실행시간 전문화를 위한 집합기반분석의 준비
(Preparing Set-Based Analysis for Run-time Specialization)

어 현 준 † 이 광 근 ††
(Hyunjun Eo) (Kwangkeun Yi)

요 약 정적 분석을 사용하여 프로그램의 입력에 의존하는 성질을 예측하는 방법을 제안한다. 제안된 방법은 입력에 무관한 성질을 예측하도록 설계된 정적 분석을 입력에 의존하는 성질을 예측하는 분석으로 변환한다. 이 방법은 실행 중에 프로그램의 성질을 알아내기 위해서 실행중인 프로그램을 관찰하는 코드가 필요 없고 계측된 자료를 모으는 과정도 필요 없다.

정적 분석의 가장 마지막 부분을 프로그램의 실행 시간으로 미루는 것이 이 논문의 핵심 아이디어다. 먼저 정적 분석을 분석하여, 프로그램의 입력에 민감하여 프로그램의 실행시간으로 연기되어야 하는 부분을 찾아낸다. 그 후, 값을 자르는 분석을 사용하여 이 부분을 재구성하여 프로그램의 입력에 대한 간단한 멤버쉽 테스트에 의해 분석이 풀어질 수 있도록 한다. 이런 재구성 과정을 통해 준비된 분석들은 프로그램의 입력이 나타나기만 하면 순간적으로, 동시에 풀려질 수 있다.

모든 과정은 엄밀하게 정의되고 증명되었다.

Abstract We present a technique of using static analysis for estimating program's input-dependent properties. A static analysis that is originally designed for estimating the input-independent properties of programs is transformed into one that can safely estimate the input-dependent properties at the programs' input occurrence. No profile is collected and no probing codes inside the running program are needed.

Our idea is to defer the finish of the static analysis to the program's run-time. By analyzing the static analysis, we identify the parts of the analysis that are sensitive to the program's inputs, hence need to be deferred to the program's run-time. Then by using an analysis named static value-slicing, we short-cut some of the dynamic parts so that they are solved by simple membership tests for the program's input. This re-formulation accelerates the analysis; once the program's input occurs the prepared dynamic parts can immediately and simultaneously start to resolve.

Every step of our technique is formally defined and proven correct.

1. 서 론

1.1 동기

입력에 무관한 프로그램의 성질들은 최적화된 코드를 생성하기 위해 필요하다. 그러나 만약, 자주 사용되는 입력에 맞게 프로그램을 맞추고 싶다면, 입력에 의존하는 프로그램의 성질들도 필요하게 된다.

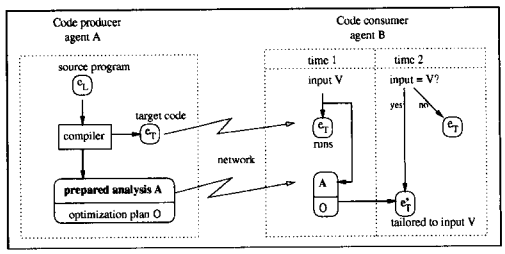

그림 1 시나리오

입력에 의존하는 성질들을 얻는 방법으로 가장 일반적인 것은 프로파일(profile)이다. 프로파일 기반 최적화 기법들은 프로파일된 결과들의 통계를 사용하여 가장 많이 사용된 입력에 대해 최적화된 목적 코드를 생성하게 된

· 본 연구는 한국과학재단 특정기초연구(과제번호 1999-12-28-406)의 지원을 받았음.

† 비 회 원 : 한국과학기술원 전산학과
poisson@ropas.kaist.ac.kr
†† 종신회원 : 한국과학기술원 전산학과 교수
kwang@ropas.kaist.ac.kr

논문접수 : 1999년 12월 28일
심사완료 : 2000년 7월 7일

다.

그러나, 프로파일 기반 분석들은 몇 가지 단점을 가지고 있다. 프로파일 자료로부터 얻어진 프로그램의 성질들은 몇 가지 최적화 기법들에 적용하기에는 부족하다. 일반적으로 프로파일된 통계 자료는 프로그램의 흐름이 특정 경로를 얼마나 자주 따라 갔나를 세는 것이다. 이런 정보는 분기 예측(branch prediction)에 민감한 최적화 기법들에는 유용하다. 그러나 병렬화와 같은 다른 최적화 방법들은 프로그램 변수들이 어떻게 읽혀지고 쓰여졌는가에 대한 정보를 필요로 하고, 이를 관찰하기 위해서는 프로그램의 어떤 경로가 몇 번 수행되었는지를 세는 것보다 더 많은 정보를 관찰할 필요가 있다. 게다가, 그런 정교한 실행시간 성질들을 수집하기 위해서는 프로그램의 실행에 훨씬 더 많은 부담(overhead)을 가지게 된다. 현재의 경로를 세는 프로파일은 15%-30%[1, 2]의 실행시간 부담이 있다. 프로그램의 내부에서 프로그램을 계측하는 코드가 더 복잡해지게 되면, 프로파일 부담과 정확성 사이의 균형을 맞추는 일은 더 어려워진다.

프로파일을 하지 않고 입력에 의존하는 프로그램의 성질들을 추정하는 일반적인 기술이 필요하다. 이런 상황은 글로벌 컴퓨팅 환경, 그리고 웹 컴퓨팅 환경에서 더욱 요구되어진다. 다수의 코드 소비자(사용자)가 네트웍 너머의 코드 제공자(컴파일러)와 떨어져 있는 상황을 생각해보자. 코드 제공자는 소스 프로그램을 컴파일하고, 컴파일된 코드를 네트웍의 다른쪽 끝에 있는 소비자에게 전달한다. 이 경우 입력에 의존하는 최적화들은 각각의 소비자들에게 위임된다. (만약 위임하지 않는 다면, 코드 제공자는 각각의 소비자들의 입맛에 맞는 코드를 만들기 위해서 궁지에 몰리게 될 것이다.) 소비자들은 소스 프로그램에 접근할 수 없기 때문에, 전달된 목적 코드에 대한 최적화는 제한될 수밖에 없다. 반면에, 코드 제공자는 각각의 소비자들의 입력 패턴에 맞추는 적극적인 최적화를 제공하기를 원한다. 코드 제공자가 원하는 적극적인 최적화는 소스 레벨에서의 정교한 프로그램 분석을 필요로 한다.

따라서, 다음의 시나리오가 예상된다. 코드 제공자는 입력에 의존하는 소스 프로그램의 성질을 소비자 싸이트(site)에서 예측할 수 있는 분석을 준비한다. 코드 제공자는 또한 추정된 소스 프로그램의 실행시간 성질로부터 최적화된 코드를 어떻게 생성할 지에 관한 정보를 가지고 있다. 코드 제공자는 소비자에게 컴파일된 목적 코드와 함께 이 두 가지 정보 - "소스 프로그램에 대한 준비된 분석 A", "목적 코드에 대한 최적화 계획 O" - 를 전달한다 (그림 1). 코드 소비자는 전달된 코드를 수행한다. 입력이 발생하면 코드의 수행과는 무관하게, 준비된 분석 A는 소스 프로그램의 동적 성질들을 추정한다. A에 의해 분석된 결과는 최적화 계획 O를 활성화 시켜서 입력에 맞게 최적화된 목적 코드를 생성하게 한다. 다음에 비슷한 입력이 발생하게 되면, 소비자는 원래의 코드 대신에 최적화된 코드를 수행한다.

1.2 개관

이 논문은 프로그램의 입력에 의존하는 성질을 추정하는 분석인 "준비된 분석 A"를 디자인하기 위한 방법을 제시한다. 이 방법은 정적 분석의 틀을 기반으로 한다. 프로그램과 무관한 성질들을 추정하도록 고안된 정적 분석은 프로그램의 입력이 나타났을때 입력에 의존하는 성질들을 안전하게 추정할 수 있는 분석으로 변환된다. 프로파일은 필요없고, 실행중에 프로그램의 상태를 조사하기 위한 코드도 필요없다.

아이디어는 단순히 정적 분석의 마지막을 프로그램의 실행시간으로 연기하는 것이다. 정적 분석을 분석하여, 프로그램의 입력에 민감한 부분들을 찾아낸다. 그리고 나서, 값을 자르는 분석을 통하여 이런 부분들이 입력에 대한 간단한 멤버쉽(membership) 테스트만으로 풀려질 수 있도록 단축시킨다. 이러한 재구성은 프로그램의 입력이 나타나기만 하면 준비된 분석이 순간적으로, 동시에 풀려질 수 있도록 준비한다.

예를 들어, 입력 α에 의존하는 다음 식(expression)의 값을 추정하는 분석을 생각해보자.

if e = 0 then 1 else 2

식의 값 X는 다음과 같은 조건부 집합 관계식이다.

$$X_e \supseteq 0 \Rightarrow X \supseteq 1$$
$$X_e \supseteq \{ n \mid n \neq 0 \} \Rightarrow X \supseteq 2.$$

조건식 e가 입력에 의존하는 식이라고 가정하자.

조건식의 값은 컴파일 시간에 알 수가 없기 때문에, 컴파일러는 if식의 값이 $\{1, 2\}$라는 결론을 내야 한다. 실행시간에는 X_e의 값을 알 수 있으므로 더 정확한 결과를 결정할 수 있다. 다음과 같이 $X_e \supseteq \{0\}$와 $X_e \supseteq \{ n \mid n \neq 0 \}$를 입력 α의 V와 W에 대한 멤버쉽 테스트로 재구성하면, 분석 결과를 더 빠르게 얻을 수 있다.

$$\alpha \in V \Rightarrow X \supseteq \{1\}$$
$$\alpha \in W \Rightarrow X \supseteq \{2\}$$
$$\text{else} \Rightarrow X \supseteq \{1, 2\}.$$

실행시간에 입력 ν이 발생하면, 이 조건들은 X_e가 계산되어질 때까지의 지연 없이 바로 참이나 거짓으로 결정된다. 따라서 입력 ν에 따른 식의 값 X가 빠르게

추정될 수 있다. V와 W는 (또 다른 정적 분석에 의해) 컴파일 시간에 추정되기 때문에 모든 값을 커버할 필요는 없다. 몇몇 입력은 $V \cup W$에 포함되지 않을 수 있고, 이런 경우는 정적 분석의 결과와 같은 정확도의 분석 결과를 낸다.

1.3 논문의 구조

2절에서는 소스 언어 L을 소개한다. 3절은 입력 변수를 가지는 L 프로그램에 대한 정적 분석을 소개한다. 이 분석으로부터, 입력에 의존하는 성질을 추정하는 동적 분석을 정의한다. 4절과 5절에서는 동적 분석의 부담을 컴파일 시간으로 옮기는 방법을 제시한다. 4절에서는 정적 분석에서 입력에 의존하는 부분을 식별하고, 입력에 무관한 부분은 컴파일 시간에 풀면서 입력에 의존하는 부분은 실행시간으로 연기하는 방법을 제시하고, 5절에서는 입력에 의존하는 부분들을 단축하는 방법을 제시한다. 우리가 제시한 분석을 적용한 예를 6절에서 보여주고, 7에서는 관련 연구들에 대해서 논한다. 8절에서 결론을 맺는다.

2. 소스 언어 L

우리가 대상으로 하는 언어는 core ML[3]과 같이 값에 의한 호출(call-by-value)을 하는 고차(higher-order)언어이다. 고차 언어란, 함수를 다른 값들과 구분하지 않고 다른 함수의 인수나 결과값으로 사용하는 언어이다. 언어에서 사용하는 식은 변수, 함수 호출식, 자료 생성(data construction), 자료값에서 인수의 추출(deconstrution) 또는 경우에 따른 분기(case)이다(그림 2). 각각의 모든 식

문법:

Expr	e	::=	$e_l \mid x \mid v \mid e\ e \mid \mathbf{con}\ \kappa\ e$	식
			$\mid\ \mathbf{decon}\ \kappa\ e \mid \mathbf{case}\ e\ \kappa\ e\ e$	
Val	v	::=	$\lambda x.e \mid \kappa(v)$	값
Con	κ			자료 생성자
Var	x, f			변수
Label	l			식의 꼬리표

의미:

$$eval(e) = e \longmapsto e' \longmapsto \cdots.$$

변형 규칙:

$$\begin{aligned}
\mathcal{E}[(\lambda x.e)\ v] &\longmapsto \mathcal{E}[[v/x]e] \\
\mathcal{E}[\mathbf{con}\ \kappa\ v] &\longmapsto \mathcal{E}[\kappa(v)] \\
\mathcal{E}[\mathbf{decon}\ \kappa\ \kappa(v)] &\longmapsto \mathcal{E}[v] \\
\mathcal{E}[\mathbf{case}\ \kappa(v)\ \kappa\ e_1\ e_2] &\longmapsto \mathcal{E}[e_1] \\
\mathcal{E}[\mathbf{case}\ \kappa'(v)\ \kappa\ e_1\ e_2] &\longmapsto \mathcal{E}[e_2] \\
\mathcal{E}[v_l] &\longmapsto \mathcal{E}[v] \quad \text{꼬리표 떼기}
\end{aligned}$$

계산 문맥:

$$\mathcal{E} ::= [\,] \mid \mathcal{E}\ e \mid v\ \mathcal{E} \mid \mathbf{con}\ \kappa\ \mathcal{E} \mid \mathbf{decon}\ \kappa\ \mathcal{E} \mid \mathbf{case}\ \mathcal{E}\ \kappa\ e\ e \mid \mathcal{E}_l$$

그림 2 소스 언어 L:문법과 의미

에 꼬리표(label)를 붙일 수 있다. 값은 함수이거나 자료(data)이다. 자료 값 $\kappa(v)$는 식 "con κ e"에 의해, 자료 생성자 κ와 식 e의 값 v로부터 만들어진다. 인수 값 v는 e의 값이 $\kappa(v)$일 때 "decon κ e"로 부터 얻을 수 있다. "case e_1 κ e_2 e_3"는 e_1의 값에 때라 e_2나 e_3로 분기한다.

