, e_n)) \quad \rho \vdash e \Downarrow'_M V, F \quad \forall i_j \text{ s.t. } p_{i_j} \Vdash'_M V \Rightarrow \rho_{i_j}, F_{i_j} \quad \{f \to \text{fcl}((p_1, e_1) | \ldots | (p_n, e_n))\} \oplus \rho_{i_j} \vdash e_{i_j} \Downarrow'_M V_{i_j}, F'_{i_j} \quad (1 \le j \le k)}{\rho \vdash (f\ e) : l \Downarrow'_M V'', F \cup F_{i_1} \cup \ldots \cup F_{i_k} \cup F'_{i_1} \cup \ldots \cup F'_{i_k} \cup \{l \to V'\} \text{ if } l \in M \quad \Downarrow'_M V', F \cup F_{i_1} \cup \ldots \cup F_{i_k} \cup F'_{i_1} \cup \ldots \cup F'_{i_k} \cup \{l \to V'\} \text{ otherwise}} \ \text{App}
$$

where $V' = V_{i_1} \cup \ldots \cup V_{i_k}$ and $V'' = \{v_\bot \mid v \in V'\}$

그림 7 요약된 의미 정의 - 표현식의 계산

$$
\begin{array}{l}
p \Vdash'_M V \Rightarrow \rho, F \quad \text{where } \exists v_i \in V_i \text{ s.t. } p \Vdash' v_i \Rightarrow \rho_i, F_i, \text{ where } \rho = \rho_1 \cup \ldots \cup \rho_n \text{ and } F = F_1 \cup \ldots \cup F_n.\\
\\
\_ \Vdash'_M v \Rightarrow \{\}, \{\}\\
c \Vdash'_M v \Rightarrow \{\}, \{\}\\
x : l \Vdash'_M v \Rightarrow \{x \to \{v, Mask'(v)\}\}, \{l \to \{v\}\} \quad \text{if } l \in M \text{ or } Eval'(v) = \bot\\
x : l \Vdash'_M v \Rightarrow \{x \to \{v\}\}, \{l \to \{v\}\} \quad \text{otherwise}\\
(p_1, p_2) \Vdash'_M (v_1, v_2) \Rightarrow \rho_1 \cup \rho_2, F_1 \cup F_2 \quad \text{where } p_i \Vdash'_M v_i \Rightarrow \rho_i, F_i \ (i = 1, 2)\\
(p_1, p_2) \Vdash'_M v_\bot \Rightarrow \rho, F \quad \text{where } (p_1, p_2) \Vdash'_M v \Rightarrow \rho, F\\
(p_1, p_2) \Vdash'_M \mathbf{v} \Rightarrow \rho_1 \cup \rho_2, F_1 \cup F_2 \quad \text{where } p_i \Vdash'_M \mathbf{v} \Rightarrow \rho_i, F_i \ (i = 1, 2)\\
(p_1, p_2) \Vdash'_M \bot \Rightarrow \rho_1 \cup \rho_2, F_1 \cup F_2 \quad \text{where } p_i \Vdash'_M \bot \Rightarrow \rho_i, F_i \ (i = 1, 2)\\
(p_1, p_2) \Vdash'_M \top \Rightarrow \rho_1 \oplus \rho_2, F_1 \cup F_2 \quad \text{where } p_i \Vdash'_M \top \Rightarrow \rho_i, F_i \ (i = 1, 2)\\
\kappa\ p \Vdash'_M \kappa\ v \Rightarrow \rho, F \quad \text{where } p \Vdash'_M v \Rightarrow \rho, F\\
\kappa\ p \Vdash'_M v_\bot \Rightarrow \rho, F \quad \text{where } \kappa\ p \Vdash'_M v \Rightarrow \rho, F\\
\kappa\ p \Vdash'_M \mathbf{v} \Rightarrow \rho, F \quad \text{where } p \Vdash'_M \mathbf{v} \Rightarrow \rho, F\\
\kappa\ p \Vdash'_M \bot \Rightarrow \rho, F \quad \text{where } p \Vdash'_M \bot \Rightarrow \rho, F\\
\kappa\ p \Vdash'_M \top \Rightarrow \rho, F \quad \text{where } p \Vdash'_M \top \Rightarrow \rho, F
\end{array}
$$

그림 8 요약된 의미 정의 - 패턴 매칭 규칙

다음의 보조 정리와 정리는 요약된 의미 정의에 의한 프로그램 수행 결과를 실제화 한 것이 원래의 의미론에 의한 수행 결과를 포함함을 나타낸다.

