주어진 프로그램에서 예외상황을 발생시키는 테스트 데이타 생성 방법

(A Test Data Generation to Raise User-Defined Exceptions in First-Order Functional Programs)

류 석 영 † 이 광 근 ††

(Sukyoung Ryu) (Kwangkeun Yi)

요 약 주어진 프로그램에서 예외상황(exception)을 발생시키는 테스트 데이타를 자동으로 생성해주는 분석 방법을 제안한다. 분석 결과로 얻은 테스트 데이타를 사용하여, 프로그램 내에서 발생한 예외상황들이 프로그래머의 의도대로 처리되는지를 검사할 수 있다.

본 논문에서 제안하는 분석 방법은 입력으로 받은 프로그램에서 특정 예외상황이 발생한다는 조건을 시작으로 하여, 프로그램의 입력 값에 대한 제약식(constraints)을 만들어간다. 이 분석 방법이 옳다는 증명에 의해서, 분석 결과로 얻은 테스트 데이타를 입력으로 하여 프로그램을 수행시키면 지정한 예외상황이 항상 발생한다는 것을 보장할 수 있다.

함수를 인수나 결과값으로 전달하지 않고(first-order) ML 스타일의 예외상황 관리 방법을 제공하는 언어를 대상으로 하여 테스트 데이타 생성 방법을 제안하고, 이 분석 방법이 옳다는 것을 증명한 후 몇 가지 예를 사용하여 분석 과정을 설명한다.

Abstract We present a static analysis method to automatically generate test data that raise exceptions in input programs. Using the test data from our analysis, a programmer can check whether the raised exceptions are correctly handled with respect to the program's specification.

For a given program, starting from the initial constraint that a particular raise expression should be executed, our analysis derives necessary constraints for its input variable. Correctness of our analysis assures that any value that satisfies the derived constraints for the input variable will activate the designated raise expression.

In this paper, we formally present such an analysis for a first-order language with the ML-style exception handling constructs and algebraic data values, prove its correctness, and show a set of examples.

1. 서 론

예외상황 관리 방법을 제공하는 언어 (예를 들어, ML[1], Modula-3[2], Java[3], Ada[4]는 프로그래머로 하여금 프로그램의 예외적인 상황을 간단하게 정의하고 발생하고 처리할 수 있도록 한다. 프로그래머가 정의한 예외상황은 예외상황 발생 표현식(raise expression)에 의해 발생되고, 발생한 예외상황을 소멸할 수 있는 처리 표현식(handle expression)에 의해 처리된다.

예외상황 관리 방법은 프로그래머에 의해 아주 유용하게 사용된다. 단순히 프로그램 수행 중에 생기는 잘못된 경우를 처리하는 데에만 사용되는 것이 아니라, 여러 겹으로 겹쳐져 있는 프로그램의 제어 구조(control structure)를 한 번에 빠져 나오는 데에도 사용될 수 있고, 같은 연산 결과를 경우에 따라 다르게 처리해줄 때에도 사용될 수 있다.

하지만, 예외상황을 주의깊게 사용하지 않으면 오히

· 본 연구는 과학기술부 창의적연구진흥사업 추진으로 얻어진 결과임.

† 비 회 원 : 한국과학기술원 전산학과
puppy@cs.kaist.ac.kr
†† 종신회원 : 한국과학기술원 전산학과 교수
kwang@cs.kaist.ac.kr

논문접수 : 1999년 5월 24일
심사완료 : 2000년 1월 17일

려 프로그램의 안전도를 해치는 상황으로 발전할 수가 있다. 첫째, ML과 같이 타입을 검사하는 프로그래밍 언어라 할지라도, 예외상황에 의하여 프로그램의 안전도가 깨질 수 있다. ML 프로그램은 예외상황이 발생한 후에 적절하게 처리되지 않으면 프로그램이 비정상적으로 멈추게 되기 때문이다. 프로그램 내에서 발생한 예외상황이 제대로 처리되지 않으면 매우 위험한 상황이 발생할 수 있다[5]. 둘째, 프로그램에서 발생한 후 처리되지 않는 예외상황이 전혀 없는 경우라도, 발생한 예외상황이 프로그래머의 의도대로 맞게 처리되었는지를 확인하기는 매우 어렵다.

첫번째 문제에 대한 해결 방법은 이미 개발되어 있지만, 우리가 아는 바에 의하면, 두번째 문제는 아직 다루어진 적이 없다. 첫번째 문제에 대한 해결 방법으로는 주어진 프로그램에서 발생한 후 처리되지 않을 수 있는 예외상황을 예측하여 프로그래머에게 알려주는 도구가 개발되어 있다. 예를 들어, ML 프로그램에 대한 예외상황 분석기로[6, 7, 8, 9] 등이 있다.

두번째 문제에 대해 본 논문에서 제안하는 해결 방법은 주어진 프로그램 내의 모든 예외상황 발생 표현식을 수행시키는 테스트 데이타를 생성해주는 것이다. 자동으로 만들어낸 테스트 데이타를 입력값으로 프로그램을 수행시켜 예외상황이 발생하도록 한 후, 발생한 예외상황이 프로그램 내에 흘러다니는 것을 관찰하면 프로그래머의 의도대로 처리되는지를 확인할 수 있게 된다. 프로그램에서 raise에 의해 발생되는 예외상황이 아니고 수치 연산에 의한 Overflow 같이 다른 표현식에 의해 발생하는 예외상황에 대해서는 테스트 데이타를 생성하지 않는다.

1.1 관련 연구

테스트 데이타 생성에는 다음과 같은 세 가지 방법이 있다: 프로그램 내의 단위 구조(structural elements)에 대한 테스트 데이타를 만들어내는 방법[16, 17, 18, 19, 20, 21, 22, 23], 엄밀하게 정의된 문법(grammar)을 만족하는 테스트 데이타를 만들어내는 방법[24], 그리고 임의로 테스트 데이타를 만들어내는 방법[25]. 본 논문에서 제안하는 방법은 특정 수행 경로 (특정 예외상황을 발생시키는 수행 경로)에 대한 테스트 데이타를 생성하므로, 첫번째 방법에 속한다.

프로그램 내의 단위 구조에 대한 테스트 데이타를 만들어내는 방법[19, 20, 21, 22, 23]은 주로 요약 수행 방법(symbolic execution)을 사용해왔다. 프로그램을 직접 수행하지 않고 주어진 프로그램의 특정 수행 경로를 요약하여 따라가면서 프로그램 입력값에 대한 제약식을 만들어낸다. 대상 언어에 따라 분석이 만들어내는 제약식은, 변수값이 가져야 할 정수 범위(integer-intervals)[20] (Fortran), 변수 사이의 수치 관계(numeric formula)[21] (Fortran), 데이타 사이의 부등식[19] (Cobol-like language) 등이 된다. 요약 수행 방법과 다른 방법으로, Korel[16]은 동적으로 테스트 데이타를 생성하는 방법을 제안하였다. 주어진 프로그램에 대해 실제 프로그램의 수행을 관찰하다가, 프로그램이 정상적이지 않은 동작을 하게 되면 그렇게 만드는 입력값을 제외시켜 나가는 것이다.

우리가 알고 있는 한 예외상황을 발생시키는 테스트 데이타를 자동으로 생성하는 방법은 본 논문에서 처음으로 제안하는 것이다. 이 방법의 장점 중 한 가지는 테스트 데이타 생성 분석을 엄밀하게 다루었다는 것이다. 프로그램의 제약식을 만들어내는 방법을 정확하게 정의하고, 분석 방법의 보증성을 엄밀하게 증명하였다.

1.2 논문 구조

본 논문의 전체적인 구조는 다음과 같다. 본 논문에서 대상 언어로 사용하는 L 언어의 문법구조와 의미구조를 2장에서 정의한다. 3장에서는 테스트 데이타 생성 분석 방법을 제안하고 4장에서 분석의 보증성을 증명한다. 5장에서는 몇 가지 예를 사용하여 분석 방법을 설명하고, 6장에서 결론을 맺는다.

2. 언어 L

본 논문에서 다루는 언어 L은 함수를 인수나 결과값으로 전달하지 않는 일차 언어이고(first-order), ML 스타일의 예외상황 관리 방법을 제공하며, 적극적으로 값을 계산하는(call-by-value) 언어이다:

- 프로그램 내 표현식의 값은 함수가 될 수 없다. 따라서, 함수가 다른 함수의 인수나 결과값으로 전달되거나 변수에 할당될 수 없고, 프로그램 내의 모든 함수 호출 표현식에서 어떤 함수가 호출되는지를 프로그램 텍스트만 보고 바로 알아낼 수 있다.
- 데이타 값은 유한한 갯수의 데이타 컨스트럭터(constructor)로 만들어진다. 예를 들어, 자연수를 나타내는 데이타는 두 가지 방법으로 만들어질 수 있다. ZERO와 임의의 인수를 가지고 만들어지거나, SUC과 자연수 인수를 가지고 만들어진다.
- 예외상황은 인수를 가질 수 없다. L 언어에서 다루는 예외상황은 인수가 없이 예외상황 이름만으로 만들어진다.
- 예외상황은 함수의 인수나 결과값으로 전달될 수 없다. 따라서, 함수 호출의 결과값으로 예외상황을

e	::=	0	상수
		x	변수
		$(\lambda x.e_1)\ e_2$	무명함수 호출
		$(\mathtt{fix}\ f\ \lambda x.e_1)\ e_2$	재귀함수 호출
		$f\ e$	함수 호출
		$\kappa\ e$	데이타 생성
		$\kappa^{-1}\ e$	데이타 인수 얻기
		$\mathtt{case}\ e_1\ \kappa\ e_2\ e_3$	조건 분기문
		$\mathtt{raise}\ E$	예외상황 발생
		$\mathtt{handle}\ e_1\ E\ e_2$	예외상황 처리

그림 1 L언어의 문법 구조

전달하는 "raise f(0)"와 같은 것은 표현할 수 없다. 예외상황을 발생시킬 때에는 "raise ERROR"와 같이 예외상황 이름을 분명하게 나타내야 한다.