L의 의미(semantics, 그림 2)는 계산 문맥(evaluation context)[4, 5] 방법을 사용하여 정의하였다. 이 계산 문맥은 값에 의한 호출을 따르는 변형(reduction) 방법을 정의한다. 문맥 $\mathcal{E}$의 구멍이 식 e로 채워질 때, $\mathcal{E}[e]$로 표기한다. 변형 규칙은 전통적인 변형 규칙을 따른다. unlabel 규칙이 꼬리표를 떼어 내는 것을 제외하면 모든 규칙들은 꼬리표를 유지한다. 치환 연산 $[v/x]e$는 e에 나타나는 모든 자유 변수(free variable) x를 v로 치환한다. 치환 연산은 x의 꼬리표들을 유지한다. 예를 들어, $[v/x]x_l = v_l$ 이다.

3. 출발점 : 집합 기반 분석

이 절에서는 집합 기반 분석의 틀[6, 7, 8, 9] 위에서 출발점이 될 정적 분석을 정의하고, 다음절에서 이 정적 분석을 이용하여 프로그램의 입력에 의존하는 성질을 분석하는 분석을 이끌어 낼 것이다.

집합 기반 분석은 집합 관계식 모으기와 풀기의 두 단계로 이루어진다. 첫번째 단계는 프로그램 식들 사이의 자료 흐름을 기술하는 관계식들을 이끌어 내는 것이고, 두번째 단계는 그런 관계식들을 만족하는 값의 집합을 찾아가는 단계이다. 결과는 집합 변수로부터 그러한 값의 집합으로의 테이블이다. 값의 집합의 크기는 무한할 수 있지만, 그것을 유한하게 기술할 수 있는 방법인 정규 트리 문법(regular tree grammar)이 존재한다.

식 e_l과 프로그램의 변수 x는 집합 변수 X_l과 X_x을 갖는다. 각각은 식의 값과 변수에 결합된(bound) 값을 나타낸다. 집합 관계식은 $X \supseteq se$의 형태를 갖는다. X는 집합 변수이고, se는 집합식이다. 이 관계식은 집합 X가 집합 se를 포함한다는 의미를 가진다. 집합식 se는 일곱 가지가 있다. 각각은 프로그램 식들과 대응된다(그림 3). 집합식의 의미는 대응되는 프로그램 식으로부터 자연스럽게 이끌어진다. 예를 들어, $\kappa\ X_1$는 v가 X_1의 원소일 때 $\kappa(v)$들의 집합을 의미한다. $app(X_1, X_2)$는 인수를 X_2로하여 함수 X_1를 호출한 호출식으로부터 반환된 값의 집합을 나타낸다.

$case(X_1, \kappa, X_2, X_3)$는 X_1의 값에 $\kappa(v)$가 있으면

집합 식의 문법:

se	::=	$\mathcal{X}_i$	집합 변수
	|	$\lambda x.e$	함수값 집합
	|	$\kappa\,\mathcal{X}_i$	자료 집합
	|	$\kappa^{-1}\,\mathcal{X}_i$	자료의 인수 집합
	|	$app(\mathcal{X}_i,\mathcal{X}_i)$	호출식에 대한 집합
	|	$case(\mathcal{X}_i,\kappa,\mathcal{X}_i,\mathcal{X}_i)$	분기문에 대한 집합
	|	$\top$	전체 집합
i	$\in$	$\{l,x \mid e_l \in Expr, x \in Var\}$	하위 식과 변수들에 대한 색인

집합 식의 의미:

$$\begin{aligned}
\mathcal{I}(\mathcal{X}_i) &\subseteq Val\\
\mathcal{I}(\top) &= Val\\
\mathcal{I}(\lambda x.e) &= \{\lambda x.e\}\\
\mathcal{I}(\kappa\,\mathcal{X}_1) &= \{\kappa(v) \mid v \in \mathcal{I}(\mathcal{X}_1)\}\\
\mathcal{I}(\kappa^{-1}\,\mathcal{X}_1) &= \{v \mid \kappa(v) \in \mathcal{I}(\mathcal{X}_1)\}\\
\mathcal{I}(app(\mathcal{X}_1,\mathcal{X}_2)) &= \{v \mid \lambda x.e \in \mathcal{I}(\mathcal{X}_1), v \in \mathcal{I}(\mathcal{X}_e),\\
&\qquad \mathcal{I}(\mathcal{X}_x) \supseteq \mathcal{I}(\mathcal{X}_2)\}\\
\mathcal{I}(case(\mathcal{X}_1,\kappa,\mathcal{X}_2,\mathcal{X}_3)) &= \{v \mid v \in \mathcal{I}(\mathcal{X}_2), \kappa(v') \in \mathcal{I}(\mathcal{X}_1)\} \cup\\
&\quad \{v \mid v \in \mathcal{I}(\mathcal{X}_3), \kappa'(v') \in \mathcal{I}(\mathcal{X}_1), \kappa' \neq \kappa\}
\end{aligned}$$

프로그램 $\wp$으로부터 관계식 C를 끌어내는 규칙 $\wp \rhd C$:

$$x_l \rhd \{\mathcal{X}_l \supseteq \mathcal{X}_x\}$$

$$\frac{e_1 \rhd C_1}{(\mathbf{con}\ \kappa\ e_1)_l \rhd \{\mathcal{X}_l \supseteq \kappa(\mathcal{X}_1)\} \cup C_1}$$

$$\frac{e_1 \rhd C_1}{(\mathbf{decon}\ \kappa\ e_1)_l \rhd \{\mathcal{X}_l \supseteq \kappa^{-1}(\mathcal{X}_1)\} \cup C_1}$$

$$\frac{e_1 \rhd C_1}{(\lambda x.e_1)_l \rhd \{\mathcal{X}_l \supseteq \lambda x.e_1\} \cup C_1}$$

$$\frac{e_1 \rhd C_1 \quad e_2 \rhd C_2}{(e_1\ e_2)_l \rhd \{\mathcal{X}_l \supseteq app(\mathcal{X}_1,\mathcal{X}_2)\} \cup C_1 \cup C_2}$$

$$\frac{e_1 \rhd C_1 \quad e_2 \rhd C_2 \quad e_3 \rhd C_3}{(\mathbf{case}\ e_1\ \kappa\ e_2\ e_3)_l \rhd \{\mathcal{X}_l \supseteq case(\mathcal{X}_1,\kappa,\mathcal{X}_2,\mathcal{X}_3)\} \cup C_1 \cup C_2 \cup C_3}$$

그림 3 집합 기반 분석을 위한 집합 간계식: 문법, 의미, 유고규칙

X_2의 값들을, X_1의 값에 $\kappa'(v)$가 있으면 X_3의 값들을 나타낸다. 집합식의 의미는 집합식들로부터 값의 집합으로의 매핑인 해석 $\mathcal{I}$로 엄밀하게 정의된다(그림 3). 모든 $X \supseteq se \in C$에 대해 $\mathcal{I}(X) \supseteq \mathcal{I}(se)$ 이면, 해석 $\mathcal{I}$를 관계식 집합 C의 모델이라고 부른다.

정적 분석은 집합 관계식들의 최소 모델로 정의될 수 있다. 이 논문에서 제안한 집합 관계식은 모든 연산들이 단조적(monotonic)이고 각 관계식의 왼쪽이 변수로만 이루어지기 때문에 모든 관계식들을 만족하는 최소모델이 존재함을 보장한다[6].

3.1 초기 관계식들을 모으기

그림 3은 프로그램 $\mathcal{S}$로부터 초기 관계식 C를 모으기 위한 규칙들을 보여준다. 이 규칙들은 프로그램을 한번만훑어봄으로써 초기 관계식을 모을 수 있다.

$$\mathcal{S} \rhd C$$

각 식으로부터 만들어지는 관계식들은 하위식들로부터 얻어진 관계식들과, 그 식 자체에 대한 관계식을 합친 것이다. 예를 들어, 함수 호출식 $e_1\ e_2$에 대해 만들어지는

$$\frac{\mathcal{X} \supseteq \kappa^{-1}\mathcal{Y} \quad \mathcal{Y} \supseteq \kappa\mathcal{Z}}{\mathcal{X} \supseteq \mathcal{Z}} \qquad \frac{\mathcal{X} \supseteq case(\mathcal{X}_1,\kappa,\mathcal{X}_2,\mathcal{X}_3) \quad \mathcal{X}_1 \supseteq \kappa\mathcal{Y}}{\mathcal{X} \supseteq \mathcal{X}_2}$$

$$\frac{\mathcal{X} \supseteq case(\mathcal{X}_1,\kappa,\mathcal{X}_2,\mathcal{X}_3) \quad \mathcal{X}_1 \supseteq \kappa'\mathcal{Y} \quad \kappa' \neq \kappa}{\mathcal{X} \supseteq \mathcal{X}_3}$$

$$\frac{\mathcal{X} \supseteq app(\mathcal{X}_1,\mathcal{X}_2) \quad \mathcal{X}_1 \supseteq \lambda x.e_3}{\mathcal{X} \supseteq \mathcal{X}_3 \quad \mathcal{X}_x \supseteq \mathcal{X}_2} \qquad \frac{\mathcal{X} \supseteq \mathcal{Y} \quad \mathcal{Y} \supseteq ae}{\mathcal{X} \supseteq ae}$$

그림 4 집합 관계식을 풀기 위한 규칙 S

관계식들은 두개의 하위식 e_1, e_2로부터 만들어진 두 관계식들 C_1, C_2와 $X_l \supseteq app(X_1,X_2)$를 합친 것이다. $X_l \supseteq app(X_1,X_2)$의 의미는 X_l이 가질 수 있는 값은 X_2(e_2에 대한 집합 변수)를 인수로 한 X_1(e_1에 대한 집합 변수)이 가지는 함수의 결과값을 포함한다는 의미이다.

3.2 관계식들을 풀기

관계식을 푸는 단계는 초기 관계식 집합 C를 그림 4의 규칙에 따라서 더 이상 풀 것이 없을 때까지 푸는 것이다. 직관적으로, 그림 4의 규칙들은 프로그램의 모든 가능한 자료 흐름 경로를 따라서 값을 전파시킨다. 각 전파 규칙들은 복합(compound) 집합식을 더 작은 형태로 분해시킨다. 이 과정은 식들 사이로 값이 흐르는 단계를 요약(approximate)한 것이다. 호출식에 대한 규칙을 생각해 보자. $X \supseteq app(X_1,X_2)$:

$$\frac{X \supseteq app(X_1,X_2) \qquad X_1 \supseteq \lambda x.e_3}{X \supseteq X_3 \qquad X_x \supseteq X_2}$$

이 규칙은 호출될 함수가 $\lambda x.e_3$일 경우 $X \supseteq X_3$를 생성한다. 그리고, 실인수 X_2가 형식인수 x에 결합되는 것을 흉내내기 위해 $X_x \supseteq X_2$를 추가한다. ($X_1 \supseteq \lambda x.e_3$와 같이 복합 집합식을 분해하는 방아쇠 역할을 하는 관계식을 "점화 관계식"이라고 부른다.) 다른 규칙들은 집합식에 대응하는 의미들로부터 쉽게 이끌어진다.

규칙 R들을 사용하여 c가 A로 부터 유도될 수 있으면 $A \vdash_R c$라고 쓰고, 규칙 R에 대한 A의 닫힌집합(closure) $\{c \mid A \vdash_R c\}$을 $R^*(A)$라고 쓴다.[1]

집합 관계식의 집합 C에 대해, $Vars(C)$는 C에 있는 집합 변수들의 집합을, $C(X)$는 $\{X \supseteq se \mid X \supseteq se \in C\}$를 나타낸다.

1) 이 닫힌집합은 다음과 같은 최소 고정점(least fixpoint)으로 정의될 수도 있다. $lfp\,(\lambda X.A \cup \{c \mid X \vdash_R c\})$.