**보조정리 1 요약된 의미 정의의 패턴 매치의 건전성**

$\forall v \in Val$ and $v' \in Val'$, *if* $\rho \vdash_M v \Rightarrow v, F$ *and* $v \in Conc_v(v')$, *then* $p \vdash_{M'} v' \Rightarrow \rho', F'$ *where* $\rho \in Conc_{Env}(\rho')$, *and* $F \subset Conc_{Col}(F')$ ■

**정리 1 요약된 오퍼레이셔널 의미 정의의 건전성**

$\forall \rho'$ *s.t.* $\rho \in Conc_{Env}(\rho')$, *if* $\rho \vdash e \Downarrow_M v, F$ *and* $\rho' \vdash e \Downarrow'_M V, F'$, *then* $v \in Conc_{Val}(V)$ *and* $F \subset Conc_{Col}(F')$. ■

## 4. 정적 분할의 분석

본 절에서는 요약된 오퍼레이셔널 의미 정의에 기반하여 정적 분할을 분석하는 방법을 기술한다. 정적 분할의 정의에서 고려한 함수의 모든 실인자들은 $\perp$을 포함하지 않으므로, 요약된 의미론에서는 **v**로 이러한 실인자의 집합을 건전하게 근사(approximate)할 수 있다. 따라서, 요약된 의미 정의에 기반하여 주어진 함수에 **v**를 실인자값으로 적용하여 계산함으로써, 정적 분할의 건전한 분석을 수행할 수 있다.

그런데, 실제 수행에서 어떤 표현식의 계산이 종결(terminate)된다 하더라도, 이에 대응하는 요약된 의미 정의에 의한 계산에서는 프로그램이 종결하지 않는 경우가 발생한다. 이러한 경우 분석이 끝나지 않게 되므로 이에 대한 해결책이 필요하다. 본 연구에서는, 요약된 수행을 위한 값들의 집합을 유한개의 원소를 갖는 집합으로 근사함으로써, 유한한 시간 내에 정적 분할을 분석하고자 하였다.

- Pruning Abstract Values

$Prune_{Val}(d, V) = \{Prune_v(d, v) \mid v \in V\}$
$(Prune_{Col}(d, F))(l) = Prune_{Val}(d, F(l))$

$Prune_v(\neg 1) = \mathbf{1}$
$Prune_v(\neg \mathbf{v}) = \mathbf{v}$
$Prune_v(\neg \perp) = \perp$
$Prune_v(\neg \top) = \top$
$Prune_v(0, v) = \top$
$Prune_v(n, v_\perp) = (Prune_v(n, v))_\perp$
$Prune_v(n, \kappa\, v) = \kappa\,(Prune_v(n-1), v)$
$Prune_v(n, (v_1, v_2))$
$= (Prune_v(n-1, v_1), Prune_v(n-1, v_2))$

그림 9 요약 수행을 위한 값들의 중첩 깊이 제한

그림 9에서 정의된 함수 $Prune_v$는 주어진 값에 대하여 값의 구조적인 깊이를 제한하기 위하여 일정 깊이보다 안쪽의 부분들을 $\top$으로 대치한다. 이렇게 값의 깊이를 제한함으로써, 요약된 수행을 위한 값들의 집합을 유한하게 만들 수가 있다. 예를 들면, 요약 수행 공간의 값 $((\top,(1,(\perp,(1,\top)))),(1,(\top,(1,x\ \perp))))$은 상수값의 깊이를 0으로 생각할 때, 최대 깊이 5의 중첩된 구조를 가지고 있다. 만약 이 값에 대하여 깊이를 3으로 제한하면 다음과 같이 계산된다.

$Prune_v(3, ((\top,(1,(\perp,(1,\top)))),(1,(\top,(1,x\ \perp)))))$
$= ((\top,(1,\top)),(1,(\top,\top)))$

즉 깊이 3보다 안쪽에 위치한 중첩된 구조를 가진 부분들인 $(\perp,(1,\top))$과 $(1,x\ \perp)$을 모든 실제적인 값들은 나타내는 $\top$으로 대치함으로써, 건전성을 유지하면서 깊이를 제한하게 된다.

복잡하게 중첩된 값의 구조를 요약된 수행에서 유지하는 것은, 자료 흐름의 분석시에 패턴 매칭에 대한 좀 더 세밀한 분석을 수행하기 위해서인데, 일반적인 프로그램에서 단순한 튜플이나 자료 생성자 패턴이 아닌 매우 복잡한 패턴을 사용하는 경우는 흔하지 않으므로, 5 이하의 적당한 $d$값을 사용함으로써 분석의 정확도를 유지할 수가 있다. 다음 보조 정리는 $Prune_v$가 주어진 값에 대한 건전한 근사를 수행함을 나타낸다.