L 언어의 문법 구조는 그림 1에 나타나 있다. 설명의 편의성을 위해서, 문자열, 숫자, 메모리 관련 연산(메모리 할당, 참조) 등은 생략하였다.

프로그램 내의 데이타는 "$x\ e$"에 의해서 x라는 이름과 e라는 인수를 갖도록 만들어진다. 데이타의 인수는 "$x^{-1}\ e$"에 의해서 얻을 수 있다. 재귀함수 f는 "fix $f\ \lambda x.e$"에 의해서 정의된다. 조건 분기문 "case $e_1\ x\ e_2\ e_3$"는 e_1의 값이 x라는 이름의 데이타일 경우에 e_2로 분기하고, 그렇지 않을 때에는 e_3로 분기한다. 예외상황 E는 "raise E"에 의해서 발생된다. 예외상황 처리 표현식 "handle e_1 E e_2"는 e_1을 먼저 계산한다. e_1을 계산하는 도중에 예외상황이 발생하지 않으면 e_1의 값이 예외상황 처리 표현식의 결과값이 된다. e_1을 계산하는 도중에 예외상황 E가 발생하면 e_2의 값이 결과값이 되고, E가 아닌 다른 예외상황이 발생하면 그 예외상황은 처리되지 못하고 다시 발생된다. 여러 가지 예외상황을 처리하려면 예외상황 처리 표현식을 반복해서 (nested handle expressions) 사용하면 된다. 예를 들어서, handle (handle e_1 E e_2) F e_3는 두 가지 예외상황 E와 F를 처리할 수 있다.

본 논문에서는 앞으로,

- 모든 변수가 서로 다른 이름을 갖는다고 가정한다 (alpha-converted).
- 프로그램 내에 실행되지 않는 코드(dead code)는 없다고 가정한다.
- "L 프로그램"이라는 말로 입력 변수(free variables)가 있는 L 표현식 (expression)을 의미한다.

$$
\begin{array}{llcll}
\sigma & \in & Env & = & Var \xrightarrow{\text{fin}} Val & \text{환경} \\
v & \in & Val & = & \{0\} + RecFtn + Data & \text{값} \\
 & & RecFtn & = & Env \times Expr & \text{재귀함수} \\
\kappa v & \in & Data & = & Con \times Val & \text{데이타} \\
\kappa & \in & Con & & & \text{데이타 컨스트럭터} \\
E & \in & Exn & & & \text{예외상황 이름} \\
\hat{E} & \in & Packet & & & \text{발생한 예외상황}
\end{array}
$$

$$\text{[C-s]}\quad \sigma \vdash 0 \to 0 \qquad \text{[NV-s]}\quad \frac{\sigma(x) = v}{\sigma \vdash x \to v}$$

$$\text{[CONN-s]}\quad \frac{\sigma \vdash e \to v}{\sigma \vdash \kappa\ e \to \kappa\ v} \qquad \text{[DCONN-s]}\quad \frac{\sigma \vdash e \to \kappa\ v}{\sigma \vdash \kappa^{-1}\ e \to v}$$

$$\text{[APPN-1s]}\quad \frac{\sigma \vdash e_2 \to v \quad \sigma[x_n \mapsto v] \vdash [x_n/x]e_1 \to v' \ \text{새 번호 } n}{\sigma \vdash (\lambda x.e_1)\ e_2 \to v'}$$

$$\text{[APPN-2s]}\quad \frac{\sigma \vdash e_2 \to v \ \text{새 번호 } n \quad \sigma[f \mapsto \langle \sigma, \mathtt{fix}\ f\ \lambda x.e_1\rangle][x_n \mapsto v] \vdash [x_n/x]e_1 \to v'}{\sigma \vdash (\mathtt{fix}\ f\ \lambda x.e_1)\ e_2 \to v'}$$

$$\text{[APPN-3s]}\quad \frac{\sigma(f) = \langle \sigma', \mathtt{fix}\ f\ \lambda x.e_1\rangle \quad \sigma \vdash e_2 \to v \quad \text{새 번호 } n \quad \sigma'[f \mapsto \langle \sigma', \mathtt{fix}\ f\ \lambda x.e_1\rangle][x_n \mapsto v] \vdash [x_n/x]e_1 \to v'}{\sigma \vdash f\ e_2 \to v'}$$

$$\text{[CASEN-1s]}\quad \frac{\sigma \vdash e_1 \to \kappa\ v \quad \sigma \vdash e_2 \to v'}{\sigma \vdash \mathtt{case}\ e_1\ \kappa\ e_2\ e_3 \to v'}$$

$$\text{[CASEN-2s]}\quad \frac{\sigma \vdash e_1 \to \kappa'\ v \quad \sigma \vdash e_3 \to v'}{\sigma \vdash \mathtt{case}\ e_1\ \kappa\ e_2\ e_3 \to v'} \quad \kappa' \neq \kappa$$

$$\text{[HNDLN-1s]}\quad \frac{\sigma \vdash e_1 \to v}{\sigma \vdash \mathtt{handle}\ e_1\ E\ e_2 \to v}$$

$$\text{[HNDLN-2s]}\quad \frac{\sigma \vdash e_1 \to \hat{E} \quad \sigma \vdash e_2 \to v}{\sigma \vdash \mathtt{handle}\ e_1\ E\ e_2 \to v}$$

그림 2 언어 L의(예외상황을 발생시키지 않는)의미 구조

2.1 L 언어의 의미 구조

본 논문에서는 자연적 의미론 (natural semantics) [10]으로 L 언어의 의미 구조를 엄밀하게 정의한다. L 언어의 의미 구조는 그림 2, 3에 나와있다: 그림 2는 예외상황을 발생시키지 않는 정상적인 값을 계산하는 규칙을 나타내고 있고, 그림 3은 예외상황을 발생시키는 규칙을 나타내고 있다. 그림 2에 나와있는 것처럼, 예외상황 이름은 E로 표시하고 발생한 예외상황은 $\hat{E}$으로 표시하기로 한다.

언어 L의 의미 구조를 정의하는 데 사용된 $\sigma \vdash e \to v$ (또는 $\sigma \vdash e \to \hat{E}$)는 "$\sigma$라는 환경 하에서 표현식 e를 계산한 값이 v (또는 $\hat{E}$)"라는 것을 의미한다. 환경 σ는 프로그램 내의 각 변수에 해당하는 값을 매핑(mapping)해주는 유한 함수이다. 프로그램의 값은 상수

$$[\text{RS-s}] \quad \sigma \vdash \texttt{raise}\ \mathrm{E} \to \hat{\mathrm{E}}$$

$$[\text{CONX-s}] \quad \frac{\sigma \vdash e \to \hat{\mathrm{E}}}{\sigma \vdash \kappa\, e \to \hat{\mathrm{E}}} \qquad [\text{DCONX-s}] \quad \frac{\sigma \vdash e \to \hat{\mathrm{E}}}{\sigma \vdash \kappa^{-1}\, e \to \hat{\mathrm{E}}}$$

$$[\text{APPX-1s}] \quad \frac{\sigma \vdash e_2 \to \hat{\mathrm{E}}}{\sigma \vdash (\lambda x.e_1)\ e_2 \to \hat{\mathrm{E}}} \qquad [\text{APPX-2s}] \quad \frac{\sigma \vdash e_2 \to \hat{\mathrm{E}}}{\sigma \vdash f\ e_2 \to \hat{\mathrm{E}}}$$

$$[\text{APPX-3s}] \quad \frac{\sigma \vdash e_2 \to \hat{\mathrm{E}}}{\sigma \vdash (\texttt{fix}\ f\ \lambda x.e_1)\ e_2 \to \hat{\mathrm{E}}}$$

$$[\text{APPX-4s}] \quad \frac{\sigma \vdash e_2 \to v \qquad \sigma[x_n \mapsto v] \vdash [x_n/x]e_1 \to \hat{\mathrm{E}}\ \text{새 번호 } n}{\sigma \vdash (\lambda x.e_1)\ e_2 \to \hat{\mathrm{E}}}$$

$$[\text{APPX-5s}] \quad \frac{\sigma \vdash e_2 \to v \quad \text{새 번호 } n \qquad \sigma[f \mapsto \langle \sigma, \texttt{fix}\ f\ \lambda x.e_1\rangle][x_n \mapsto v] \vdash [x_n/x]e_1 \to \hat{\mathrm{E}}}{\sigma \vdash (\texttt{fix}\ f\ \lambda x.e_1)\ e_2 \to \hat{\mathrm{E}}}$$

$$[\text{APPX-6s}] \quad \frac{\sigma(f) = \langle \sigma', \texttt{fix}\ f\ \lambda x.e_1\rangle \quad \sigma \vdash e_2 \to v \quad \text{새 번호 } n \qquad \sigma'[f \mapsto \langle \sigma', \texttt{fix}\ f\ \lambda x.e_1\rangle][x_n \mapsto v] \vdash [x_n/x]e_1 \to \hat{\mathrm{E}}}{\sigma \vdash f\ e_2 \to \hat{\mathrm{E}}}$$