$S^*(C)$에 있는 관계식 집합 사이에서, 완전히 분해된 관계식 집합 $atom(S^*(C))$는 C의 최소 모델을 이룬다. 그런 관계식들을 "핵(atomic)"이라 부른다. 핵 관계식(atomic constraint)은 관계식의 오른쪽 ae(atomic expression)가 다음의 값 집합일 때 $X \supseteq ae$의 형태이다.

$$ae ::= \lambda x.e \mid xX \mid \top .$$

핵 관계식의 집합 $G = \{X_1 \supseteq ae_1, \ldots, X_n \supseteq ae_n\}$는 G의 문법적 해석(정규 트리 문법, 집합 변수는 비단말(non-terminal)) $X_1 ::= ae_1, \ldots, X_n ::= ae_n$에 의해 생성되는 문장(sentence)들의 집합을 의미한다. $[\![G]\!]$는 그런 문장들의 집합이다.

정리1. $\wp$에 자유 변수가 없고, 집합 관계식 C가 $\wp \rhd C$에 의해 도출되었다고 하면, C의 최소 모델은 $\{X \mapsto [\![atom(S^*(C)(X))]\!] \mid X \in Vars(C)\}$ 이다.

증명. [6, 7] □

정의1.(sba) 자유변수가 없는 프로그램 $\wp$에 대한 집합 관계식 C가 $\wp \rhd C$에 의해 도출되었다고 하면, $sba(\wp)(l) \leftrightarrows [\![atom(S^*(C)(X_l))]\!]$. 집합 기반 분석 $sba(\wp)$는 프로그램 값들을 안전하게 예측한 것(safe approximation)이다.

정리2. (sba의 안전성) 자유변수가 없는 프로그램 $\wp$의 관계식이 $\wp \rhd C$라 하자. 만약 $\wp \mapsto^* \mathcal{E}[v_l]$ 이라면 $v \in sba(\wp)(l)$.

증명. [6, 7]의 증명 방법을 따라서 증명가능. [7]의 집합 기반 동적 의미 구조를 정리 1을 증명하기 위한 중간 단계로 사용한다. □

3.3 입력(자유 변수)이 있을 경우의 분석

지금까지 우리는 분석될 프로그램이 입력(자유) 변수를 가지고 있지 않다고 가정했다. 이제부터는 모든 프로그램 $\wp$이 하나의 입력 변수 α를 가진다고 가정하겠다.

3.3.1 정적 분석: 입력에 무관한 불변 성질의 예측

입력 변수 α를 가지는 프로그램 $\wp$에 대한 초기 관계식들의 집합 $C(\wp \rhd C)$는 X_α에 대해서는 아무런 관계식을 가지고 있지 않다. 따라서, X_α를 사용하는 집합 관계식들은 풀려질 수 없다.

그러나, C를 사용하여 입력에 무관한 성질(불변 성질)들을 추정하는 정적 분석을 얻는 것은 쉽다. 입력 변수 α에 대한 또 다른 관계식 $X_\alpha \supseteq \top$로부터 푸는 과정 $S^*(\cdot)$를 시작한다. $\top$은 전체 값들의 집합 Val을 나타낸다. 분석 과정은 다음과 같다.

```
let  ℘ ▷ C                  초기 관계식 (컴파일시간)
in   S*(C ∪ X_α ⊇ ⊤ )       관계식 풀기 (컴파일시간)
```

입력에 대해서는 아무런 정보도 없다(전체 집합)고 가정했기 때문에, 위의 분석은 입력에는 무관한 성질을 예측하게 된다.

정의2. ($sba_\top$) 프로그램 $\wp$가 입력 α를 가지고, 그에 대한 초기 관계식이 $\wp \rhd C$에 의해 도출되었다고 하자.

$sba_\top(\wp)(l) \leftrightarrows [\![atom(S^*(C \cup \{X_\alpha \supseteq \top\})(X_l))]\!]$.

정리3. ($sba_\top$의 안전성) 모든 $\nu \in Val$에 대해, $[\nu/\alpha]\wp \mapsto^* \mathcal{E}[v_l]$ 이면 $v \in sba_\top(\wp)(l)$이다.

증명. $\forall \nu \in Val$, $sba([\nu/\alpha]\wp)(l) \subseteq sba_\top(\wp)(l)$이기 때문에 정리 2에 의해 증명 가능. sba와 $sba_\top$는 같은 규칙을 사용하고, 이 규칙은 단조(monotonic)적이다. $sba([\nu/\alpha]\wp)$는 $X_\alpha \supseteq \nu$를 가지는 반면, $sba_\top(\wp)$는 더 큰 $X_\alpha \supseteq \top$를 가지는 것을 제외하고는 두 개의 초기 관계식이 같기 때문에 증명 가능하다.

3.3.2 동적 분석: 입력에 따라 변하는 성질들의 예측

입력에 의존하는 성질을 분석을 하기 위해서는, 푸는 단계을 실행시간으로 미루어야 한다. 실행시간에는 입력에 대한 관계식 $X_\alpha \supseteq \top$에서 $\top$을 실제 입력값 ν으로 대체할 수 있다:

```
let  ℘ ▷ C                  초기 관계식 (컴파일 시간)
in   S*(C ∪ X_α ⊇ ν)        관계식 풀기 (실행 시간)
```

위의 과정은 입력값 ν에 특화된 성질을 예측한다.

정의 3. (sba_ν) 프로그램 $\wp$이 입력 α를 가지고, 초기 관계식이 $\wp \rhd C$에 의해 도출되었다고 하면, $sba_\nu(\wp)(l) \leftrightarrows [\![atom(S^*(C \cup X_\alpha \supseteq \nu)(X_l))]\!]$.

정리 4. (sba_ν의 안전성) $[\nu/\alpha]\wp \mapsto^* \mathcal{E}[v_l]$ 이면 $v \in sba_\nu(\wp)(l)$이다.

증명. $sba_\nu(\wp) = sba([\nu/\alpha]\wp)$이기 때문에, 정리 2에 의해 증명 가능.

음절에서는 이 동적 분석을 바탕으로 동적 관계식을 미루고, 이 부분을 단축시킨 분석을 유도한다. 유도된 분석은 프로그램의 입력이 나타나면 안전하면서도 빠르게 결과를 낼 것이다.

4. 파생 분석 I: 동적 관계식을 미루자

동적 분석 $S^*(C \cup X_\alpha \supseteq \nu)$을 계산하는 과정 중 많은 부분이 컴파일 시간에 수행될 수 있다. 관계식 $X_\alpha \supseteq \nu$이 프로그램의 실행시간이 되어야 제공된다고 해도, 컴파일 시간에 C중에서 입력에 무관한 관계식들은 풀고 나머지를 실행시간으로 미룰 수 있다.

프로그램의 입력에 의존하는 관계식 $X \supseteq se_\square$은 집합식 $se_\square$가 입력에 의존한다(아래첨자 $\square$)고 표시된 것을 제외하고는 보통의 관계식들과 같은 형태를 가진다. 입력에 의존하는 집합 식 $se_\square$는 컴파일 시간에는 분해될 수 없다. 입력 의존적 집합식은 다음의 네 가지가 있다.

$$se_\square ::= X_\alpha \mid \kappa_\square^{-1} \mid app_\square(X_1, X_2) \mid case_\square(X_1, \kappa, X_2, X_3)$$

입력 변수에 대한 집합 변수 X_α는 입력에 의존적이다. 복합 집합식들은 점화 관계식이 입력에 의존적이면 입력에 의존적이다.

입력에 의존적이 동적 관계식들을 식별하는 것은 간단하다. 관계식에 푸는 규칙 S를 적용할 때, 우리는 집합 관계식 $X \supseteq X_\alpha$와 같이 입력 집합 변수를 오른쪽에 가지는 관계식들로부터 출발하여 입력에 대한 의존성도 함께 전파하게 된다. S에 있는 푸는 규칙들은 점화 관계식이 주어지면, 복합 집합 관계식들을 더 작은 것으로 분해한다는 것에 주목하자. 따라서, 점화 관계식이 입력에 의존적이면 복합 집합 관계식도 입력 의존적이 된다. 예를 들어, 다음의 규칙을 생각해 보자.

$$\frac{X \supseteq app(X_1, X_2) \quad X_1 \supseteq \lambda x.e}{X \supseteq X_e \quad X_x \supseteq X_2}.$$

점화 관계식 $X_1 \supseteq \lambda x.e$ 는 $X \supseteq app(X_1, X_2)$가 분해되게 해 준다. 만약 이 점화 관계식이 입력에 의존적이라면 우리는 다음과 같이 $app(X_1, X_2)$도 입력에 의존적이라고 표시할 수 있다.

$$\frac{X \supseteq app(X_1, X_2) \quad X_1 \supseteq se_\square}{X \supseteq app_\square(X_1, X_2)}.$$

다른 경우들도 비슷하게 정의된다.

그림 5는 입력에 의존하는 관계식들을 식별하는 규칙을 나타낸다. 각각의 규칙은 S에서 대응되는 규칙들과 나란히 놓여져 있다. 마지막 규칙은 입력에 대한 의존성을 전달하는 규칙이다.

$$\frac{\mathcal{X} \supseteq app(\mathcal{X}_1, \mathcal{X}_2) \quad \mathcal{X}_1 \supseteq se_\square}{\mathcal{X} \supseteq app_\square(\mathcal{X}_1, \mathcal{X}_2)}$$

$$\frac{\mathcal{X} \supseteq case(\mathcal{X}_1, \kappa, \mathcal{X}_2, \mathcal{X}_3) \quad \mathcal{X}_1 \supseteq se_\square}{\mathcal{X} \supseteq case_\square(\mathcal{X}_1, \kappa, \mathcal{X}_2, \mathcal{X}_3)}$$

$$\frac{\mathcal{X} \supseteq \kappa^{-1}\mathcal{Y} \quad \mathcal{Y} \supseteq se_\square}{\mathcal{X} \supseteq \kappa_\square^{-1}\mathcal{Y}}$$

$$\frac{\mathcal{X} \supseteq \mathcal{Y} \quad \mathcal{Y} \supseteq se_\square}{\mathcal{X} \supseteq se_\square}$$

그림 1 입력에 의존하는 관계식을 찾아내고 연기시키는 규칙 D

모든 과정은 다음과 같이 정의된다:

let $\mathcal{S} \triangleright C$ 초기 관계식 (컴파일시간)

$C' = (S \cup D)^*(C)$ 부분적으로 풀기(S), 연기하기(D) (컴파일시간)

in $S_\square^*(|C'| \cup X_\alpha \supseteq \nu)$ 연기된 관계식 풀기 (실행시간)

먼저, 프로그램 $\mathcal{S}$로부터 초기 관계식 집합 C를 모은다(그림 3의 $\mathcal{S} \triangleright C$). 그리고 나서, 입력에 무관한 관계식들을 풀기 위해 규칙 S를 C에 적용하고, 동시에 규칙 D를 적용하여 입력에 의존하는 관계식들을 식별하여 미룬다. 이 두 과정은 컴파일 시간에 수행될 수 있다. 결과로 만들어진 관계식 집합 C'은 실행시간에 입력값 ν과 함께 풀어지게 된다. 실행시간에 우리는 규칙 $S_\square$를 적용한다. 규칙 $S_\square$는 $\square$로 표시된 입력 의존적 집합식들을 다룬다는 것 외에는 규칙 S와 같다. 관계식 집합 $|C'|$는 C'에 있는 관계식들 중 완전히 풀려진 것 또는 연기된 관계식들을 모은 집합이다.

$$|C'| = \{X \supseteq ae \mid X \supseteq ae \in C'\} \cup \{X \supseteq se_\square \mid X \supseteq se_\square \in C'\}$$

정의 4. (연기된 분석 $dsba$) 프로그램 $\mathcal{S}$가 입력 α를 가지고, 초기관계식이 $\mathcal{S} \triangleright C$에 의해 도출되었으며, ν은 실행시간에 주어진 입력이라고 하자.

$C' = (S \cup D)^*(C)$이고,

$C'' = S_\square^*(|C'| \cup X_\alpha \supseteq \nu)$일때, 연기된 분석 $dsba_\nu(\mathcal{S})(l)$ 쁘 $[\![atom(C''(X_l))]\!]$ 이다.

정리 5. ($dsba$의 안전성) $[\nu/\alpha]\mathcal{S} \mapsto^* \mathcal{E}[v_l]$ 이면 $v \in dsba_\nu(\mathcal{S})(l)$ 이다.