**보조정리 2**

$\forall v \in Val'$ and $d \geq 0$, $v \widehat{\subset} Prune_v(d, v)$. ■

정적 분할의 분석을 정의하기 위하여 먼저 $\perp$을 포함하지 않는 실인자들에 대한 함수 적용의 모든 결과값들을 근사하는 방법을 제시한다.

**정의 2 함수 적용 결과의 근사**

*$f\ p_1 = e_1 \mid \ldots \mid p_n = e_n$로 정의되는 함수 $f$에, $\perp$을 포함하지 않는 값으로 계산되는 임의의 표현식 $e$를 적용한 $(f\ e)$의 수행 결과들에 대한 요약된 근사값 $Res_{f,M,d}$는 다음과 같이 정의된다.*

$\widehat{A} = \{\ \mathbf{v}\ \}$
$\widehat{V} = \{\ \}$
$\widehat{F} = \{\ \}$

*repeat until there is no change in* $\widehat{A}$, $\widehat{V}$, and $\widehat{F}$
{
$\forall p_i, e_i\ (1 \leq i \leq n)$ *s.t.* $p_i \mid \vdash' \widehat{A} \Rightarrow \rho_i, F_i$
$\rho_i \vdash e_i \Downarrow_{(V,F),M} V_i, F_i$
(*where* $\forall\ (f\ e_{ij})$ *in* $e_i\ (1 \leq j \leq r_i)$,
$e_{ij}$ *has been evaluated to* $V_{ij}, F_{ij}$)
$\widehat{V} = Prune_{Val}(d, (\widehat{V} \cup V_1 \cup \ldots \cup V_n))$
$\widehat{F} = Prune_{Col}(d, (\widehat{F} \cup F_1 \cup \ldots \cup F_n \cup F_1 \cup \ldots \cup F_n))$
$\widehat{A} = \widehat{A} \cup V_{11} \cup \ldots \cup V_{1r_1} \cup \ldots \cup V_{n1} \cup \ldots \cup V_{nr_n}$
}
$Res_{f,M,d} = (\widehat{A}, \widehat{V}, \widehat{F})$

*이때* $\Downarrow''_{(V,F),M}$ *은* $\Downarrow'_M$*과 재귀 호출의 계산을 위한 다음 규칙만 다르다.*

$$\frac{\rho \vdash e \Downarrow''_{(V,F),M} V'', F'}{\rho \vdash (f\ e):l \Downarrow''_{(V,F),M} V', F \cup F' \text{ if } l \in M \quad \Downarrow''_{(V,F),M} V, F \cup F' \text{ if } l \notin M}$$

*where* $V' = \{v_\perp \mid v \in V\}$. ■

위에서 정의된 $Res_{f,M,d}$에서 $f$는 주어진 함수이고 $M$은 값이 차단되는 레이블의 집합이며, $d$는 요약된 수행을 위한 값들의 집합을 유한하게 만들기 위하여 사용되는 최대 깊이값이다. 위의 알고리즘에서는 최초 실인자의 집합을 {v}로 주고, 재귀 호출들에서 사용된 인자 값들을 실인자의 집합에 더해가면서 더 이상 결과값과 실인자 집합이 증가하지 않을 때까지 함수의 적용을 반복하게 된다. 결과값의 집합과 실인자 집합은 $d$에 의하여 유한하게 크기가 제한되고, 반복마다 실인자 집합과 결과값 집합은 감소하지 않으므로, 위의 반복은 반드시 종료하게 된다.

다음 정리는 정의 1에 의한 분석의 결과가 건전함을 나타낸다.

**정리 2**

*주어진 함수* $f\ p_1 = e_1 \mid \ldots \mid p_n = e_n$*에 대하여* $Res_{f,M,d} = (\hat{A}, \hat{V}, \hat{F})$*라고 하자. 이 때 어떤 표현식 e가* $v \in Conc_{Val}(\hat{A})$*인 v로* $\{\} \vdash \Downarrow_M v, F$*와 같이 계산되고 e에 대한 함수 f의 적용* $(f\ e):l_0$*가* $\{f \rightarrow \mathrm{fcl}((p_1, e_1) \mid \ldots \mid (p_n, e_n))\} \vdash (f\ e):l_0 \Downarrow_M v', F'$*와 같이 계산된다면,* $v' \in Conc_{Val}(\hat{V})$*이고 함수 몸체 내의 임의의 레이블 l에 대하여* $F'(l) \subset Conc_{Col}(\hat{F})(l)$*이 성립한다.* ■

정의 2에 의하여 얻어진 $Res_{f,M,d}$를 사용하여 함수 $f$의 정적 분할을 다음과 같이 직접 분석할 수가 있다. 다음 정의에서 $SS'_{f,d}$는 함수 $f$의 정적 분할을 근사하며, 그 결과의 건전성은 정리 3에 의하여 증명된다.