$$[\text{CASEX-1s}] \quad \frac{\sigma \vdash e_1 \to \hat{\mathrm{E}}}{\sigma \vdash \texttt{case}\ e_1\ \kappa\ e_2\ e_3 \to \hat{\mathrm{E}}}$$

$$[\text{CASEX-2s}] \quad \frac{\sigma \vdash e_1 \to \kappa\, v \quad \sigma \vdash e_2 \to \hat{\mathrm{E}}}{\sigma \vdash \texttt{case}\ e_1\ \kappa\ e_2\ e_3 \to \hat{\mathrm{E}}}$$

$$[\text{CASEX-3s}] \quad \frac{\sigma \vdash e_1 \to \kappa'\, v \quad \sigma \vdash e_3 \to \hat{\mathrm{E}}}{\sigma \vdash \texttt{case}\ e_1\ \kappa\ e_2\ e_3 \to \hat{\mathrm{E}}} \quad \kappa' \neq \kappa$$

$$[\text{HNDLX-1s}] \quad \frac{\sigma \vdash e_1 \to \hat{\mathrm{E}}'}{\sigma \vdash \texttt{handle}\ e_1\ \mathrm{E}\ e_2 \to \hat{\mathrm{E}}'} \quad \mathrm{E}' \neq \mathrm{E}$$

$$[\text{HNDLX-2s}] \quad \frac{\sigma \vdash e_1 \to \hat{\mathrm{E}} \quad \sigma \vdash e_2 \to \hat{\mathrm{E}}'}{\sigma \vdash \texttt{handle}\ e_1\ \mathrm{E}\ e_2 \to \hat{\mathrm{E}}'}$$

그림 3 언어 L의(예외상황을 발생시키는)의미 구조

0이거나 데이타, 또는 σ에서 정의된 함수 $\mathrm{fix}\, f\, \lambda x.e$를 나타내는 $\langle \sigma,\ \mathrm{fix}\, f\, \lambda x.e\rangle$이다. 일반적인 표기법을 사용하여, 유한 함수는 $f \in A \xrightarrow{fin} B$로 나타내고, $f[a \mapsto b]$는 a를 b로 매핑하고 f의 정의구역에서 a를 제외한 다른 모든 원소 a'은 $f(a')$으로 매핑하는 새로운 함수를 나타낸다.

예외상황을 발생시키지 않는 규칙의 예로, 그림 2에 있는 case 표현식에 대한 규칙을 살펴보기로 한다. 규칙 [CASEN-1s]는 e_1의 값이 x라는 이름의 데이타일 때 e_2가 계산된다는 것을 나타낸다:

$$[\text{CASEN}-1\text{s}] \quad \frac{\sigma \vdash e_1 \to x\, v \qquad \sigma \vdash e_2 \to v'}{\sigma \vdash \text{case}\ e_1\ x\ e_2\ e_3 \to v'}.$$

규칙 [CASEN-2s]는 e_1의 값이 x가 아닌 다른 이름의 데이타일 때 e_3가 계산된다는 것을 나타낸다:

$$[\text{CASEN}-2\text{s}] \quad \frac{\sigma \vdash e_1 \to x'\, v \qquad \sigma \vdash e_3 \to v'}{\sigma \vdash \text{case}\ e_1\ x\ e_2\ e_3 \to v'} \quad x' \neq x.$$

예외상황을 발생시키는 규칙에 대한 예로, 그림 3에 있는 handle 표현식에 대한 규칙을 다루기로 한다. 규칙 [HNDLX-1s]는 e_1에서 발생한 예외상황을 handle 표현식이 처리하지 못하는 경우이다:

$$[\text{HNDLX}-1\text{s}] \quad \frac{\sigma \vdash e_1 \to \hat{\mathrm{E}}'}{\sigma \vdash \text{handle}\ e_1\ \mathrm{E}\ e_2 \to \hat{\mathrm{E}}'} \quad \mathrm{E}' \neq \mathrm{E}.$$

규칙 [HNDLX-2s]는 e_1에서 발생한 예외상황을 e_2가 처리하지만, e_2에서 다시 예외상황을 발생시키는 경우이다:

$$[\text{HNDLX}-2\text{s}] \quad \frac{\sigma \vdash e_1 \to \hat{\mathrm{E}} \qquad \sigma \vdash e_2 \to \hat{\mathrm{E}}'}{\sigma \vdash \text{handle}\ e_1\ \mathrm{E}\ e_2 \to \hat{\mathrm{E}}'}.$$

3. 집합 제약식

본 논문에서 제안하는 분석은 주어진 프로그램의 각 표현식 e에 대해서 $se \rhd e : C$ 관계를 만들어내는 것이다. 집합식 se는 e의 값이 포함되어야 할 집합을 나타내고, C는 e가 se에 포함되기 위해 만족해야 할 제약식들(constraints)의 집합을 나타낸다. 따라서 "$se \rhd e : C$"의 의미는 "e의 값이 se에 포함되려면 C를 만족해야 한다"가 된다. 집합식(set expressions)과 제약식의 문법 구조와 의미 구조는 그림 4에 나와있다.

해석(interpretation) I는 집합 제약식 C (또는 집합식 se)가 *true*인지 *false* (또는 프로그램 값이나 발생한 예외상황)인지를 결정해준다. C에 있는 각 제약식 $X \subseteq se$와 $a \subseteq \mathrm{X}$에 대해서, $I(se)$와 $I(a)$가 정의되어 있고 $I(X) \subseteq I(se)$와 $I(a) \subseteq I(X)$를 각각 만족할 때, 해석 I는 C에 있는 모든 조건들을 만족시키는 *모델*(해(solution))이라고 한다. C의 가장 큰 모델을 $gm(C)$라고 적기로 한다.

본 논문에서 집합식 se는 발생한 예외상황이나 프로그램 값들의 집합을 나타내게 된다. 예를 들어, $x\, X$는 X에 있는 값들을 인수로 하고 x로 만들어지는 데이타를 나타내고:

$$I(x\, X) = \{x\, v \mid v \in I(X)\},$$

$x^{-1}\, X$는 X에 있는 값들 중 x로 만들어지는 데이타의 인수들을 나타낸다:

$$I(x^{-1}\, X) = \{v \mid x\, v \in I(X)\}.$$

분석에서 사용하는 집합 변수는 다음과 같이 세 가지

$$
\begin{array}{rcll}
v & \in & Val & = \{0\} + FtnExpr + Data \quad \text{값} \\
\lambda x.e & \in & FtnExpr & \text{함수 표현식} \\
\kappa\, v & \in & Data & = Con \times Val \quad \text{데이타} \\
\kappa,\ \kappa' & \in & Con & \text{데이타 컨스트럭터} \\
\mathrm{E},\ \mathrm{E}' & \in & Exn & \text{예외상황 이름} \\
\hat{\mathrm{E}},\ \hat{\mathrm{E}}' & \in & Packet & \text{발생한 예외상황}
\end{array}
$$

$$
\begin{array}{rcl@{\quad}l}
\mathcal{C} & ::= & \{\mathtt{true}\} & \text{전체집합} \\
 & | & \{\mathtt{false}\} & \text{집합변수} \\
 & | & \{\mathcal{X} \subseteq se\} & \kappa\text{로 만들어진 데이타 집합} \\
 & | & \{a \subseteq \mathcal{X}\} & \kappa\text{로 만들어진 데이타의 인수 집합} \\
 & | & \mathcal{C}_1 \cup \mathcal{C}_2 & \kappa\text{로 만들어지지 않은 데이타 집합}
\end{array}
\qquad
\begin{array}{rcl@{\quad}l}
se & ::= & \mathcal{U} & \text{전체집합} \\
 & | & \mathcal{X} & \text{집합변수} \\
 & | & \kappa\, \mathcal{X} & \kappa\text{로 만들어진 데이타 집합} \\
 & | & \kappa^{-1}\, \mathcal{X} & \kappa\text{로 만들어진 데이타의 인수 집합} \\
 & | & \bar{\kappa}\, \mathcal{X} & \kappa\text{로 만들어지지 않은 데이타 집합}
\end{array}
$$

$$
\begin{array}{rcl@{\quad}l}
a & ::= & 0 & a \quad \text{원소가 하나인 집합} \\
 & | & \lambda x.e & \text{발생한 예외상황} \\
 & | & \kappa\, a &
\end{array}
$$

$$
\begin{array}{lcl}
\mathcal{I}(\mathcal{C}) & \in & \{true,\ false\} \\
\mathcal{I}(\{\mathtt{true}\}) & = & true \\
\mathcal{I}(\{\mathtt{false}\}) & = & false \\
\mathcal{I}(\{\mathcal{X} \subseteq se\}) & = & \mathcal{I}(\mathcal{X}) \subseteq \mathcal{I}(se) \\
\mathcal{I}(\{a \subseteq \mathcal{X}\}) & = & \mathcal{I}(a) \subseteq \mathcal{I}(\mathcal{X}) \\
\mathcal{I}(\mathcal{C}_1 \cup \mathcal{C}_2) & = & \mathcal{I}(\mathcal{C}_1) \wedge \mathcal{I}(\mathcal{C}_2)
\end{array}
$$

$$
\begin{array}{lcl}
\mathcal{I}(se) & \subseteq & Val + Packet \\
\mathcal{I}(\mathcal{U}) & = & Val + Packet \\
\mathcal{I}(\mathcal{X}) & \subseteq & Val \\
\mathcal{I}(\kappa\, \mathcal{X}) & = & \{\kappa v \mid v \in \mathcal{I}(\mathcal{X})\} \\
\mathcal{I}(\kappa^{-1}\, \mathcal{X}) & = & \{v \mid \kappa v \in \mathcal{I}(\mathcal{X})\} \\
\mathcal{I}(\bar{\kappa}\, \mathcal{X}) & = & \{\kappa' v \mid v \in \mathcal{I}(\mathcal{X}),\ \kappa' \neq \kappa\} \\
\mathcal{I}(0) & = & \{0\} \\
\mathcal{I}(\lambda x.e) & = & \{\lambda x.e\} \\
\mathcal{I}(\kappa\, a) & = & \{\kappa v \mid v \in \mathcal{I}(a)\} \\
\mathcal{I}(\hat{\mathrm{E}}) & = & \{\hat{\mathrm{E}}\}
\end{array}
$$

그림 4 집합 제약식

로 분류할 수 있다: 1. 프로그램에 있는 각 변수 x의 값을 모아가는 X, 2. 프로그램에 있는 각 재귀 함수 f의 값을 나타내는 F, 3. 분석 과정 중에 새로 만들어내는 집합 변수이다.