증명. $dsba_\nu(\mathcal{S}) = sba_\nu(\mathcal{S})$ 이므로 증명 가능. 두 과

정의 다른 점은 $dsba_\nu(\mathcal{S})$는 입력에 의존하는 관계식들을 미루어 실행시간에 푼다는 것 뿐이다. 여기서, 실행시간에 푸는 규칙은 $sba_\nu(\mathcal{S})$에서 사용하는 규칙과 정확하게 일치하며, 연기된 관계식들의 의미도 앞에서와 같다. 따라서, 연기된 집합 기반 분석의 결과도 $sba_\nu(\mathcal{S})$의 결과와 일치한다. □

연기된 관계식 $|C'|$를 풀기위한 실행시간 과정 $S^*_\nu(|C'| \cup X_a \supseteq \nu)$에도 약간의 실행시간 지연이 남아 있다. 연기된 관계식의 점화 관계식은 그 점화 관계식의 점화 관계식이 풀어져야 풀려질 수 있고, 점화 관계식의 점화 관계식은 점화 관계식의 점화 관계식의 점화 관계식이 풀어져야 풀려질 수 있다. 이렇게 $X_a \supseteq \nu$으로부터 점화 관계식이 풀려질 때까지 값이 전파된 후에야 풀고자 하는 관계식을 풀 수 있는 것이다.
다음절에서는 이런 지연을 피하기 위한 관계식 변환에 대해 설명한다.

5. 파생 분석 II: 값을 자르는 분석을 통해 동적 관계식을 준비하자

만약 점화 관계식이 $se_\square$의 하위구조에 관한 것이 아니라 프로그램의 입력에 관한 것이라면, 연기된 관계식 $X \supseteq se_\square$을 푸는 시간은 단축될 수 있을 것이다. 예를 들어, 다음의 $X \supseteq case_\square(X_1, x, X_2, X_3)$를 푸는 규칙을 생각해 보자.

$$\frac{X \supseteq case_\square(X_1, x, X_2, X_3) \qquad X_1 \supseteq xY}{X \supseteq X_2}$$

점화식 $X_1 \supseteq xY$는 프로그램내의 식 X_1에 대한 것이기 때문에, X_1이 xY로 풀려질 때까지 기다려야 한다.
변환 규칙의 점화 관계식을 다음과 같이 입력에 대한 조건으로 바꾸면 이런 지연이 없어진다. "만약, 프로그램의 입력이 V에 포함되면 case식 X는 X_2의 값을 가진다." V에 관한 조건은 다음과 같다. V에 속한 모든 프로그램의 입력에 대해 e_1은 x로 시작되는 값으로 계산되어진다.

"값을 자르는 분석"은 그러한 집합 V를 예측한다. 값을 자르는 분석이란 다음의 질문에 대한 답을 얻기 위한 정적분석이다: 프로그램내의 한 식의 값이 집합 V에 포함되기 위한 프로그램의 입력의 조건은 무엇인가?

5.1 값을 자르는 분석

$$\frac{\lambda x.e \in p}{\lambda x.e \to \lambda x.e}$$

$$\frac{e_1 \to \lambda x.e_1' \quad e_2 \to \lambda y.e_2' \quad e_1\ e_2 \in p}{x \to \lambda y.e_2'}$$

$$\frac{e_1 \to \lambda x.e_1' \quad e_1' \to \lambda y.e_2'}{e_1\ e_2 \to \lambda y.e_2'} \qquad \frac{\lambda x.e \in p}{\mathbf{decon}\ \kappa\ e_1 \to \lambda x.e}$$

$$\frac{e_2 \to \lambda x.e}{\mathbf{case}\ e_1\ \kappa\ e_2\ e_3 \to \lambda x.e} \qquad \frac{e_3 \to \lambda x.e}{\mathbf{case}\ e_1\ \kappa\ e_2\ e_3 \to \lambda x.e}$$

그림 1 안전한 함수 호출 그래프 예측 규칙 $L_\mathcal{S}$

일반적으로, 자르는 분석(slicing) [10]이라 하면 기준이 되는 식에 관련된 프로그램의 부분(프로그램의 식 또는 변수)들을 고르는 방법이다.
값을 자르는 분석은 프로그램의 일부분을 자르는 것이 아니라, 프로그램내의 식이 가질 수 있는 값을 자르는 분석이다. 자르는 기준인 $e \subseteq V$는 식 e의 값이 V에 속하기 위한 조건이다. 자른 결과인 $SVS_\mathcal{S}(e \subseteq V)$는 프로그램 $\mathcal{S}$의 식들로부터 그 값의 집합으로의 매핑이다. 각 값의 집합들은 프로그램이 e의 값을 V에 속하게끔 만들어주기 위해 필요한 값들이다. 더 엄밀하게 말하면, 자른 결과 $SVS_\mathcal{S}(e \subseteq V)(a)$에 속한 모든 입력에 대해서 프로그램이 e의 값을 계산해 낸다면, 그 값은 V에 속해야 한다.
예 1 다음의 프로그램 $\mathcal{S}$을 생각해보자.

```
datatype t = A | B | C
case x of
  A => B
| _ => x
```

입력 x가 A또는 B일 때, 프로그램은 B를, 입력이 C일때는 C를 결과로 낸다. 자르는 기준이 $\mathcal{S} \subseteq$ {B}일 때, 값을 자르는 분석 $SVS_\mathcal{S}(\mathcal{S} \subseteq \{B\})(x)$는 A, B의 부분집합이 되어야 한다. 만약, 기준이 $\mathcal{S} \subseteq$ {C}라면, $SVS_\mathcal{S}(\mathcal{S} \subseteq \{C\})(x) \subseteq \{C\}$가 되어야 한다.
이 논문에서 우리는 값을 자르는 분석을 집합 관계식의 틀 위에서 정의하려고 한다. 집합 관계식은 다음의 형태를 가진다.

$$se_l \subseteq se_r$$

이것의 의미는 집합 se_l는 집합 se_r에 포함되어야 한다는 것이다.
그림 7은 관계식의 정의이다. 관계식 시스템은 초기 관계식을 이끌어 내기 위한 세 개의 규칙들과 슬라이싱 전달

규칙 B로 이루어진다.[2] 이 세 개의 규칙들은 한꺼번에 적용하는 것이 아니라 따로 따로 적용하게 된다.

규칙 B는 관계식의 의미에서부터 직관적으로 만들어진다. 초기 관계식을 이끌어내기 위한 규칙 ($\triangleright_i$)도 역시 직관적이다:

자르는 관계식 $se_l \subseteq se_r$.

자르는 집합 식의 문법:

se_l				se_r		
	::=	$\mathcal{X}_i$	집합 변수	::=	$\mathcal{X}_i$	집합 변수
	|	$\lambda x.e$	함수	|	$\kappa\,\mathcal{X}_i$	자료 집합
	|	$\kappa\,\mathcal{X}_i$	자료 집합	|	$\kappa\,\top$	자료 집합
				|	$\overline{\kappa}\,\top$	자료 집합
				|	$Lam_p(e)$	함수 집합
				|	$\top$	전체 집합

$i \in \{l, x \mid e_l \in Expr, x \in Var\}$ 하위 식과 변수들에 대한 색인

$ae ::= \kappa\,\mathcal{X}_i \mid \kappa\,\top \mid \overline{\kappa}\,\top \mid \top \mid Lam_p(e)$ 핵 집합 식

자르는 집합식의 의미:

$$\begin{array}{rcl@{\qquad}rcl} \mathcal{I}(\mathcal{X}_i) & \subseteq & Val & & & \\ \mathcal{I}(\top) & = & Val & \mathcal{I}(\lambda x.e) & = & \{\lambda x.e\} \\ \mathcal{I}(Lam_p(e)) & = & Lam_p(e) & \mathcal{I}(\kappa\,\mathcal{X}) & = & \{\kappa(v) \mid v \in \mathcal{I}(\mathcal{X})\} \\ \mathcal{I}(\kappa\,\top) & = & \{\kappa(v) \mid v \in Val\} & & & \\ \mathcal{I}(\overline{\kappa}\,\top) & = & \{\kappa'(v) \mid \kappa' \neq \kappa, v \in Val\} & & & \end{array}$$

초기의 자르는 관계식을 유도하기 위한 세개의 규칙 $\triangleright_1$, $\triangleright_2$, $\triangleright_3$:

$$x_l \triangleright_i \{\mathcal{X}_x \subseteq \mathcal{X}_l\} \qquad \frac{e_1 \triangleright_i \mathcal{C}_1}{(\lambda x.e_1)_l \triangleright_i \{\lambda x.e_1 \subseteq \mathcal{X}_l\} \cup \mathcal{C}_1}$$

$$\frac{e_1 \triangleright_i \mathcal{C}_1}{(\mathbf{con}\ \kappa\ e_1)_l \triangleright_i \{\kappa\,\mathcal{X}_1 \subseteq \mathcal{X}_l\} \cup \mathcal{C}_1}$$

$$\frac{e_1 \triangleright_i \mathcal{C}_1}{(\mathbf{decon}\ \kappa\ e_1)_l \triangleright_i \{\mathcal{X}_1 \subseteq \kappa\,\mathcal{X}_l\} \cup \mathcal{C}_1}$$

$$\frac{e_1 \triangleright_i \mathcal{C}_1 \quad e_2 \triangleright_i \mathcal{C}_2 \quad Lam_p(e_1) \ni \lambda x.e_3}{(e_1\ e_2)_l \triangleright_i \{\mathcal{X}_1 \subseteq Lam_p(e_1), \mathcal{X}_2 \subseteq \mathcal{X}_x, \mathcal{X}_3 \subseteq \mathcal{X}_l\} \cup \mathcal{C}_1 \cup \mathcal{C}_2}$$

$$\frac{e_1 \triangleright_1 \mathcal{C}_1 \quad e_2 \triangleright_1 \mathcal{C}_2 \quad e_3 \triangleright_1 \mathcal{C}_3}{(\mathbf{case}\ e_1\ \kappa\ e_2\ e_3)_l \triangleright_1 \{\mathcal{X}_1 \subseteq \kappa\,\top, \mathcal{X}_2 \subseteq \mathcal{X}_l\} \cup \mathcal{C}_1 \cup \mathcal{C}_2 \cup \mathcal{C}_3}$$

$$\frac{e_1 \triangleright_2 \mathcal{C}_1 \quad e_2 \triangleright_2 \mathcal{C}_2 \quad e_3 \triangleright_2 \mathcal{C}_3}{(\mathbf{case}\ e_1\ \kappa\ e_2\ e_3)_l \triangleright_2 \{\mathcal{X}_1 \subseteq \overline{\kappa}\,\top, \mathcal{X}_3 \subseteq \mathcal{X}_l\} \cup \mathcal{C}_1 \cup \mathcal{C}_2 \cup \mathcal{C}_3}$$

$$\frac{e_1 \triangleright_3 \mathcal{C}_1 \quad e_2 \triangleright_3 \mathcal{C}_2 \quad e_3 \triangleright_3 \mathcal{C}_3}{(\mathbf{case}\ e_1\ \kappa\ e_2\ e_3)_l \triangleright_3 \{\mathcal{X}_2 \subseteq \mathcal{X}_l, \mathcal{X}_3 \subseteq \mathcal{X}_l\} \cup \mathcal{C}_1 \cup \mathcal{C}_2 \cup \mathcal{C}_3}$$

슬라이싱 전달을 위한 규칙 B:

$$\frac{\mathcal{X} \subseteq \mathcal{Y} \quad \mathcal{Y} \subseteq ae}{\mathcal{X} \subseteq ae} \qquad \frac{\kappa\,\mathcal{X} \subseteq \mathcal{Y} \quad \mathcal{Y} \subseteq \kappa\,\mathcal{Z}}{\mathcal{X} \subseteq \mathcal{Z}} \qquad \frac{\kappa\,\mathcal{X} \subseteq \mathcal{Y} \quad \mathcal{Y} \subseteq \kappa\,\top}{\mathcal{X} \subseteq \top}$$

$$\frac{\kappa\,\mathcal{X} \subseteq \mathcal{Y} \quad \mathcal{Y} \subseteq \overline{\kappa'}\,\top \quad \kappa' \neq \kappa}{\mathcal{X} \subseteq \top} \qquad \frac{\kappa\,\mathcal{X} \subseteq \mathcal{Y} \quad \mathcal{Y} \subseteq \top}{\mathcal{X} \subseteq \top}$$

그림 7 값을 자르는 분석을 위한 관계식: 문법, 의미, 자르는 규칙

• (con x e_1)$_l$이 1 X_l로 잘려진다면, 하위식 e_1을 다음과 같이 잘라야 한다: $xX_1 \subseteq X_l$. decon의 경우에도 마찬가지이다.

• 호출식 $(e_1\ e_2)_l$이 X_l로 잘려진다면, e_1, e_2와 호출된 함수에 대한 세 개의 집합을 잘라야 한다: 함수식 e_1에 대한 집합은 e_1에서 호출 가능한 함수들의 집합인 $Lam_{\mathcal{S}}(e_1)$로 잘라져야 한다. $\lambda x.e_3 \in Lam_{\mathcal{S}}(e_1)$에 속한 모든 함수들에 대해, 인수식 e_2는 x의 값으로 잘려져야 하고, 호출된 함수는 호출식 X_l로 잘라져야 한다.