**정의 3 정적 분할의 분석**

주어진 함수 $f\ p_1 = e_1 \mid \ldots \mid p_n = e_n$에 대하여 함수 몸체 내의 레이블 $l$에 대한 정적 분할의 분석 $SS'_{f,d}(l)$은 다음과 같이 정의된다.

$$SS'_{f,d}(l) = \{l' \in L_f \mid Res_{f,\{l'\},d} = (V, F) \text{ and } \perp \in \overline{Eval'}(F(l'))\}$$ ■

**정리 3 정적 분할 분석의 건전성**

$$\forall l \in L_f \text{ and } d \geq 0,\ SS_f(l) \subset SS'_{f,d}(l).$$ ■

```
datatype tree   = Leaf of int
                | Node of int * tree * tree

f  (Leaf x, n) = ((Leaf n1)2,l3)4
|  (Node(x,lt,rt), n) =
 let
   (t1_5,s1_6) = (f (lt7,n8)9)10;
   (t2_11,s2_12) = (f (rt13,((l14+n15)16+s1_17)18)19)20
 in
   ((Node ((n21+s1_22)23,t1_24,t2_25)26)27,
    ((s1_28+s2_29)30+l31)32)33
 end34
```

$SS_f(1) = SS_f(8) = SS_f(15) = SS_f(16) = SS_f(18) = SS_f(21) = \{3, 6, 8, 12, 14, 15, 16, 17, 18, 28, 29, 30, 31, 32\}$
$SS_f(2) = \{1, 3, 6, 8, 12, 14, 15, 16, 17, 18, 28, 29, 30, 31, 32\}$
$SS_f(3) = SS_f(7) = SS_f(13) = SS_f(14) = SS_f(31) = \{\}$
$SS_f(4) = \{1, 2, 3, 6, 8, 12, 14, 15, 16, 17, 18, 28, 29, 30, 31, 32\}$
$SS_f(5) = SS_f(11) = SS_f(24) = SS_f(25) = SS_f(26) = SS_f(27) = SS_f(33) = \{1, 2, 3, 5, 6, 8, 11, 12, 14, 15, 16, 17, 18, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32\}$
$SS_f(6) = SS_f(12) = SS_f(17) = SS_f(22) = SS_f(28) = SS_f(29) = SS_f(30) = SS_f(32) = \{3, 6, 12, 28, 29, 30, 31, 32\}$
$SS_f(9) = \{3, 6, 7, 8, 12, 14, 15, 16, 17, 18, 28, 29, 30, 31, 32\}$
$SS_f(10) = SS_f(20) = \{1, 2, 3, 4, 5, 6, 8, 11, 12, 14, 15, 16, 17, 18, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34\}$
$SS_f(19) = \{3, 6, 8, 12, 13, 14, 15, 16, 17, 18, 28, 29, 30, 31, 32\}$
$SS_f(23) = \{3, 6, 8, 12, 14, 15, 16, 17, 18, 21, 22, 28, 29, 30, 31, 32\}$
$SS_f(34) = \{1, 2, 3, 5, 6, 8, 11, 12, 14, 15, 16, 17, 18, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33\}$

그림 10 정적 분할의 예

그림 10은 정적 분할의 예로서 함수 f의 모든 부분식들에 대하여 정적 분할을 분석한 결과이다. f는 트리(tree)와 시작 번호를 인자로 받아서 인오더(inorder) 순서로 번호를 붙인 트리와 트리 크기의 튜플을 생성하는 함수이다. 각각의 부분식에 붙여진 첨자는 레이블을 나타낸다. 결과를 보면, 각각의 부분식 $l$에 대하여 $l$에 속하는 부분식들이나, 또는 계산된 값이 지정문, 함수 호출시의 실인자, 재귀 호출의 결과값 등을 통하여 $l$에 전달되는 식들이 $l$의 정적 분할에 포함됨을 알 수 있다. 예를 들어, 정적 함수의 수행 결과로 생성되는 튜플 값의 왼쪽 원소를 만드는 부분식 27번은 함수 수행 결과의 왼쪽 원소가 지정되는 변수 패턴 5번의 정적 분할에 포함된 것을 볼 수 있으며, 함수 수행 결과의 오른쪽 원소를 계산하는 부분식 32번은 변수 패턴 5번의 정적 분할에는 포함되지 않고 함수 결과의 오른쪽 값이 지정되는 변수 패턴 6번의 정적 분할에 포함되어 있음을 볼 수 있다.