3.1 분석 규칙

본 논문에서 제안하는 분석 규칙은 그림 5, 6에 나와 있다.

그림 5에 있는 상수 0에 대한 네 가지 규칙을 살펴보기로 한다. 0은 항상 집합 {0}에 포함되어 있다:

$$[\mathrm{C-1a}] \qquad 0 \rhd 0 : \{\mathtt{true}\}$$

0이 X에 포함되기 위해서 필요한 조건은 $\{0 \subseteq X\}$이다:

$$[\mathrm{C-2a}] \qquad X \rhd 0 : \{0 \subseteq X\}$$

0이 $x^{-1}X$에 포함되기 위해서는 X가 $x\ 0$를 포함해야 한다:

$$
\begin{array}{lll}
\text{[C-1a]} & 0 \rhd 0 : \{\mathtt{true}\} & \\
\text{[C-2a]} & \mathcal{X} \rhd 0 : \{0 \subseteq \mathcal{X}\} & \\
\text{[C-3a]} & \kappa^{-1}\, \mathcal{X} \rhd 0 : \{\kappa\, 0 \subseteq \mathcal{X}\} & \\
\text{[C-4a]} & se \rhd 0 : \{\mathtt{false}\} & se \notin \{0,\ \mathcal{X},\ \kappa^{-1}\, \mathcal{X}\}
\end{array}
$$

$$
\begin{array}{lll}
\text{[NVN-a]} & se \rhd x : \{\mathcal{X} \subseteq se\} & se \notin \{\hat{\mathrm{E}}\} \\
\text{[NVX-a]} & \hat{\mathrm{E}} \rhd x : \{\mathtt{false}\} &
\end{array}
$$

$$
\begin{array}{lll}
\text{[RSN-a]} & se \rhd \mathtt{raise\ E} : \{\mathtt{false}\} & se \notin \{\hat{\mathrm{E}}\} \\
\text{[RSX-1a]} & \hat{\mathrm{E}} \rhd \mathtt{raise\ E} : \{\mathtt{true}\} & \\
\text{[RSX-2a]} & \hat{\mathrm{E}}' \rhd \mathtt{raise\ E} : \{\mathtt{false}\} & \mathrm{E}' \neq \mathrm{E}
\end{array}
$$

$$\text{[CONN-1a]}\ \frac{\kappa^{-1}\, \mathcal{X} \rhd e : \mathcal{C}}{\mathcal{X} \rhd \kappa\, e : \mathcal{C}} \qquad \text{[CONN-2a]}\ \frac{\mathcal{X} \rhd e : \mathcal{C}}{\kappa\, \mathcal{X} \rhd \kappa\, e : \mathcal{C}}$$

$$\text{[CONN-3a]}\ \frac{\kappa^{-1}\, \mathcal{Y} \rhd e : \mathcal{C}}{\kappa'^{-1}\, \mathcal{X} \rhd \kappa\, e : \mathcal{C} \cup \{\mathcal{Y} \subseteq \kappa'^{-1}\, \mathcal{X}\}}\ \text{새 변수 } \mathcal{Y}$$

$$\text{[CONN-4a]}\ \frac{a \rhd e : \mathcal{C}}{\kappa\, a \rhd \kappa\, e : \mathcal{C}} \qquad \text{[CONX-a]}\ \frac{\hat{\mathrm{E}} \rhd e : \mathcal{C}}{\hat{\mathrm{E}} \rhd \kappa\, e : \mathcal{C}}$$

$$\text{[CONN-5a]}\ se \rhd \kappa\, e : \{\mathtt{false}\}\ se \notin \{\mathcal{X},\ \kappa\, \mathcal{X},\ \kappa'^{-1}\, \mathcal{X},\ \kappa\, a,\ \hat{\mathrm{E}}\}$$

$$\text{[DCONN-1a]}\ \frac{\kappa\, \mathcal{Y} \rhd e : \mathcal{C}}{\kappa'\, \mathcal{X} \rhd \kappa^{-1}\, e : \mathcal{C} \cup \{\mathcal{Y} \subseteq \kappa'\, \mathcal{X}\}}\ \text{새 변수 } \mathcal{Y}$$

$$\text{[DCONN-2a]}\ \frac{\kappa\, \mathcal{Y} \rhd e : \mathcal{C}}{\kappa'^{-1}\mathcal{X} \rhd \kappa^{-1}\, e : \mathcal{C} \cup \{\mathcal{Y} \subseteq \kappa'^{-1}\mathcal{X}\}}\ \text{새 변수 } \mathcal{Y}$$

$$\text{[DCONN-3a]}\ \frac{\kappa\, \mathcal{Y} \rhd e : \mathcal{C}}{\overline{\kappa'}\, \mathcal{X} \rhd \kappa^{-1}\, e : \mathcal{C} \cup \{\mathcal{Y} \subseteq \overline{\kappa'}\, \mathcal{X}\}}\ \text{새 변수 } \mathcal{Y}$$

$$\text{[DCONX-a]}\ \frac{\hat{\mathrm{E}} \rhd e : \mathcal{C}}{\hat{\mathrm{E}} \rhd \kappa^{-1}\, e : \mathcal{C}}$$

$$\text{[DCONN-4a]}\ \frac{\kappa\, se \rhd e : \mathcal{C}}{se \rhd \kappa^{-1}\, e : \mathcal{C}}\ se \notin \{\kappa'\, \mathcal{X},\ \kappa'^{-1}\, \mathcal{X},\ \overline{\kappa'}\, \mathcal{X},\ \hat{\mathrm{E}}\}$$

그림 5 분석 규칙(1)

$$[\mathrm{C-3a}] \qquad x^{-1}\, X \rhd 0 : \{x\ 0 \subseteq X\}.$$

0이 그 외의 다른 집합에 포함될 방법은 없다:

$$[\mathrm{C-4a}] \qquad se \rhd 0 : \{\,\mathrm{false}\} \ se \notin \{0,\ X,\ x^{-1}X\}.$$

전제(premise)와 결론(conclusion)이 있는 규칙의 예로 [CONN-1a]를 살펴보면:

$$[\mathrm{CONN-1a}] \qquad \frac{x^{-1}\, X \rhd e : C}{X \rhd x\, e : C},$$

$x\ e$의 값이 X에 포함되기 위해서 e의 값이 $x^{-1}X$에 포함되어야 함을 나타낸다.

그림 5에 있는 규칙 [CONN-3a], [DCONN-1a], [DCONN-2a], [DCONN-3a]는 새 변수를 만들어낸다. 다음을 예로 들면:

$$[\text{CONN}-3a]\ \frac{x^{-1}\,Y \rhd e : C}{x'^{-1}\,X \rhd x\ e : C \cup \{Y \subseteq x'^{-1}\,X\}}\ \text{새 변수 } Y.$$

$x\ e$가 $x'^{-1}\,X$에 포함되기 위해서는 e의 값이 $x^{-1}(x'^{-1}\,X)$에 포함되어야 한다. 그러나 그림 4에 나와 있는 집합식(se)의 문법 구조에 의해서 x^{-1} 다음에는 집합 변수만 올 수 있으므로, x^{-1} 다음에 $x'^{-1}\,X$가 올 수 없다. 이렇게 집합 제약식의 모양을 제한하는 이유는 집합 제약식의 해를 구하기 쉽게 하기 위해서이다. 집합 변수만 올 수 있는 곳에 집합 변수가 아닌 다른 집합식 se'이 올 경우에는, se'을 새로운 변수 Y로 바꾸고 se'이 Y를 포함하도록 만든다. 따라서, $x'^{-1}\,X$를 새 변수 Y로 바꾸고 집합 제약식 $\{Y \subseteq x'^{-1}\,X\}$를 추가한다.

그림 5에 있는 규칙과 달리, 그림 6에 있는 규칙은 주어진 se 와 e에 대해 적용 가능한 규칙이 여러 가지가 있다. 따라서, $se \rhd e : C$를 만족하는 제약식 C가 여러 가지로 만들어질 수 있다.