$Lam_{\mathcal{S}}$은 모든 입력에 대해서 각각의 식 e에서 호출되어질 수 있는 함수들을 안전하게 모은 테이블이다. 그림 6의 규칙 L에 의해 테이블이 결정된다:

$$Lam_{\mathcal{S}}(e) = \{\lambda x.e' \mid e \rightarrow \lambda x.e' \in L^*_{\mathcal{S}}(\mathcal{S})\}.$$

$e \rightarrow \lambda x.e'$는 e가 함수 $\lambda x.e'$으로 계산되어질 수 있음을 의미한다. $Lam_{\mathcal{S}}$ 테이블의 안전성은 쉽게 증명될 수 있다.

• (case e_1 x e_2 e_3)$_l$이 X_l로 잘려진다면, 자르는 방법은 세 가지가 있을 수 있다. 이 규칙들은 e_2만 수행될 경우, e_3만 수행될 경우, e_2, e_3가 모두 수행될 경우의 세 가지 가능한 수행 흐름에 따라 달라진다. 각각의 상황에 따라서 자르는 규칙을 서로 다르게 정의할 필요가 있다. 첫 번째 상황에 대해서는, e_1이 $x\,\top$ ($\top$은 전체 집합을 의미)으로, e_2는 X_l(case식)으로 잘라야 한다. 첫 번째 상황에 대해서는, e_1이 $\overline{x}\,\top$ (x로 시작하지 않는 값)으로, e_3는 X_l(case식)으로 잘라야 한다. 세 번째 상황에 대해서는, e_1에 대해서는 아무 것도 자르지 않아야 하고, e_2, e_3 모두 X_l로 잘라야 한다.

값을 자르는 분석 $SVS_{\mathcal{S}}(e \subseteq V)$의 과정을 다음과 같이 정의한다:[3]

let $\mathcal{S} \triangleright_i C_i$, for $i = 1, 2, 3$ 세 개의 초기관계식

in $\bigvee_{i=1}^{3} B^*(C_i \cup \{X_e \subseteq V\})$ 세개의자른결과들의 논리합

규칙 B에 닫힌 핵 관계식 B는 $C_i \cup \{X_e \subseteq V\}$의 최대 모델 (gm)이다. 우리가 제안한 관계식 시스템은 코데피니트(co-definite) 관계식이기 때문에, 최대 모델이 항상 존재한다[11].

정리 6. 프로그램 $\mathcal{S}$가 입력변수 α를 가지고, e는 $\mathcal{S}$의 하위식이며, V는 값의 집합이고, $\mathcal{S} \triangleright_i C_i$에 의해 초기 관계식들이 만들어진다고 하면,

2) "backward"의 의미에서 B로 이름지었다.

3) 프로그램에 n개의 case식이 있으면, 3^n개의 초기 관계식을 만들어 내야하지만, 비용과 정확성의 균형을 위해서 3개의 초기 관계식만을 만들어 낸다.

$\| atom(B^*(C_i \cup \{X_e \subseteq V\})) \|$ 는 $C_i \cup \{X_e \subseteq V\}$의 최대 모델(greates model)이다.

증명. $C' = C_i \cup \{X_e \subseteq V\}$라 하자. 먼저 $gm(C') = gm(B^*(C'))$ 임을 B가 $gm(C')$을 유지하는 관계식들만 추가한다는 것을 이용해서 보일 수 있다. 그 후 $gm(B^*(C')) = gm(atom(B^*(C')))$ 임을 보인다. A.1 절에서 자세하게 증명하였다.□

핵 관계식은 관계식 집합식 ae가 핵 집합식인 $X \subseteq ae$다.

값을 자르는 분석은 세개의 관계식들에 대한 최대 모델들의 집합이다:

정의 5. (값을 자르는 분석 $SVS_{\mathcal{E}}(e \subseteq V)$) 프로그램 $\mathcal{E}$가 입력 변수 α, 하위 식 e를 가지고, $i = 1,2,3$에 대해 $\mathcal{E} \rhd_i C_i$를 통해서 초기 관계식 C_i를 이끌어 냈다고 하면, 값을 자르는 분석은 다음과 같이 정의된다.

$$SVS_{\mathcal{E}}(e \subseteq V) \triangleq$$
$$\{\| atom(B^*(C_i \cup \{X_e \subseteq V\})) \| \mid I = 1,2,3\}.$$

다음절에서, 우리는 $SVS_{\mathcal{E}}(e \subseteq V)(\alpha)$를 $\{v \mid v \in \Sigma(\alpha), \Sigma \in SVS_{\mathcal{E}}(e \subseteq V)\}$의 의미로 사용한다.

이렇게 정의된 값을 자르는 분석은 우리가 예상한 다음의 성질을 만족한다:

정리 7. ($SVS_{\mathcal{E}}(e \subseteq V)$의 정확성) 프로그램 $\mathcal{E}$의 입력 변수가 α, 하위식 $e \in \mathcal{E}$라 하자.
$\forall v \in \Sigma(\alpha)([v/\alpha]\mathcal{E} \mapsto^* \mathcal{E}[v_e] \Rightarrow v \in V)$

$$\Sigma \in SVS_{\mathcal{E}}(e \subseteq V) \text{이면 이다.}$$

증명. 집합 기반 동적 의미구조 [6, 7] $\Sigma \vdash e \rightsquigarrow v$ 를 중간 단계로 사용하여 증명한다. 먼저, $\Sigma \vdash e \rightsquigarrow v \Rightarrow v \in \Sigma(e)$이고, Σ가 $[v/\alpha]\mathcal{E}$에 대해 안전한 집합 환경이며, $\Sigma(e) \supseteq V$임을 증명한다. 그 후, $[v/\alpha]\mathcal{E} \mapsto^* \mathcal{E}[v_e]$가 $\Sigma \vdash e \rightsquigarrow v$를 의미하고, 따라서 $v \in \Sigma(e)$를 의미하게 되고, 결국 $v \in V$를 의미하게 됨을 보인다. A.1 절에서 자세한 증명을 하였다.

5.2 동적 관계식의 단축

값을 자르는 분석을 이용하여, 입력에 의존적인 관계식을 간단한 원소 테스트를 통해서 풀 수 있다. 즉, 프로그램의 수행 중 입력이 주어지면, 그 입력이 어떤 집합의 원소인지 아닌 지의 테스트만 통해서 동적 관계식들이 즉각적으로 동시에 풀려질 수 있다.

경우식에 대한 동적 관계식 $X \supseteq case_{\square}(X_1, x, X_2, X_3)$을 생각해 보자. 점화 관계식은 $X_1 \supseteq x\,Y$이거나 $X_1 \supseteq x'\,Y (x' \neq x)$이다. V_1에 속한 입력은 X_1을 x로 시작하는 값으로 만들어 주고, V_2에 속한 입력은 X_1을 x'으로 시작하는 값으로 만들어 준다고 가정하자. 이런 집합 V_1, V_2는 값을 자르는 분석의 결과가 될 수 있다:

$$V_1 = SVS_{\mathcal{E}}(X_1 \supseteq x\top)(\alpha)$$
$$V_2 = SVS_{\mathcal{E}}(X_1 \supseteq \overline{x}\top)(\alpha).$$

이 두 집합을 이용하여, 동적 분석 $X \supseteq case_{\square}(X_1, x, X_2, X_3)$은 다음과 같은 세 가지의 관계식으로 수정되어 진다.

$$\frac{\mathcal{X} \supseteq app(\mathcal{X}_1, \mathcal{X}_2) \quad \mathcal{X}_1 \supseteq se_{\square}}{\mathcal{X} \supseteq app_{\square}(\mathcal{X}_1, \mathcal{X}_2)}$$

$$\frac{\begin{array}{c}\mathcal{X} \supseteq case(\mathcal{X}_1, \kappa, \mathcal{X}_2, \mathcal{X}_3) \quad \mathcal{X}_1 \supseteq se_{\square} \\ V_1 = \mathrm{SVS}_{\mathcal{E}}(\mathcal{X}_1 \subseteq \kappa\top)(\alpha) \\ V_2 = \mathrm{SVS}_{\mathcal{E}}(\mathcal{X}_1 \subseteq \overline{\kappa}\top)(\alpha)\end{array}}{\begin{array}{c}\mathcal{X} \supseteq if_{\square}(V_1, \mathcal{X}_2) \quad \mathcal{X} \supseteq if_{\square}(V_2, \mathcal{X}_3) \\ \mathcal{X} \supseteq if_{\square}(V_1 \cup V_2, \mathcal{X}_2 \cup \mathcal{X}_3)\end{array}}$$

$$\frac{\mathcal{X} \supseteq \kappa^{-1}\mathcal{Y} \quad \mathcal{Y} \supseteq se_{\square}}{\mathcal{X} \supseteq \kappa_{\square}^{-1}\mathcal{Y}}$$

$$\frac{\mathcal{X} \supseteq \mathcal{Y} \quad \mathcal{Y} \supseteq se_{\square}}{\mathcal{X} \supseteq se_{\square}}$$

준비하고 연기하는 규칙 P (컴파일시간)

$$\frac{\mathcal{X} \supseteq if_{\square}(S, \mathcal{X}_1) \quad \text{input } v \in S}{\mathcal{X} \supseteq \mathcal{X}_1}$$

$$\frac{\mathcal{X} \supseteq app_{\square}(\mathcal{X}_1, \mathcal{X}_2) \quad \mathcal{X}_1 \supseteq \lambda x.e}{\mathcal{X} \supseteq \mathcal{X}_e \quad \mathcal{X}_x \supseteq \mathcal{X}_2}$$

$$\frac{\mathcal{X} \supseteq \kappa_{\square}^{-1}\mathcal{Y} \quad \mathcal{Y} \supseteq \kappa\,\mathcal{Z}}{\mathcal{X} \supseteq \mathcal{Z}}$$

$$\frac{\mathcal{X} \supseteq \mathcal{Y} \quad \mathcal{Y} \supseteq ae}{\mathcal{X} \supseteq ae}$$

실행시간에 푸는 규칙 S_P

그림 8 컴파일시간에 준비하는 규칙 P와 실행시간에 푸는 규칙 S_p

$$X \supseteq \text{if}_{\neg}(V_1, X_2),$$
$$X \supseteq \text{if}_{\neg}(V_2, X_3),$$
$$X \supseteq \text{if}_{\neg}(\overline{V_1 \cup V_2}, X_2 \cup X_3)$$

집합식 $\text{if}_{\square}(V, X)$는 프로그램의 입력이 V의 원소일 때만, 집합 X가 된다. 마지막에 있는 $\text{if}_{\neg}$관계식은 입

력이 V_1, V_2에 모두 속하지 않는 경우에 사용된다.

$X \supseteq \text{if}_{\square}(V, X)$에 대한 점화관계식은 프로그램의 입력 ν에 대한 간단한 멤버쉽 테스트 $\nu \in V$이다:

$$\frac{X \supseteq \text{if}_{\square}(V, X_1) \quad \nu \in V}{X \supseteq X_1}$$

정규 트리 문법 V에 대한 테스트 $\nu \in V$는 다항식(polynomial) 시간이 걸린다[12]: ν의 크기 × V에 있는 비단말(non-terminal) × 함수(자료 생성자)의 인자 수(arity)의 합.

case식에 대한 입력에 의존적인 관계식을 정의하는 규칙은 다음과 같이 바뀐다.

$$\frac{X \supseteq case(X_1, x, X_2, X_3) \quad X_1 \supseteq se_{\square}}{X \supseteq case_{\square}(X_1, x, X_2, X_3)}$$

$$\Rightarrow \frac{\begin{array}{c} X \supseteq case(X_1, x, X_2, X_3) \quad X_1 \supseteq se_{\neg} \\ V_1 = SVS_{\mathcal{S}}(X_1 \subseteq x \top)(a) \\ V_2 = SVS_{\mathcal{S}}(X_1 \subseteq \overline{x} \top)(a) \end{array}}{\begin{array}{c} X \supseteq \text{if}_{\neg}(V_1, X_2) \quad X \supseteq \text{if}_{\square}(V_2, X_3) \\ X \supseteq \text{if}_{\square}(\overline{V_1} \cup \overline{V_2}, X_2 \cup X_3) \end{array}}$$

그림 8은 입력에 의존하는 관계식을 규명하고, 준비하고, 연기하는 규칙 P와, 준비된 규칙을 실행시간에 푸는 규칙 S_P이다.