## 5. 결 론

본 연구에서는 함수형 언어의 정적 분할을 오퍼레이셔널 의미론을 사용하여 정형적으로 정의하고, 요약 해석 방법론에 기반한 분석 방법을 제시하였다. 제시한 방법은 적절한 요약 수행 공간의 설계에 의하여 튜플 패턴에 대한 지정이나 자료 생성자 패턴에 대한 지정시의

복잡한 자료 흐름을 분석할 수 있는 특징을 가진다. 또한 제시된 분석 방법이 정적 분할의 정의에 대하여 건전한 결과를 냄을 증명하였다.

본 연구는 원래 함수형 언어의 병렬화 컴파일러 개발을 위하여 수행되었다[11]. 그러나, 본 연구에서 제시된 특정 부분식에서의 값 차단을 통한 자료 흐름 분석 방법은 불필요한 코드 제거, 처리되지 않는 예외 상황의 검출 등의 다양한 분석 및 최적화에 응용될 수 있을 것으로 판단된다.

본 연구의 향후 연구로서는 입력 언어의 범위를 고차 함수형 언어(higher-order functional languages)로 확장하는 작업을 들 수 있다. 이를 위해서는, 함수의 결과 값이나 실인자로 함수가 사용될 수 있고, 부분적으로 적용된(partially applied) 함수를 처리할 수 있도록 언어의 의미의를 확장하고, 이에 따른 적절한 요약된 의미를 설계하여야 한다. 이러한 작업을 통하여, 본 연구 결과를 고차 함수형 언어의 다양한 분석 및 최적화 작업에도 사용할 수 있을 것으로 판단된다.

## 참 고 문 헌


[1] M. Weiser. "Program slicing," *IEEE Transactions on Software Engineering*, Vol. 10 No.4 pp.352-357, April 1984.

[2] Frank Tip, "A survey of program slicing techniques," Technical Report CS-R9438, Centrum voor Wiskunde en Informatica, 1994.

[3] B. Korel and J. Laski, "Dynamic program slicing," *Information Processing Letters*, Vol. 29, pp. 155-163, 1988.

[4] Hiralal Agrawal and Joseph R. Horgan, "Dynamic program slicing". In *Proceedings of the ACM SIGPLAN '90 Conference on Programming Language Design and Implementation*, pp. 246-256, June 1990.

[5] G. A. Venkatesh, "The semantic approach to program slicing," In *Proceedings of the ACM SIGPLAN '91 Conference on Programming Language Design and Implementation*, pp. 107-119, June, 1991.

[6] Sandip K. Biswas, Dynamic Slicing in Higher-Order Program Languages, PhD thesis, University of Pennsylvania, 1998.

[7] Flemming Nielson, "A denotational framework for data flow analysis," Acta Informatica, Vol. 18, pp. 265-287, 1982.

[8] Neil D. Jones and Alan Mycroft, "Data flow analysis of applicative programmings using minimal function graphs : abridged version," In *Proceedings of the 13th ACM Symposium on Principles of Programming Languages*, pp. 296-306, Florida, January, 1986.

[9] Patrick Cousot and Radhia Cousot, "Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints." In *Proceedings of the 4th ACM Symposium on Principles of Programming Languages*, pp. 238-252, Los Angeles, California, January, 1977.

[10] Joonseon Ahn and Taisook Han, "Static Slicing of a First-Order Functional Language base on Operational Semantics," Technical Report CS/TR-99-144, KAIST, 1999.

[11] Joonseon Ahn and Taisook Han, "Analysis of Parallelism in Recursive Functions on Recursive Data Structures," In *Proceedings of the International Workshop on Implementation of Declarative Languages*, Paris, France, September, 1999.



안 준 선

1992년 서울대학교 계산통계학과 학사. 1994년 한국과학기술원 전산학과 석사. 1994년 ~ 현재 한국과학기술원 전자전산학과 전산학전공 박사과정. 관심분야는 함수형 언어, 병렬화 컴파일러, 프로그램 분석, 프로그램 최적화 등임



한 태 숙

1976년 서울대학교 전자공학과 학사. 1978년 한국과학기술원 전산학과 석사. 1990년 Univ. of North Carolina at Chapel Hill 박사. 현재 한국과학기술원 전자전산학과 전산학전공 부교수. 관심분야는 프로그래밍 언어론, 함수형 언어.