그림 6에 있는 규칙 [CASEX-a], [CASE-1a], [CASE-2a]를 살펴보기로 한다. case 표현식에서 예외상황이 발생한다면, 예외상황이 발생하는 위치에 따라 세 가지 가능성이 생긴다: e_1에서 발생하거나 e_2 또는 e_3에서 발생하는 경우이다. 규칙 [CASEX-a]는 e_1에서 예외상황이 발생하는 경우를 나타낸다:

$$[\text{CASEX}-a]\quad \frac{\widehat{\mathrm{E}} \rhd e_1 : C}{\widehat{\mathrm{E}} \rhd \text{case } e_1\ e_2\ e_3 : C}.$$

e_1이 예외상황 E를 발생하기 위해서 C가 필요하다면, case $e_1\ e_2\ e_3$가 예외상황 E를 발생하기 위해서도 C가 필요하다. 규칙 [CASE-1a]는 e_2에서 예외상황이 발생하는 경우를 나타낸다:

$$[\text{CASE}-1a]\quad \frac{x\ X \rhd e_1 : C_1 \quad se \rhd e_2 : C_2}{se \rhd \text{case } e_1\ x\ e_2\ e_3 : C_1 \cup C_2}\ \text{새 변수}$$

e_1이 $x\ X$에 포함되기 위해서 C_1이 필요하고 e_2가 예외상황 E를 발생하기 위해서 C_2가 필요하다면, case $e_1\ x\ e_2\ e_3$가 예외상황 E를 발생하기 위해서는 $C_1 \cup C_2$가 필요하다. 마지막으로 규칙 [CASE-2a]는 e_3에서 예외상황이 발생하는 경우를 나타낸다:

$$[\text{CASE}-2a]\quad \frac{\overline{x}\ X \rhd e_1 : C_1 \quad se \rhd e_2 : C_2}{se \rhd \text{case } e_1\ x\ e_2\ e_3 : C_1 \cup C_2}\ \text{새 변수 } X.$$

e_1이 $\overline{x}\ X$에 포함되기 위해서 C_1이 필요하고 e_3가 예외상황 E를 발생하기 위해서 C_2가 필요하다면, case $e_1\ x\ e_2\ e_3$가 예외상황 E를 발생하기 위해서는 $C_1 \cup C_2$가 필요하다.

그림 6에 있는 규칙 [APPX-2a]와 [APP-2a]는 함수 이름 f에 대한 제약식을 만든다. 다음을 예로 들면:

$$[\text{APP}-2a]\ \frac{X_n \rhd e_2 : C_1 \quad se \rhd [x_n/x]e_1 : C_2}{se \rhd \text{fix } f\ \lambda x.e_1 : C_1 \cup C_2 \cup \{F \subseteq \lambda x.e_1\}}\ \text{새 번호 } n,$$

이 규칙은 F의 값이 $\{\lambda x.e_1\}$에 포함되어야 한다는 제약식을 만든다. 각 함수 이름에 대한 제약식은 이 두 규칙에 의해서만 만들어지므로, 함수 이름에 대한 집합 변수 F는 항상 원소가 하나인 집합 $\{\lambda x.e_1\}$를 나타내게 된다. 프로그램 분석 중에 나타나는 모든 변수의 이름을 서로 다르게 하기 위해서, 함수 내용 e_1에 나타나는 x를 모두 새로운 변수 x_n으로 바꾼 후에 e_1을 분석한다: $[x_n/x]e_1$.

그림 6에 있는 재귀함수 호출에 대한 규칙 [APP-3a]는 무한한 횟수로 사용될 수 없다. 각각의 재귀함수 f에 대해서 $Used_f()$는 현재까지 [APP-3a] 규칙을 적용하여 f를 호출한 횟수를 기록한다. 함수 f를 호출한 횟수가 미리 정한 제한값 N을 넘으면 함수 f에 대해서는 이 규칙을 더 이상 사용할 수 없도록 한다. 다음을 예로 들면:

$$[\text{APP}-3a]\quad \frac{X_n \rhd e_2 : C_1 \quad se \rhd [x_n/x]e_1 : C_2}{se \rhd f\ e_2 : C_1 \cup C_2}\ \begin{array}{l}\text{프로그램 내의 fix } f\ \lambda x.e_1 \text{의}\\ \text{새 번호 } n, Used_f(n) \le N\end{array},$$

$Used_f() \le N$이라는 조건은 프로그램을 분석해가는 도중에 f의 호출 횟수를 N이하로 제한하게 된다. 재귀함수의 호출 횟수를 임의로 제한함으로써 가능한 테스트 데이타를 만들어내지 못할 수는 있지만, 테스트 데이타 생성 분석의 보증성(safety)을 위배하지는 않는다. 왜냐하면 테스트 데이타를 만들어내지 못하는 경우는 그 자체로 다음과 같은 테스트 데이타 생성 분석의 보증성(soundness) 조건을 만족하기 때문이다: 분석 결과로 얻은 테스트 데이타를 프로그램 입력으로 사용하여 프로그램을 수행시키면, 지정한 예외상황이 항상 발생한다.

규칙 [CASE-2a]와 [CASE-2a]는 e_1을 계산하여 얻은 데이타의 인수 값을 모으기 위해서 새 변수를 만들어낸다. 다음을 예로 들면:

$$[\text{CASE}-1a]\quad \frac{x\ X \rhd e_1 : C_1 \quad se \rhd e_2 : C_2}{se \rhd \text{case } e_1\ x\ e_2\ e_3 : C_1 \cup C_2}\ \text{새 변수 } X.$$

e_1이 $x\ X$에 포함되기 위해서 C_1이 필요하고 e_2가

se에 포함되기 위해서 C_2가 필요하다면, case e_1 x e_2 e_3가 se에 포함되기 위해서는 $C_1 \cup C_2$가 필요하다. 이 규칙에서 새로 만든 변수 X는 e_1을 계산하여 얻은 데이타의 인수 값을 모으게 된다.

3.2 집합 제약식을 만드는 방법

본 논문에서 제안하는 분석 방법은 입력으로 받은 프로그램 e에서 특정 예외상황 E가 발생한다는 조건을 시작으로 하여, 프로그램의 입력 값에 대한 제약식(constraints)을 만들어간다. 그림 5, 6에 나와있는 규칙 중에서 $\mathfrak{E}$과 e에 대해 적용 가능한 규칙을 적용해가면서 $\mathfrak{E} \triangleright e : C$를 만족하는 제약식의 집합 C를 만들어 가는 것이다.

3.3 집합 제약식의 해를 구하는 방법

집합 제약식의 해를 구하는 알고리즘은 집합 제약식의 성질에 따라 여러 가지[11, 12, 13, 14, 15]가 개발되어 있다. 일반적으로 널리 사용되고 있는 definite 집합 제약식[11, 12]은 최소모델 (minimal model)을 그 해로 갖게 되고, co-definite 집합 제약식[13, 14]은 최대모델(maximal model)을 해로 갖는다. 본 논문에서 다루는 집합 제약식은 Charatonik과 Podelski[13, 14]에 의해 정의된 *co-definite* 집합 제약식에 속하므로, 집합 제약식 C의 해가 존재한다면 가장 큰 모델 $gm(C)$를 갖게 된다.

본 논문에서 다루는 집합 제약식의 해는 반복적 부동점 방법(iterative fixpoint method)을 사용하여 구할 수 있다. 주어진 프로그램의 분석 결과가 C일 때, C에 있는 각 집합 변수 X의 값을 전체집합 U로 초기화한 후 C에 있는 각 집합 제약식에 따라 집합 변수들의 값을 계산한다. 모든 집합 변수들의 값이 더 이상 변하지 않을 때까지 반복해서 계산하면, 집합 제약식 C의 가장 큰 모델 $gm(C)$를 구하게 된다.

4. 보증성 (Soundness)

분석 $\mathfrak{E} \triangleright e : C$에 의해 얻은 테스트 데이타를 입력으로 하여 프로그램 e를 수행시키면 예외상황 $\mathfrak{E}$이 발생한다는 것을 증명한다. 증명에 들어가기 전에, 증명에 필요한 기호를 먼저 정의한다.

정의 1 ($\lfloor\ \mathfrak{E} \triangleright e : C\ \rfloor$) *프로그램 e에 대한 분석을 $\mathfrak{E} \triangleright e : C$라 할 때, 분석 $\mathfrak{E} \triangleright e : C$의 과정(derivation tree)을 $\lfloor\ \mathfrak{E} \triangleright e : C\ \rfloor$로 표시하기로 한다.*

정의 2 ($\lfloor\ \sigma \vdash e \rightarrow o\ \rfloor$) *프로그램 e에 대한 수행을 $\sigma \vdash e \rightarrow o$라 할 때, 수행 $\sigma \vdash e \rightarrow o$의 과정을 $\lfloor\!\lfloor\ \sigma \vdash e \rightarrow o\ \rfloor\!\rfloor$로 표시하기로 한다.*

보조정리 1 *프로그램 e_0의 입력 x_0의 값을 v로 하여 e_0를 수행한 결과가 $[x_0 \mapsto v] \vdash e_0 \rightarrow o$라고 하자. 수행 과정 $\lfloor\!\lfloor\ [x_0 \mapsto v] \vdash e_0 \rightarrow o\ \rfloor\!\rfloor$에 나타나는 모든 σ는 다음의 성질을 만족한다:*

$dom(\sigma)$에 있는 변수들의 값은 프로그램 수행 중에 덮어씌워지지 않는다.