분석을 준비하는 과정은 다음과 같이 정의된다.

let $\mathcal{S} \rhd C$ 관계식의 도출(컴파일시간)

$C' = (S \cup P)^*(C)$ 동적 관계식을 준비하기 (컴파일시간)

in $S_P^*(|C'| \cup \{X_a \supseteq \nu\})$ 준비된 관계식을 풀기 (실행시간)

먼저, 프로그램의 초기 관계식 C를 그림 3의 $\rhd$ 규칙을 이용해서 모은 후, S규칙을 C에 적용하여 입력에 무관한 관계식을 풀고, 동시에 규칙 P를 적용하여 입력에 의존하는 관계식을 규명하고, 준비한다. 결과로 나온 관계식의 집합 C'은 실행시간에 프로그램의 입력 ν에 의해 풀어지게 된다. 앞에서와 같이, $|C'|$는 C'의 관계식들 중 핵 관계식으로 풀려진 것 또는 동적 관계식으로 준비된 관계식들의 집합이다.

정의 6. (준비된 분석 $psba$) 프로그램 $\mathcal{S}$의 입력 변수가 a, 도출된 초기 관계식이 $\mathcal{S} \rhd C$, 입력 값이 ν라 하자. 준비된 분석은 $C' = (S \cup P)^*(C)$이고, $C' = S_P^*(|C'| \cup \{X_a \supseteq \nu\})$일 때, $psba_\nu(\mathcal{S})(l) \equiv |atom(C'(X_l))|$로 정의된다.

준비된 분석은 입력에 의존하는 성질을 안전하게 예측한다:

정리 8. ($psba$의 안전성) $[\nu/a]\mathcal{S} \mapsto^* \mathcal{E}[v_l]$이면 $v \in psba_\nu(\mathcal{S})(l)$이다.

증명. $psba$가 sba_ν의 안전한 예측이라는 것을 보임으로써 $psba$의 안전성을 증명한다. $sba_\nu(\mathcal{S})$와 $psba_\nu(\mathcal{S})$의 닫힌 집합(closure)에 해당하는 두 개의 연속(continuous) 함수 F_ν와 P_ν를 정의한다[13]. 그 후, P_ν의 최소 고정점(least fixpoint)이 F_ν의 최소 고정점보다 크거나 같다는 것을 보인다. A.2 절에 자세한 증명을 하였다.

6. 예

[14]에서 사용되었던 그림 9의 패킷 필터 프로그램을 생각해 보자. 이 프로그램은 다음과 같은 자료구조를 가지는 패킷을 입력으로 받아, 패킷의 ip 소스가 FOO이면 true를 반환한다[4].

```
p =
(λpacket.
  (λethertype. λip. λarp.
    case ethertype of
      IP => (λipsrc. λipdst. case_a ipsrc of FOO => TRUE  (*)
                                          | _ => e
            ) #1(ip) #2(ip)
    | _ => ···
  ) #1(packet) #1(#2(packet)) #2(#2(packet))
) α
```

그림 9 예-패킷 필터 프로그램

$$packet ::= \langle ethertype, \langle ip, arp \rangle \rangle$$

$$ip ::= \langle ipsrc, ipdst \rangle$$

$$arp ::= \langle arpsrc, arpdst \rangle$$

(*)로 마크된 casea식에 대한 분석을 생각해 보자. $\mathcal{S} \rhd C$로부터 만든 casea의 초기 관계식은 다음과 같다.

$$X_{\ case_a} \supseteq case(X_{\ \text{ipsrc}}, \text{FOO}, X_{\ \text{TRUE}}, X_e)$$

변수 ipsrc가 입력 a의 하위구조에 결합(bind)되므로,

4) 예로 든 프로그램은 값의 쌍(pair)와 쌍에서의 첫번째(#1), 두번째(#2) 값을 고르는 식을 가지고 있다. 이 논문에서는 위의 두 식에 대해서는 분석 규칙을 생략하였다. 전체 시스템이 코데피니트(co-definite)한 것을 깨지 않고, 슬라이싱 전달 규칙을 추가할 수 있다.

X_{ipsrc}는 입력에 의존한다.

값을 자르는 분석을 이용하여, 점화 관계식 $X_{ipsrc} \supseteq FOO\ Y$와 $X_{ipsrc} \supseteq x\ Y(x \neq FOO)$를 단축시켜, 프로그램의 입력에 대한 멤버쉽(membership) 테스트가 되도록 만들 수 있다.

$$SVS_{\&}(X_{ipsrc} \subseteq FOO\ \top)$$
$$SVS_{\&}(X_{ipsrc} \subseteq \overline{FOO}\ \top)$$

그림 10은 $SVS_{\&}(X_{ipsrc} \subseteq FOO\ \top)(\alpha)$를 푸는 과정을 보여준다. 입력 X_α에 대한 결과는

$$X_\alpha \subseteq \langle IP, \langle\langle FOO, FOO\rangle, \top\rangle\rangle \text{ 또는}$$
$$X_\alpha \subseteq \langle \top r, \langle\langle FOO, \top\rangle, \top\rangle\rangle$$

이다. 따라서, 점화 관계식 $X_{ipsrc} \supseteq FOO\ Y$를 멤버쉽 테스트 $\alpha \in \langle \top, \langle\langle FOO, \top\rangle, \top\rangle\rangle$로 대체할 수 있다. 즉, 실행 중에, 입력이 위의 테스트를 통과하면 준비된 관계식

$X_{case_\alpha} \supseteq if_{\ }(\langle \top, \langle\langle FOO, \top\rangle, \top\rangle\rangle, X_{TRUE})$, 는 $X_{case_\alpha} \supseteq X_{TRUE}$ 로 풀려진다.

7. 관련 연구

자료 흐름 분석을 전문화하는 다른 연구들은 우리 연구에 비해 다음의 세 가지 한계를 가진다: 프로그램의 흐름을 미리 알고 있어야 한다. 실행시간에 빠르게 풀기 위한 컴파일 시간 준비(값을 자르는 분석을 통한)가 없다. 전문화 과정의 정확성이 수학적으로 정의되고 증명되지 않았다. Sharma등의 방법[15]은 여러 가지의 자료 흐름들을 모으게끔 만드는 지점을 찾아내어, 그 점에 대한 분석을 실행시간으로 미룬다. 실행 시간에 연기된 점으로 여러 개의 자료 흐름 엣지들 중 하나가 선택되고, 정적 자료 흐름 분석에 비해 정확한 결과를 내게된다. 이 방법은 프로그램의 함수 흐름 그래프(control flow graph)를 미리 알고 있다고 가정하기 대문에 함수가 다른 값과 같이 사용될 수 있는 고차(higher-order) 언어에서는 사용할 수 없다. 또, 실행 시간 전문화를 위해서 실행중인 프로그램의 흐름이 연기된 점까지 도달하기를 기다려야 한다. 이런 지연을 줄이기 위해서 컴파일 시간에 어떤 준비도 하지 않는다. Moon등의 포트란 프로그램에 대한 동적 의존 분석(dynamic dependence analysis)[16]도 같은 한계를 가진다.

Ammons와 Larus는 자료 흐름 분석의 정확성을 향상시키기 위해서 계측된 자료(profiled data)를 사용한다[17]. 프로그램의 함수 흐름 그래프에서 자주 수행되는 경로를 분리하여 자료 흐름 분석을 함으로써 정확성을 향상시켰다. 이 방법은 계측 자료를 모아서 자주 수행되는 경로를 찾아내고, 그 경로에 대한 자료 흐름 분석을 분리시키고, 전체 프로그램에 대해 안전한 분석을 재구성하기 위한 부담이 있다. 이러한 과정은 코드 제공자/소비자 모델에서 볼 때, 코드 제공자측에서 하는 것이 아니라, 사용자측에서 하게 되므로 큰 부담이 될 수 있다. 우리 방법은 실행과정을 관찰하여 자료를 계측하지 않기 때문에 이런 부담이 없다.

실행시간 코드 생성 기법[18, 19, 20, 21]은 입력에 의해 만들어지는 동적 성질을 찾아내는 것이 아니라, 실행시간에 목적 코드를 최적화 하는데 초점을 두고 있다. 이 방법은 실행시간에 특정 입력에 대한 프로그램의 동적 성질을 분석하지 않는다. 모든 분석은 컴파일시간에 이루어진다. 기본적으로 프로그램의 어떤 부분이 전문화될 것인지에 대한 프로그래머의 주석(annotation)을 바탕으로 한 결합 시간 분석(binding-time analysis)이다. 실행시간에 여러 개의 입력중 일부분을 알게 되면, 부분 계산(partial evaluation)[22]에 바탕을 둔 코드 전문화 과정에 의해 부분 입력에 최적화 된 코드를 생성하게 된다.

Flanagan과 Felleisen의 부품별(componential) 집합 기반 분석[5]은 우리 방법과 전혀 다른 관점에서 적용될 수 있다. 이 방법은 우리 분석을 분리해서, 부품별로 준비하여 모듈별 실행시간 전문화 기법을 만드는 데 적용될 수 있을 것이다. 이 방법에서 사용된 관계식의 단순화 알고리즘은 관계식을 푸는 과정에서 새로운 관계식이 무한하게 생성되는 경우 이를 제한하는 방법이다. 본 논문에서 사용된 분석은 생성되는 관계식의 수가 프로그램의 크기에 의해 제한되기 때문에 이 과정이 필요 없다. 만약 시작점으로 사용된 정적 분석이 위의 클래스에 들어간다면 부품별 집합 기반 분석에서 제시된 단순화 과정을 적용할 수 있을 것이다.

값을 자르는 분석은 자르는 기준에 해당하는 사용자의 주석(annotation)을 사용하는 하위타입 또는 정제된 타입 유추[23, 24, 25]로 볼 수 있다. 다른 점은 하위 타입이나 정제된 타입 시스템에서 사용하는 타입 계층(hierarchy)이 우리가 사용하는 정규 트리 문법의 도메인보다 더 작다는 것이다. Reps와 Turnidge의 문맥 분석(context analysis)[26]과 Liu의 의존성 분석[27]은 고정된 아주 작은 래티스를 사용한다. Bourdoncle의 Syntox 시스템[28]은 값을 자르는 분석의 한 종류이다. Syntox에서 사용된 값을 자르는 분석이 파스칼 프로그램의 디버깅에 사용할 수 있음을 보였다. 그러나, 우리 방법은 임의의 자료형을

지원하는 반면, Syntox의 값은 정수의 범위(range)로 제한되어 있다. Biswas가 제안한 필요할 때 푸는(demand-driven) 집합 기반 분석[29]은 전체 프로그램의 값을 계산하는데 영향을 준 하위 식들을 찾는 방법이다. 이 방법은 프로그램의 식을 자를 뿐, 식의 값을 자르지는 않는다.

8. 결 론

본 논문에서 우리는 정적 분석을 사용하여 프로그램의 입력에 의존하는 성질을 예측하는 방법을 제안하였다. 우리가 제안한 방법은 입력에 무관한 성질을 예측하도록 설계된 정적 분석을 입력에 의존하는 성질을 예측하는 분석으로 변환할 수 있다. 이 방법은 실행 중에 프로그램의 성질을 알아내기 위해서 실행중인 프로그램을 관찰하는 코드나 계측 과정을 필요로 하지 않는다.

정적 분석의 가장 마지막 부분을 프로그램의 실행 시간으로 미루는 것이 이 논문의 핵심 아이디어다. 먼저 정적 분석을 분석하여, 프로그램의 입력에 민감하여 프로그램의 실행시간으로 연기되어야 하는 분석의 부분을 찾아낸다. 그 후, 값을 자르는 분석을 이용하여 이 부분을 재구성하여 프로그램의 입력에 대한 간단한 멤버쉽 테스트에 의해 분석이 풀어질 수 있도록 한다. 이런 재구성과정을 통해 준비된 분석들은 프로그램의 입력이 나타나기만 하면 순간적으로, 동시에 풀려질 수 있다.

이 방법의 장점은 1) 집합 기반 분석이 주어지면 그에 해당하는 어떤 입력에 기반한 성질도 분석할 수 있다. 따라서, 단순히 실행 횟수만을 계측하는 전통적인 계측 방법에 비해 더 일반적인 분석 방법이다. 2) 수행중인 프로그램에 대해 실행시간 부담(overhead)이 없다. 3) 함수 값이 다른 값과 구별 없이 사용되는 고차 언어(higher-order language)에 적용할 수 있다. 4) 엄밀히 정의되었고, 정확하게 증명되었다.

참 고 문 헌

[1] Thomas Ball and James R. Larus. Optimally profiling and tracing program. ACM Transactions on Programming Languages and Systems, 16(4):1319-1360, July 1994.

[2] Thomas Ball, Peter Mataga, and Mooly Sagiv. Edge profiling versus path profiling: the showdown. In Proceedings of the ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, January 1998.

[3] Robin Milner, Mads Tofte, Robert Harper, and David MacQueen. The Definition of Standard ML (Revised). MIT Press, 1997.

[4] Martin S. Feather. A survery and classification of some program transformation approaches and techniques. In L.G.L.T. Meertens, editor, Program Specification and Transformation, pages 165-195. Elsevier Science Publishers, 1987.

[5] Cormac Flanagan and Matthias Felleisen. Componential set-based analysis. In Proceedings of the SIGPLAN conference on Programming Language Design and Implementation, 1997.

[6] Nevin Heintze. Set Based Program Analysis. PhD thesis, Carnegie Mellon University, October 1992.

[7] Nevin Heintze. Set based analysis of ML Programs. Technical Report CMU-CS-93-193, Carnegie Mellon University, July 1993.

[8] Nevin Heintze and David McAllester. Linear-time subtransitive control flow analysis. In Proceedings of the SIGPLAN Conference on Programming Language Design and Implementation, pages 261-272, June 1997.

[9] Alex Aiken and Nevin Heintze. Constraint-based program analysis. Tutorial Notes of the ACM Symposium on Principles of Programming Languages, January 1995.

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

[11] Witold Charatonik and Andreas Podelski. Co-definite set constraints. In Lecture Notes in Computer Science, volume 1379, pages 211-225. Springer-Verlag, Proceedings of the 9th International Conference on Rewriting Techniques and Applications - RTA'98 edition, 1998.

[12] Hubert Comon, Max Dauchet, Remi Gilleron, Florent Jacquemard, Denis Lugiez, Sophie Tison, and Marc Tommasi. Tree automata techniques and applications, April 1999.

[13] Patrik Cousot and Radhia Cousot. Compositional and inductive semantic

definitions in fixpoint, equational, constraint, closure-condition, rule-based and game-theoretic form. In Lecture Notes in Computer Science, volume 939, pages 293-308. Springer-Verlag, proceedings of the 7th international conference on computer-aided verification edition, 1995.

[14] Steven McCanne and Van Jacobson. The BSD packet filter: A new architecture for user-level packet capture. In Proceedings of the Winter 1993 USENIX Conference, pages 259-269. USENIX Association, January 1993.

[15] Shamik Sharma, Anurag Acharya, and Joel Saltz. Deferred data-flow analysis. Technical Report TRCS98-38, University of California, Santa Barbara, December 1998.

[16] Sungdo Moon, Mary W. Hall, and Brian R. Murphy. Predicated array data-flow analysis for run-time parallelization. In Proceedings of the ACM International Conference on Supercomputing, July 1998.

[17] Glenn Ammons and James Larus. Improving data-flow analysis with path profiles. In Proceedings of the SIGPLAN Conference on Programming Language Design and Implementation, June 1998.

[18] Charles Consel and François Noël. A general approach for run-time specialization and its application to C. In Proceedings of The ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, January 1996.

[19] Mark Leone and Peter Lee. Optimizing ML with run-time speicialzation. Technical Report CMU-CS-95-205, Carnegie Mellon University, December 1995.

[20] Jeol Auslander, matthai philipose, Craig Chambers, Susan J. Eggers, and Brian N. Bershad. Fast, effective dynamic compilation. In Proceedings of SIGPLAN conference on Programming Language Design Implementation, June 1998

[21] Brian Grant, Markus Mock, Matthai Philipose, Craig Chambers, and Susan J. Eggers. Annotation-directed run-time specialization in C. In Proceedings of The ACM Symposium on Partial Evaluation and Program Manipulations.

[22] Charles Consel and Olivier Danvy. Tutorial notes on partial evaluation. In the Twentieth ACM Symposium on Principles of Programming Languages, January 1993.

[23] You-Chin Fuh and Prate다 Mishra. Type inference with subtypes. Theoretical Computer Science, 73:155-175, 1990.

[24] Tim Freeman and Frank Pfenning. Refinement types for ML. In Proceedings of the SIGPLAN Conference on Programming Language Design and Implementation, pages 268-277, 1991.

[25] Mario Coppo, Ferruccio Damiani, and Paola Giannini. Refinement types for program analysis. In Lecture Notes in Computer Science, volume 1145. Springer-Verlag, 1996.

[26] Thomas Reps and Todd Turnidge. Program specialization via program slicing. In Lecture Notes in Computer Science, volume 1110, pages 409-429. Springer-Verlag, 1996

[27] Yanhong A. Liu. Dependence analysis for recursive data. In Proceedings of the IEEE International Conference on Computer Language, pages 206-215, May 1998.

[28] François Bourdoncle. Abstract debugging of higher-order imperative Languages. In Proceedings of the SIGPLAN conference on Programming Language Design and Implementation, 1993.

[29] Sandip K. Biswas. A demand-driven set-based analysis. In Proceedings of The ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, pages 372-385, 1997.

A. 증명

A.1 SVS의 정확성

다음의 두 보조정리는 정리 6을 증명하기 위해 사용되고, 정리 6은 다시 정리 7을 증명하는 데 사용된다.

보조정리 1. $C = B^*(C_0)$, $D = atom(C)$, $I_C = gm(C)$, $I_D = gm(D)$ 라 하면,

$v \notin I_D(X) \Leftrightarrow v \notin I_D(a)$ for some $X \subseteq a$ in C

이다.

증명. $v \notin gm(C)(X)$라 하면, C는 $X \subseteq se$형태를 가지고 $v \notin gm(C)(se)$인 X의 상한(upper bound)을 가진다. 따라서, $v \notin I_D(X) \Leftrightarrow v \notin I_D(a)$, $X \subseteq a$ in D이다. 이것은 어떤 $X \subseteq a \in C$에 대해서, $v \notin I_D(X) \Leftrightarrow v \notin I_D(a)$임을 의미한다.□

보조정리 2. $C = B^*(C_0)$, $D = atom(C)$, $I_C = gm(C)$, $I_D = gm(D)$라 하면,

$I_D \subseteq I_C$

이다.

증명. I_D가 C의 모델, 즉 $v \notin I_D(se) \Rightarrow v \notin I_D(X)$임을 증명한다.

- $X \subseteq ae$이 C에 속할 경우
  $\Rightarrow$ $X \subseteq ae$이 D에 속하므로 참이다.
- $X \subseteq Y$이 C에 속하고, $v \notin I_D(Y)$인 경우
  $\Rightarrow$ C에 속한 $v \notin I_D(ae)$에 대해 $Y \subseteq ae$
  (보조정리 1에 의해)
  $\Rightarrow$ $X \subseteq Y$이고 $Y \subseteq a$이 C에 속하므로,
  $X \subseteq ae$가 C에 속한다.
  $\Rightarrow$ $X \subseteq ae$가 D에 속한다.
  $\Rightarrow$ $v \notin I_D(X)$ □

정리 6. 프로그램 $\mathcal{S}$가 입력변수 a를 가지고, e는 $\mathcal{S}$의 하위식이며, V는 값의 집합이고, $\mathcal{S} \rhd_i C_i$에 의해 초기 관계식들이 만들어진다고 하면, $\| atom(B^*(C_i \cup \{X_e \subseteq V\})) \|$는 $C_i \cup \{X_e \subseteq V\}$의 최대 모델(greates model)이다.

증명. $C' = C_i \cup \{X_e \subseteq V\}$라 하자. 먼저, $gm(C') = gm(B^*(C'))$임을 규칙 B가 $gm(C')$를 유지하는 관계식들만 추가한다는 것을 이용해서 보일 수 있다.

$$[\text{TRANS}] \quad \frac{X \subseteq Y \quad Y \subseteq ae}{X \subseteq ae}$$

$X \subseteq Y$와 $Y \subseteq ae$는 $I(X) \supseteq I(ae)$임을 의미하고, $X \subseteq ae$도 역시 이를 의미한다.

$$[\text{TRANS}-\text{CON}_1] \quad \frac{xX \subseteq Y \quad Y \subseteq xZ}{X \subseteq Z}$$

$xX \subseteq Y$는 $I(X) \subseteq \{v \mid xv \in I(Y)\}$를 의미하고, $Y \subseteq xZ$는 $I(Y) \subseteq \{xv \mid v \in I(Z)\}$를 의미한다. 따라서, $xX \subseteq Y$와 $Y \subseteq xZ$는 $I(X) \subseteq I(Z)$를 의미하고, $X \subseteq Z$도 역시 이를 의미한다.

$$[\text{TRANS}-\text{CON}_2] \quad \frac{xX \subseteq Y \quad Y \subseteq x\top}{X \subseteq \top}$$

$xX \subseteq Y$는 $I(X) \subseteq \{v \mid xv \in I(Y)\}$를 의미하고, $Y \subseteq x\top$은 $I(Y) \subseteq \{xv \mid v \in Val\}$을 의미한다. 따라서, $xX \subseteq Y$와 $Y \subseteq x\top$은 $I(X) \subseteq Val$을 의미하고, $X \subseteq \top$도 역시 이를 의미한다.

$$[\text{TRANS}-\text{CON}_3] \quad \frac{xX \subseteq Y \quad Y \subseteq \overline{x'}\top}{X \subseteq \top}$$

$xX \subseteq Y$는 $I(X) \subseteq \{v \mid xv \in I(Y)\}$를 의미하고, $Y \subseteq \overline{x'}\top$은 $I(Y) \subseteq \{x''v \mid x'' \neq x', v \in Val\}$을 의미한다. $x \neq x'$이기 때문에, $xX \subseteq Y$와 $Y \subseteq \overline{x'}\top$은 $I(X) \subseteq Val$을 의미하고, $X \subseteq \top$도 역시 이를 의미한다.

$$[\text{TRANS}-\text{CON}_4] \quad \frac{xX \subseteq Y \quad Y \subseteq \top}{X \subseteq \top}$$

$xX \subseteq Y$는 $I(X) \subseteq \{v \mid xv \in I(Y)\}$를 의미하고, $Y \subseteq \top$은 $I(Y) \subseteq \{v \mid v \in Val\}$을 의미한다. $\{v \mid xv \in Val\} = Val$이기 때문에, $xX \subseteq Y$와 $Y \subseteq \top$은 $I(X) \subseteq Val$을 의미하고, $X \subseteq \top$도 역시 이를 의미한다.

이제, $gm(B^*(C_i)) = gm(atom(B^*(C_i)))$을 증명해야 한다. $atom(B^*(C_i)) \subseteq B^*(C_i)$이므로, $gm(B^*(C_i)) \subseteq gm(atom(B^*(C_i)))$이고, 보조정리 2에 의해서, $gm(atom(B^*(C_i))) \subseteq gm(B^*(C_i))$이 증명된다. □

다음의 보조정리는 정리 7을 증명하기 위해 사용된다.

보조정리 3. $\mathcal{S} \rhd_i C_i$이고, I가 $C_i \cup \{X_0 \supseteq V_0\}$의 최소 모델이라고 하자. 모든 $e \in \mathcal{S}$에 대해, $I \vdash e \rightsquigarrow v$는 $v \in I(X_e)$임을 의미하고, $\nu \in I(\alpha)$일 때, I는 $[\nu/\alpha]\mathcal{S}$에 대해 안전한 집합 환경이다.

증명. $I \vdash e \rightsquigarrow v$의 집합 기반 의미 구조의 증명 트리 크기에 대한 귀납법을 사용해서 증명.

- 증명 트리 크기가 1일 때

  [VAR] $e = x$

  $I(x) \vdash x \rightsquigarrow v \Rightarrow v \in I(X_x)$(의미 구조에 의해)

  [LAM] $e = \lambda x.e'$

  $I \vdash \lambda x.e' \rightsquigarrow \lambda x.e' \Rightarrow \lambda x.e' \in I(X_e)$

  (I와 $\rhd_i$의 정의에 의해)

• 증명 트리 크기가 1 이상일 때

[CON] $e = \mathrm{con}\, x\, e$

$I(X_e) = V$

$\Rightarrow$ $I(X_1) = \{v \mid x v \in V\}$

(I와 $\rhd_i$의 정의에 의해)

$\Rightarrow$ $(I \vdash e_1 \rightsquigarrow v' \Rightarrow v' \in \{v \mid x v \in V\})$

(귀납 가설에 의해)}

$\Rightarrow$ $(I \vdash \mathrm{con}\, x\, e_1 \rightsquigarrow v \Rightarrow v \in V)$

(의미 구조에 의해)

[DECON] $e = \mathrm{decon}\, x\, e_1$

$I(X_e) = V$

$\Rightarrow$ $I(X_1) = \{x v \mid v \in V\}$

(I와 $\rhd_i$의 정의에 의해)

$\Rightarrow$ $(I \vdash e_1 \rightsquigarrow v' \Rightarrow v' \in \{x v \mid v \in V\})$

(귀납 가설에 의해)}

$\Rightarrow$ $(I \vdash \mathrm{decon}\, x\, e_1 \rightsquigarrow v \Rightarrow v \in V)$

(의미 구조에 의해)

[APP] $e = e_1\, e_2$

$I(X_e) = V$

$\Rightarrow$ $I(X_e) \subseteq V$

$I(X_1) \subseteq Lam_{\mathcal{S}}(e_1)$

$I(X_2) \subseteq I(X_x)$

(I와 $\rhd_i$의 정의에 의해)