Proof. 프로그램 e_0 안에 있는 변수들은 서로 다른 이름을 갖는다(alpha-converted). 프로그램 수행 중에 나타나는 변수는 프로그램의 입력 x_0나 함수 호출 표현식에서 만들어지는 변수 x_n뿐이다. x_0의 값은 초기환경 $[x_0 \mapsto v]$에 정의되어 있고 그 이후에는 변경되지 않는다. 그림 2, 3에 있는 함수 호출 표현식에 대한 의미 구조를 살펴보면, 함수가 호출될 때마다 매번 다른 이름의 변수를 함수 인수로 사용하여 σ에 그 값을 넣어놓는다. 따라서 프로그램 수행 중에 나타나는 모든 변수들은 그 변수가 만들어질 때 한 번만 σ에 그 값을 넣어놓고, 그 이후에는 σ에 있는 값을 바꾸지 않는다. 그러므로 $dom(\sigma)$에 있는 모든 변수들의 값은 프로그램 수행 중에 덮어씌워지지 않는다. □

정의 3 (σ^v) *프로그램 e_0의 입력 x_0의 값을 v로 하여 e_0를 수행한 결과가 $[x_0 \mapsto v] \vdash e_0 \rightarrow o$라고 하자. 다음과 같이 재귀적으로(recursive) 정의한 환경을 σ^v로 표시하기로 한다.*

수행 과정 $\lfloor\!\lfloor\ [x_0 \mapsto v] \vdash e_0 \rightarrow o\ \rfloor\!\rfloor$에 나타나는 모든 σ에 대해서,

(1) σ에 정의되어 있는 모든 $[x \mapsto v']$를 다 모으고

(2) σ에 정의되어 있는 모든 $[f \mapsto \langle \sigma', \mathrm{fix} f\, \lambda x.e \rangle]$를 $[f \mapsto \langle \sigma^v, \mathrm{fix} f\, \lambda x.e \rangle]$로 바꾸어서 모은다.

보조정리 2 *프로그램 e_0의 입력 x_0의 값을 v로 하여 e_0를 수행한 결과가 $[x_0 \mapsto v] \vdash e_0 \rightarrow o$라 하면, 수행 과정 $\lfloor\!\lfloor\ [x_0 \mapsto v] \vdash e_0 \rightarrow o\ \rfloor\!\rfloor$ 중에 σ^v는 변하지 않는다.*

Proof. 프로그램 e_0의 수행 중에 환경 σ^v가 변하는 경우는 함수 호출 표현식에 대한 경우밖에 없으므로, 각각의 함수 호출 표현식에 의해 변한 환경 σ가 σ^v와 같다는 것을 보이면 된다. 증명 중에 여러 번 사용되는 내용을 *이라 하여 미리 증명하기로 한다.

* 보조정리 1에 의해 $\sigma^v(x_0)$의 값은 프로그램 수행 중에 덮어씌워지지 않으므로 $\sigma^v(x_0) = v$이다. 프로그램 e_0의 입력은 x_0뿐이고 $[x_0 \mapsto v](x_0)$와 $\sigma^v(x_0)$는 같은 값을 가지므로, $\vdash$의 왼쪽에 있는 환경을 고려하지 않는다면 $\lfloor\!\lfloor\ [x_0 \mapsto v] \vdash e_0 \rightarrow o\ \rfloor\!\rfloor$와 $\lfloor\!\lfloor\ \sigma^v \vdash e_0 \rightarrow o\ \rfloor\!\rfloor$는 똑같다. 다시 말해서 $\lfloor\!\lfloor\ \sigma^v \vdash e_0 \rightarrow o\ \rfloor\!\rfloor$에 나타나는 모든 $\sigma \vdash e \rightarrow o$에 대해서

$\sigma' \vdash e \rightarrow \sigma'$ 이 $\lceil [x_0 \mapsto v] \vdash e_0 \rightarrow o \rceil$에 나타나고, σ^v의 정의에 의해 $dom(\sigma') \subseteq dom(\sigma^v)$이고 $dom(\sigma')$의 모든 변수 x에 대해 $\sigma'(x) = \sigma^v(x)$이다.

각각의 함수 호출 표현식에 대한 증명은 다음과 같다.

1. $(\lambda x.e_1)\ e_2$일 때

$$\frac{\sigma^v \vdash e_2 \rightarrow v_2 \quad \sigma^v[x_n \mapsto v_2] \vdash [x_n/x]e_1 \rightarrow \sigma'}{\sigma^v \vdash (\lambda x.e_1)e_2 \rightarrow \sigma'}$$ 이 $\lceil \sigma^v \vdash e_0 \rightarrow o \rceil$에 나타난다면, * 에서 보인 것처럼

$$\frac{\sigma \vdash e_2 \rightarrow v_2 \quad \sigma[x_n \mapsto v_2] \vdash [x_n/x]e_1 \rightarrow \sigma'}{\sigma \vdash (\lambda x.e_1)e_2 \rightarrow \sigma'}$$ 이 $\lceil [x_0 \mapsto v] \vdash e_0 \rightarrow o \rceil$에 나타나고, σ^v의 정의에 의해 $\sigma^v(x_n) = v_2$이고 $\sigma[x_n \mapsto v_2] = \sigma^v$이다. 따라서 $\sigma^v[x_n \mapsto v_2] = \sigma^v$이다.

2. $(\text{fix } f\lambda x.e_1)\ e_2$일 때

$$\frac{\sigma^v \vdash e_2 \rightarrow v_2 \quad \sigma^v[f \mapsto \langle \sigma^v, \text{fix } f\lambda x.e_1 \rangle][x_n \mapsto v_2] \vdash [x_n/x]e_1 \rightarrow \sigma'}{\sigma^v \vdash (\text{fix } f\lambda x.e_1)\ e_2 \rightarrow \sigma'}$$

이 $\lceil \sigma^v \vdash e_0 \rightarrow o \rceil$에 나타난다면, * 에서 보인 것처럼 $\lceil [x_0 \mapsto v] \vdash e_0 \rightarrow o \rceil$에

$$\frac{\sigma \vdash e_2 \rightarrow v_2 \quad \sigma[f \mapsto \langle \sigma, \text{fix } f\lambda x.e_1 \rangle][x_n \mapsto v_2] \vdash [x_n/x]e_1 \rightarrow \sigma'}{\sigma \vdash (\text{fix } f\lambda x.e_1)\ e_2 \rightarrow \sigma'}$$ 이 나타난다. 또한 σ^v의 정의에 의해 $\sigma^v(x_n) = \sigma[f \mapsto \langle \sigma, \text{fix } f\lambda x.e_1 \rangle][x_n \mapsto v_2](x_n) = v_2$이고 $\sigma^v(f)$는 $\sigma[f \mapsto \langle \sigma, \text{fix } f\lambda x.e_1 \rangle][x_n \mapsto v_2](f) = \langle \sigma, \text{fix } f\lambda x.e_1 \rangle$의 σ를 σ^v로 바꾼 $\langle \sigma^v, \text{fix } f\lambda x.e_1 \rangle$이다. 따라서 $\sigma^v[f \mapsto \langle \sigma^v, \text{fix } f\lambda x.e_1 \rangle][x_n \mapsto v_2] = \sigma^v$이다.

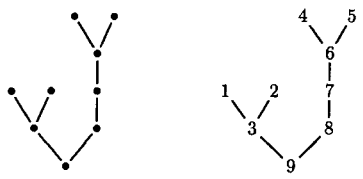
3. $f\ e_2$일 때 (f는 $\text{fix } f\lambda x.e_1$일 때)

$$\frac{\sigma^v(f) = \langle \sigma^v, \text{fix } f\lambda x.e_1 \rangle \quad \sigma^v \vdash e_2 \rightarrow v_2 \quad \sigma^v[f \mapsto \langle \sigma^v, \text{fix } f\lambda x.e_1 \rangle][x_n \mapsto v_2] \vdash [x_n/x]e_1 \rightarrow \sigma'}{\sigma^v \vdash f\ e_2 \rightarrow \sigma'}$$ 이 $\lceil \sigma^v \vdash e_0 \rightarrow o \rceil$에 나타나면, * 에 의해

$$\frac{\sigma(f) = \langle \sigma', \text{fix } f\lambda x.e_1 \rangle \quad \sigma \vdash e_2 \rightarrow v_2 \quad \sigma'[f \mapsto \langle \sigma', \text{fix } f\lambda x.e_1 \rangle][x_n \mapsto v_2] \vdash [x_n/x]e_1 \rightarrow \sigma'}{\sigma \vdash f\ e_2 \rightarrow \sigma'}$$ 이 $\lceil [x_0 \mapsto v] \vdash e_0 \rightarrow o \rceil$에 나타난다. 또한 σ^v의 정의에 의해 $\sigma^v(x_n) = \sigma'[f \mapsto \langle \sigma', \text{fix } f\lambda x.e_1 \rangle][x_n \mapsto v_2](x_n) = v_2$이고 $\sigma^v(f)$는 $\sigma'[f \mapsto \langle \sigma', \text{fix } f\lambda x.e_1 \rangle][x_n \mapsto v_2](f) = \langle \sigma', \text{fix } f\lambda x.e_1 \rangle$의 σ'을 σ^v로 바꾼 $\langle \sigma^v, \text{fix } f\lambda x.e_1 \rangle$이다. 따라서 $\sigma^v[f \mapsto \langle \sigma^v, \text{fix } f\lambda x.e_1 \rangle][x_n \mapsto v_2] = \sigma^v$이다. □

정의 4 (*Post(node)*) *유도 과정(derivation tree)을 (1) 왼쪽 부분 과정(subtree) (2) 오른쪽 부분 과정 (3) 루트노드의 순서로 돌아다니면서(postorder traversal) 각 노드 node에 붙인 번호를 Post(node)라 한다.*

예를 들어, 다음의 왼쪽 구조를 돌아다니면서 각 노드 *node*에 대해 *Post(node)*를 붙이면 오른쪽 구조와 같이 된다:



프로그램 e에 대한 수행을 $\sigma \vdash e \rightarrow o$라고 할 때, 수행 과정 $\lceil \sigma \vdash e \rightarrow o \rceil$의 각 노드 *node*에 붙인 *Post(node)*는 그 노드의 수행 순서를 나타낸다.