$\Rightarrow$ $(I \vdash e_1 \rightsquigarrow \lambda x.e' \Rightarrow \lambda x.e' \in Lam_{\mathcal{S}}(e_1)$

$(I \vdash e' \rightsquigarrow v \Rightarrow v \in V)$

$I(x) \ni v'$, if $I \vdash e_2 \rightsquigarrow v'$

(귀납 가설에 의해) … (*)

$\Rightarrow$ $(I \vdash e_1\, e_2 \rightsquigarrow v \Rightarrow v \in V)$

(의미 구조에 의해)

[CASE] $e = \mathrm{case}\, e_1\, x\, e_2\, e_3$

$I(X_e) = V$

1) $\Rightarrow$ $I(X_1) \subseteq x \top$

$I(X_2) \subseteq V$

(I와 $\rhd_i$의 정의에 의해)

$\Rightarrow$ $I \vdash e_1 \rightsquigarrow v_1 \Rightarrow v_1 \in x \top$

$I \vdash e_2 \rightsquigarrow v_2 \Rightarrow v_2 \in V$

(귀납 가설에 의해)

$\Rightarrow$ $I \vdash \mathrm{case}\, e_1\, x\, e_2\, e_3 \rightsquigarrow v \Rightarrow v \in V$

(의미구조에 의해)

2) $\Rightarrow$ $I(X_1) \subseteq \overline{x} \top$

$I(X_3) \subseteq V$

(I와 $\rhd_i$의 정의에 의해)

$\Rightarrow$ $I \vdash e_1 \rightsquigarrow v_1 \Rightarrow v_1 \in \overline{x} \top$

$I \vdash e_3 \rightsquigarrow v_3 \Rightarrow v_3 \in V$

(귀납 가설에 의해)

$\Rightarrow$ $I \vdash \mathrm{case}\, e_1\, x\, e_2\, e_3 \rightsquigarrow v \Rightarrow v \in V$

(의미구조에 의해)

3) $\Rightarrow$ $I(X_2) \subseteq V$

$I(X_3) \subseteq V$

(I와 $\rhd_i$의 정의에 의해)

$\Rightarrow$ $I \vdash e_2 \rightsquigarrow v_2 \Rightarrow v_2 \in V$

$I \vdash e_3 \rightsquigarrow v_3 \Rightarrow v_3 \in V$

(귀납 가설에 의해)

$\Rightarrow$ $I \vdash \mathrm{case}\, e_1\, x\, e_2\, e_3 \rightsquigarrow v \Rightarrow v \in V$

(의미 구조에 의해)

집합 환경이 모든 닫힌 식 e_0의 수행 중에 일어날 수 있는 결합을 가진다면, 그 집합 환경은 e_0에 대해 안전하다고 한다[20]. $Lam_{\mathcal{S}}(e_1) \ni \lambda x.e'$, $I(\alpha) \ni \nu$일 때 모든 호출식 $e_1\, e_2$에 대해, $I(X_x) \subseteq I(X_2)$이므로, (*)는 I가 $[\nu/\alpha]\mathcal{S}$에 대해 안전하다는 것을 보여준다. □

정리 7. ($SVS_{\mathcal{S}}(e \subseteq V)$의 안전성) 프로그램 $\mathcal{S}$의 입력 변수가 α, 하위식 $e \in \mathcal{S}$라 하자. $\Sigma \in SVS_{\mathcal{S}}(e \subseteq V)$이면

$\forall \nu \in \Sigma(\alpha)([\nu/\alpha]\mathcal{S} \mapsto^* \mathcal{E}[v_e] \Rightarrow v \in V)$이다.

증명. 정리 6에 의해, Σ는 $C_i \cup \{X_e \subseteq V\}$의 최대 모델이다. 따라서, 보조정리 3에 의해, $\Sigma \vdash e \rightsquigarrow v \Rightarrow v \in \Sigma(e)$이고, Σ는 $[\nu/\alpha]\mathcal{S}$에 대해 안전한 집합 환경이다. 또, Σ가 $C_i \cup \{X_e \subseteq V\}$의 최대 모델이므로, $\Sigma(e) \subseteq V$이다.

따라서, $[\nu/\alpha]\mathcal{S} \mapsto^* \mathcal{E}[v_e]$

$\Rightarrow$ $\Sigma \vdash e \rightsquigarrow v$

(Σ는 $[\nu/\alpha]\mathcal{S}$에 대해 안전하므로)

$\Rightarrow$ $v \in \Sigma(e)$

(보조정리 3에 의해)

$\Rightarrow \quad v \in V$

(Σ는 최대 모델이므로)

A.2 psba의 안전성

정리 8. (*psba*의 안전성) $[\nu/\alpha]\,\mathcal{S} \mapsto^{*} \mathcal{E}\,[v_l]$ 이면 $v \in psba_\nu(\mathcal{S})(l)$이다.

증명. *psba*가 sba_ν의 안전한 예측(approximation)이라는 것을 보임으로써 *psba*의 안전성을 증명한다. 먼저, $sba_\nu(\mathcal{S})$와 $psba_\nu(\mathcal{S})$의 닫힌 집합(closure)에 해당하는 두 개의 연속(continuous) 함수 F_ν와 P_ν를 정의한다[13]. 그리고 나서, P_ν의 최소 고정점(least fixpoint)이 F_ν의 최소 고정점보다 크거나 같음을 증명한다.

프로그램 $\mathcal{S}$의 초기 관계식 C가 $\mathcal{S} \rhd C$에 의해 도출되어졌다고 하면, $sba_\nu(\mathcal{S})$는 $F_\nu : (Vars(C) \to 2^{Val}) \to (Vars(C) \to 2^{Val})$의 최소 고정점 $lfp\,F_\nu$와 같다[13].

또 다른 연속 함수 $P_\nu : (Vars(C) \to 2^{Val}) \to (Vars(C) \to 2^{Val})$를

$$
\begin{array}{lll}
\mathcal{F}_\nu(\rho)(\mathcal{X}_l) = & \textbf{case } e \textbf{ of} & \\
\quad \lambda x.e' & : & \{\lambda x.e'\} \\
\quad x & : & \{v \mid e_1\ e_2 \in \wp, \lambda x.e' \in \rho(\mathcal{X}_1), v \in \rho(\mathcal{X}_2)\} \\
\quad \alpha & : & \{\nu\} \\
\quad e_1\ e_2 & : & \{v \mid \lambda x.e_3 \in \rho(\mathcal{X}_1), v \in \rho(\mathcal{X}_3)\} \\
\quad \textbf{con}\ \kappa\ e_1 & : & \{\kappa(v) \mid v \in \rho(\mathcal{X}_1)\} \\
\quad \textbf{decon}\ \kappa\ e_1 & : & \{v \mid \kappa(v) \in \rho(\mathcal{X}_1)\} \\
\quad \textbf{case}\ e_1\ \kappa\ e_2\ e_3 & : & \{v \mid v \in \rho(\mathcal{X}_2), \kappa(v') \in \rho(\mathcal{X}_1)\} \\
 & & \cup\{v \mid v \in \rho(\mathcal{X}_3), \kappa'(v') \in \rho(\mathcal{X}_1), \kappa' \neq \kappa\}
\end{array}
$$

$$
\begin{array}{lll}
\mathcal{P}_\nu(\varphi)(\mathcal{X}_e) = \textbf{case } e \textbf{ of} & & \\
\quad \mathbf{1} & : & \{1\} \\
\quad \lambda x.e' & : & \{\lambda x.e'\} \\
\quad \alpha & : & \{\nu\} \\
\quad e_1\ e_2 & : & \{v \mid \lambda x.e' \in \varphi(\mathcal{X}_1), v \in \varphi(\mathcal{X}'_e)\} \\
\quad \textbf{con}\ \kappa\ e_1 & : & \{\kappa(v) \mid v \in \varphi(\mathcal{X}_1)\} \\
\quad \textbf{decon}\ \kappa\ e_1 & : & \{v \mid \kappa(v) \in \varphi(\mathcal{X}_1)\} \\
\quad \textbf{case}\ \overline{e_1}\ \kappa\ e_2\ e_3 & : & \{v \mid v \in \varphi(\mathcal{X}_2), \kappa(v') \in \varphi(\mathcal{X}_1)\} \\
 & & \cup\{v \mid v \in \varphi(\mathcal{X}_3), \kappa'(v') \in \varphi(\mathcal{X}_1)\} \\
\quad \textbf{case}\ \underline{e_1}\ \kappa\ e_2\ e_3 & : & \{v \mid v \in \varphi(\mathcal{X}_2), \nu \notin \mathrm{SVS}_\wp(\mathcal{X}_1 \subseteq \overline{\kappa}\top)(\alpha)\} \\
 & & \cup\{v \mid v \in \varphi(\mathcal{X}_3), \nu \notin \mathrm{SVS}_\wp(\mathcal{X}_1 \subseteq \kappa\top)(\alpha)\}
\end{array}
$$

where

e, 컴파일시간 준비 후 $\mathcal{X}_e \supseteq se_{\square}$일 경우

$\bar{e}$, 그렇지 않을 경우

$psba_\nu(\mathcal{S})$이 P_ν의 최소 고정점 $lfp\,P_\nu$와 같아 지도록 정의한다. 식 $\overline{e}$는 그 식의 집합 관계식이 입력과 무관하다는 것을 나타내고, $\underline{e}$는 그 식의 집합 관계식이 입력에 의존적인 $X_e \supseteq se_{\square}$이라는 것을 나타낸다. 아무런 줄도 그어지지 않은 식은 입력에 의존적인지, 무관한지를 구별할 필요가 없는 식임을 의미한다. case $e_1\ x\ e_2\ e_3$의 e_1을 제외하고는, $P_\nu(\varphi)(\overline{e})$이 $P_\nu(\varphi)(\underline{e})$와 정확하게 일치한다.

$Q(\varphi, \rho)$가

$$(\forall e \in \mathcal{S}.\varphi(X_e) \supseteq \rho(X_e)) \wedge (\varphi \subseteq lfp\,P_\nu) \wedge (\rho \subseteq lfp\,F_\nu)$$

일 때, $Q(lfp\,P_\nu, lfp\,F_\nu)$를 고정점 귀납법(fixpoint induction)을 사용해서 증명한다.

$Q(\emptyset, \emptyset)$는 당연히 참이다.

이제, 귀납 가설이 $Q(\varphi, \rho)$일 때, $Q(P_\nu(\varphi), F_\nu(\rho))$가 성립함을 보이면 된다. 즉, $P_\nu(\varphi)(X_e) \supseteq F_\nu(\rho)(X_e)$를 보이면 된다.

[CON] $e = \mathrm{con}\ x\ e_1$.

$$
\begin{array}{ll}
P_\nu(\varphi)(X_e) = \{xv \mid v \in \varphi(X_1)\} & \\
\quad \text{(정의에 의해)} & \\
\supseteq \{xv \mid v \in \rho(X_1)\} & \\
\quad \text{(귀납 가설에 의해)} & \\
= F_\nu(\rho)(X_e) & \\
\quad \text{(정의에 의해)} &
\end{array}
$$

[CASE]를 제외한 다른 경우들은 모두 비슷하게 증명되어진다.

[CASE] $e = case\ \underline{e_1}\ x\ e_2\ e_3$.

$$
\begin{array}{ll}
P_\nu(\varphi)(X_e) & \\
= \{v \mid v \in \varphi(X_2), \nu \notin SVS_{\mathcal{S}}(X_1 \subseteq \overline{x}\top)(\alpha)\} & \\
\quad \cup \{v \mid v \in \varphi(X_3), \nu \notin SVS_{\mathcal{S}}(X_1 \subseteq x\top)(\alpha)\} & \\
\quad \text{(정의에 의해)} & \\
= \{v \mid v \in \varphi(X_2), xv' \notin (lfp\,F_\nu)(X_1)\} & \\
\quad \cup \{v \mid v \in \varphi(X_3), x'v'' \notin (lfp\,F_\nu)(X_1), x' \neq x\} & \\
\quad (*) & \\
= \{v \mid v \in \varphi(X_2), xv' \notin \rho(X_1)\} & \\
\quad \cup \{v \mid v \in \varphi(X_3), x'v' \notin \rho(X_1), x' \neq x\} & \\
\quad \text{(귀납 가설에 의해)} & \\
= F_\nu(\rho)(X_e) & \\
\quad \text{(정의에 의해)} &
\end{array}
$$

정리 4와 정리 7에 의해 $v \in sba_\nu(X) \wedge v \notin V \Rightarrow \nu \notin SVS_{\mathcal{S}}(X \subseteq V)(\alpha)$이므로, (*)는 사실이다. □

어 현 준

1996년 한국과학기술원 전산학과 학사(B.S.). 1998년 한국과학기술원 전산학과 석사(M.S.). 1998년 ~ 현재 한국과학기술원 전산학과 박사과정.

이 광 근

정보과학회논문지 : 소프트웨어 및 응용 제 27 권 제 3 호 참조