정의 5 ($e_0 \overset{0}{\sim} e$) *프로그램 e_0에 대한 분석을 $\hat{\mathbf{E}} \triangleright e_0 : C_0$라고 할 때, 분석 과정 $\lceil \hat{\mathbf{E}} \triangleright e_0 : C_0 \rceil$에서 루트노드 $\hat{\mathbf{E}} \triangleright e_0 : C_0$로부터 임의의 노드 $se \triangleright e : C$까지의 길이가 n인 경로(path) $e_0 \overset{n}{\sim} e$를 다음과 같이 정의한다:*

$$(1)\ e_0 \overset{0}{\sim} e_0 \qquad (2)\ \frac{e_0 \overset{n-1}{\sim} e' \quad \frac{\cdots se \triangleright e : C \cdots}{se' \triangleright e' : C'} \in [\hat{\mathbf{E}} \triangleright e_0 : C_0]}{e_0 \overset{n}{\sim} e}$$

프로그램 e_0에 대한 수행을 $\sigma \vdash e_0 \rightarrow o$라고 할 때, 수행 과정 $\lceil \sigma \vdash e \rightarrow o \rceil$에서 루트노드 $\sigma \vdash e_0 \rightarrow o$로부터 임의의 노드 $\sigma' \vdash e \rightarrow o'$까지의 길이가 n인 경로(path) $e_0 \overset{n}{\sim} e$도 비슷하게 정의한다.

보조정리 3 *x_0를 입력으로 갖는 프로그램 e_0의 분석을 $\hat{\mathbf{E}} \triangleright e_0 : C_0$라고 하면, $\lceil \hat{\mathbf{E}} \triangleright e_0 : C_0 \rceil$에 나타나는 모든 $se \triangleright e : C$는 다음 성질을 만족한다:*

임의의 $v \in gm(C_0)(X_0)$에 대해 (1) $\sigma^v \vdash e \rightarrow o'$이 $\lceil \sigma^v \vdash e_0 \rightarrow o \rceil$에 존재하고 (2) $o' \in gm(C_0)(se)$이다.

Proof. $I = gm(C_0)$라 하자. $\lceil \hat{\mathbf{E}} \triangleright e_0 : C_0 \rceil$에 나타나는 규칙 중, {false}를 만들어내는 규칙 ($se \triangleright e$: {false})에 대해서는 증명할 필요가 없다. 그 이유는 분석 과정 중에 {false}가 나타나면 그 제약식들의 해가 공집합이 되므로 보조정리 3이 바로 증명되기 때문이다. 분석 과정 $\lceil \hat{\mathbf{E}} \triangleright e_0 : C_0 \rceil$에 나타나는 노드 $se \triangleright e : C$의 $Post(se \triangleright e : C) \overset{\text{let}}{=} m$에 대한 귀납법(induction)으로 증명하기로 한다:

m=1인 경우 : *Post(node)*의 정의에 의해 그 값이 1인 경우는 유도 과정의 루트노드로부터 왼쪽 전제(premise)만을 따라온 잎노드(leaf)이므로, 표현식 e는 0, x, raise E 중 하나이다.

(1) $\sigma^v \vdash e \rightarrow o'$이 $\lceil \sigma^v \vdash e_0 \rightarrow o \rceil$에 존재한다는 것을 증

명해야 한다. $se \triangleright e : C$가 $\lfloor \widehat{\mathrm{E}} \triangleright e_0 : C_0 \rfloor$에 나타나므로, $e_0 \overset{n}{\sim} e$가 $\lfloor \widehat{\mathrm{E}} \triangleright e_0 : C_0 \rfloor$에 존재한다. 똑같은 $e_0 \overset{n}{\sim} e$가 $\lfloor \sigma^v \vdash e_0 \rightarrow o \rfloor$에도 존재함을 보여서 $\sigma^v \vdash e \rightarrow o'$이 $\lfloor \sigma^v \vdash e_0 \rightarrow o \rfloor$에 존재한다는 것을 보인다. $e_0 \overset{n}{\sim} e$의 n에 대한 귀납법으로 증명하기로 한다.

* n=0인 경우 : e가 e_0인 경우이므로, $e_0 \overset{0}{\sim} e_0$가 $\lfloor \sigma^v \vdash e_0 \rightarrow o \rfloor$에 존재한다.
* n=k ($k \geq 1$)인 경우 : $e_0 \overset{n'}{\sim} e'$ ($n' < k$)가 $\lfloor \widehat{\mathrm{E}} \triangleright e_0 : C_0 \rfloor$에 존재할 때 똑같은 $e_0 \overset{n'}{\sim} e'$ ($n' < k$)가 $\lfloor \sigma^v \vdash e_0 \rightarrow o \rfloor$에 존재한다면, $e_0 \overset{k}{\sim} e$가 $\lfloor \widehat{\mathrm{E}} \triangleright e_0 : C_0 \rfloor$에 존재할 때 똑같은 $e_0 \overset{k}{\sim} e$가 $\lfloor \sigma^v \vdash e_0 \rightarrow o \rfloor$에 존재함을 보여야한다. $e_0 \overset{k}{\sim} e$가 $\lfloor \widehat{\mathrm{E}} \triangleright e_0 : C_0 \rfloor$에 존재한다고 하자. $k \geq 1$이므로 $se \triangleright e : C$의 결론 $se' \triangleright e' : C'$이 $\lfloor \widehat{\mathrm{E}} \triangleright e_0 : C_0 \rfloor$에 존재하고, 따라서 $e_0 \overset{k-1}{\sim} e'$이 $\lfloor \widehat{\mathrm{E}} \triangleright e_0 : C_0 \rfloor$에 존재하므로 귀납법 가정에 의해 똑같은 $e_0 \overset{k-1}{\sim} e'$이 $\lfloor \sigma^v \vdash e_0 \rightarrow o \rfloor$에 존재한다. e_0로부터 유도 과정의 왼쪽 전제만을 따라온 경우이므로 e와 e'의 가능한 관계는 다음의 두 경우밖에 없다.
  - e'이 $x\ e$이거나 $x^{-1}\ e$인 경우

    $e_0 \overset{k-1}{\sim} e'$이 $\lfloor \sigma^v \vdash e_0 \rightarrow o \rfloor$에 존재하므로 $\sigma^v \vdash e' \rightarrow o''$이 수행 과정에 존재하고, 의미 구조에 의해 $\sigma^v \vdash e' \rightarrow o''$의 하나뿐인 전제 $\sigma^v \vdash e \rightarrow o'$이 항상 수행 과정에 존재하므로 $e_0 \overset{k}{\sim} e$가 $\lfloor \sigma^v \vdash e_0 \rightarrow o \rfloor$에 존재한다.
  - e'이 $(\lambda x.e_1)\ e$, $(\mathrm{fix}\ f\ \lambda x.e_1)\ e$, $f\ e$ ($\mathrm{fix}\ f\ \lambda x.e_1$일 때), $\mathrm{case}\ e\ x\ e_1\ e_2$, $\mathrm{handle}\ e\ \mathrm{E}\ e_1$ 중 한 가지인 경우

    $e_0 \overset{k-1}{\sim} e'$이 $\lfloor \sigma^v \vdash e_0 \rightarrow o \rfloor$에 존재하므로 $\sigma^v \vdash e' \rightarrow o''$이 수행 과정에 존재하고, 의미 구조에 의해 $\sigma^v \vdash e' \rightarrow o''$의 왼쪽 전제인 $\sigma^v \vdash e \rightarrow o'$이 항상 수행 과정에 존재하므로, $e_0 \overset{k}{\sim} e$가 $\lfloor \sigma^v \vdash e_0 \rightarrow o \rfloor$에 존재한다.

(2) $\sigma^v \vdash e \rightarrow o'$이 $\lfloor \sigma^v \vdash e_0 \rightarrow o \rfloor$에 존재함을 보였고, 이제 $o' \in gm(C_0)(se)$라는 것을 증명해야 한다. 각 표현식에 대하여 증명하기로 한다.

[C-1a] $0 \triangleright 0 : \{\mathrm{true}\}$

(1)에 의해 $\sigma^v \vdash 0 \rightarrow 0$이 성립하므로 $0 \in \{0\}$이다.

[C-2a] $X \triangleright 0 : \{0 \subseteq X\}$

(1)에 의해 $\sigma^v \vdash 0 \rightarrow 0$이 성립하고 $\{0 \subseteq X\} \subseteq C_0$이므로 $0 \in I(X)$이다.

[C-3a] $x^{-1}\ X \triangleright 0 : \{x\ 0 \subseteq X\}$

(1)에 의해 $\sigma^v \vdash 0 \rightarrow 0$이 성립하고 $\{x\ 0 \subseteq X\} \subseteq C_0$이므로 $x\ 0 \in I(X)$이다. 따라서 $0 \in I(x^{-1}\ X)$이다.

[NVN-a] $se \triangleright x : \{X \subseteq se\}$ $se \notin \{\widehat{\mathrm{E}}\}$

*Post(node)*가 1인 경우는 유도 과정의 루트노드로부터 왼쪽 전제만을 따라온 경우이고, 새로운 변수는 함수 호출 규칙의 오른쪽 전제에서만 만들어지므로 x는 프로그램의 입력인 x_0이다. 보조정리 2에 의해 $\sigma^v(x_0)$는 프로그램 수행 중에 덮어씌워지지 않으므로 $\sigma^v(x_0) = v$이다. $v \in I(X_0)$이고 $\{X_0 \subseteq se\} \subseteq C_0$이므로 $v \in I(se)$이다.

[RSX-1a] $\widehat{\mathrm{E}} \triangleright \mathrm{raise\ E} : \{\mathrm{true}\}$

(1)에 의해 $\sigma^v \vdash \mathrm{raise\ E} \rightarrow \widehat{\mathrm{E}}$이 성립하므로 $\widehat{\mathrm{E}} \in \{\widehat{\mathrm{E}}\}$이다.

m=k ($k > 1$)인 경우 : 분석 과정 $\lfloor \widehat{\mathrm{E}} \triangleright e_0 : C_0 \rfloor$에 나타나는 $se \triangleright e : C$에 대해서 다음을 증명한다: $Post(se' \triangleright e' : C')$이 k 미만인 $se' \triangleright e' : C'$에 대해서 $\sigma^v \vdash e' \rightarrow o''$이 $\lfloor \sigma^v \vdash e_0 \rightarrow o \rfloor$에 존재하고 $o'' \in I(se')$이면, (1) $\sigma^v \vdash e \rightarrow o'$이 $\lfloor \sigma^v \vdash e_0 \rightarrow o \rfloor$에 존재하고 (2) $o' \in I(se)$이다.

(1) $\sigma^v \vdash e \rightarrow o'$이 $\lfloor \sigma^v \vdash e_0 \rightarrow o \rfloor$에 존재한다는 것을 증명해야 한다. $se \triangleright e : C$가 $\lfloor \widehat{\mathrm{E}} \triangleright e_0 : C_0 \rfloor$에 나타나므로, $e_0 \overset{n}{\sim} e$가 $\lfloor \widehat{\mathrm{E}} \triangleright e_0 : C_0 \rfloor$에 존재한다. 똑같은 $e_0 \overset{n}{\sim} e$가 $\lfloor \sigma^v \vdash e_0 \rightarrow o \rfloor$에도 존재함을 보여서 $\sigma^v \vdash e \rightarrow o'$이 $\lfloor \sigma^v \vdash e_0 \rightarrow o \rfloor$에 존재한다는 것을 보인다. $e_0 \overset{n}{\sim} e$의 n에 대한 귀납법으로 증명하기로 한다.

* n=0인 경우 : e가 e_0인 경우이므로, $e_0 \overset{0}{\sim} e_0$가 $\lfloor \sigma^v \vdash e_0 \rightarrow o \rfloor$에 존재한다.
* n=k ($k \geq 1$)인 경우 : $e_0 \overset{n'}{\sim} e'$ ($n' < k$)가 $\lfloor \widehat{\mathrm{E}} \triangleright e_0 : C_0 \rfloor$에 존재할 때 똑같은 $e_0 \overset{n'}{\sim} e'$ ($n' < k$)가 $\lfloor \sigma^v \vdash e_0 \rightarrow o \rfloor$에 존재한다면, $e_0 \overset{k}{\sim} e$가 $\lfloor \widehat{\mathrm{E}} \triangleright e_0 : C_0 \rfloor$에 존재할 때 똑같은 $e_0 \overset{k}{\sim} e$가 $\lfloor \sigma^v \vdash e_0 \rightarrow o \rfloor$에 존재함을 보여야한다.

$e_0 \overset{k}{\sim} e$가 $\lfloor \widehat{\mathrm{E}} \triangleright e_0 : C_0 \rfloor$에 존재한다고 하자. $k \geq 1$이므로 $se \triangleright e : C$의 결론 $se' \triangleright e' : C'$이 $\lfloor \widehat{\mathrm{E}} \triangleright e_0 : C_0 \rfloor$에 존재하고, 따라서 $e_0 \overset{k-1}{\sim} e'$이 $\lfloor \widehat{\mathrm{E}} \triangleright e_0 : C_0 \rfloor$에 존재하므로 귀납법 가정에 의해 똑같은 $e_0 \overset{k-1}{\sim} e'$이 $\lfloor \sigma^v \vdash e_0 \rightarrow o \rfloor$에 존재한다. e와 e'의 가능한 각 경우에 대하여 증명하기로 한다. 각 함수 호출 표현식에서 새로

만들어내는 변수의 이름은 의미 구조에서 만드는 이름과 분석 규칙에서 만드는 이름이 서로 같다고 가정한다.

- e' 이 $x\ e$ 이거나 $x^{-1}\ e$ 인 경우

$e_0 \overset{k-1}{\sim} e'$ 이 $\lfloor \sigma^v \vdash e_0 \rightarrow o \rfloor$ 에 존재하므로 $\sigma^v \vdash e' \rightarrow o''$ 이 수행 과정에 존재하고, 의미 구조에 의해 $\sigma^v \vdash e' \rightarrow o''$ 의 하나뿐인 전제 $\sigma^v \vdash e \rightarrow o'$ 이 항상 수행 과정에 존재하므로 $e_0 \overset{k}{\sim} e$ 가 $\lfloor \sigma^v \vdash e_0 \rightarrow o \rfloor$ 에 존재한다.

- e' 이 $(\lambda x.e_1)\ e$, (fix $f\ \lambda x.e_1$) e, $f\ e$ (fix $f\ \lambda x.e_1$ 일 때), case $e\ x\ e_1\ e_2$, handle e E e_1 중 한 가지인 경우

$e_0 \overset{k-1}{\sim} e'$ 이 $\lfloor \sigma^v \vdash e_0 \rightarrow o \rfloor$ 에 존재하므로 $\sigma^v \vdash e' \rightarrow o''$ 이 수행 과정에 존재하고, 의미 구조에 의해 $\sigma^v \vdash e' \rightarrow o''$ 의 왼쪽 전제인 $\sigma^v \vdash e \rightarrow o'$ 이 항상 수행 과정에 존재하므로, $e_0 \overset{k}{\sim} e$ 가 $\lfloor \sigma^v \vdash e_0 \rightarrow o \rfloor$ 에 존재한다.

- e' 이 $(\lambda x.e_2)\ e_1$, (fix $f\ \lambda x.e_2$) e_1, $f\ e_1$ (fix $f\ \lambda x.e_2$ 일 때) 중 하나이고 e 가 $[x_n/x]e_2$ 인 경우

$X_n \rhd e_1 : C_1$ 이 $se' \rhd e' : C'$ 의 왼쪽 전제이므로 $Post(X_n \rhd e_1 : C_1)$ 이 $Post(se \rhd e : C)$ 보다 작다. 따라서 *Post(node)*에 대한 귀납법 가정에 의해 $\sigma^v \vdash e_1 \rightarrow v_1$ 이 $\lfloor \sigma^v \vdash e_0 \rightarrow o \rfloor$ 에 존재하고 $v_1 \in I(X_n)$이다. $e_0 \overset{k-1}{\sim} e'$ 이 $\lfloor \sigma^v \vdash e_0 \rightarrow o \rfloor$ 에 존재하므로 $\sigma^v \vdash e' \rightarrow o''$ 이 수행 과정에 존재하고, e_1을 수행하는 중에 예외상황이 발생하지 않으므로 의미 구조에 의해 $\sigma^v \vdash e \rightarrow o'$ 이 수행 과정에 존재한다. 따라서 $e_0 \overset{k}{\sim} e$ 가 $\lfloor \sigma^v \vdash e_0 \rightarrow o \rfloor$ 에 존재한다.

- e' 이 case $e_1\ x\ e\ e_2$ 인 경우

$x\ X \rhd e_1 : C_1$ 이 $se' \rhd e' : C'$ 의 왼쪽 전제이므로 $Post(x\ X \rhd e_1 : C_1)$ 이 $Post(se \rhd e : C)$ 보다 작다. 따라서 *Post(node)*에 대한 귀납법 가정에 의해 $\sigma^v \vdash e_1 \rightarrow v_1$ 이 $\lfloor \sigma^v \vdash e_0 \rightarrow o \rfloor$ 에 존재하고 $v_1 \in I(x\ X)$이다. $e_0 \overset{k-1}{\sim} e'$ 이 $\lfloor \sigma^v \vdash e_0 \rightarrow o \rfloor$ 에 존재하므로 $\sigma^v \vdash e' \rightarrow o''$ 이 수행 과정에 존재하고, e_1의 값이 x로 시작하므로 의미 구조에 의해 $\sigma^v \vdash e \rightarrow o'$ 이 수행 과정에 존재한다. 따라서 $e_0 \overset{k}{\sim} e$가 $\lfloor \sigma^v \vdash e_0 \rightarrow o \rfloor$ 에 존재한다.

- e' 이 case $e_1\ x\ e_2\ e$ 인 경우

$\overline{x}\ X \rhd e_1 : C_1$ 이 $se' \rhd e' : C'$ 의 왼쪽 전제이므로 $Post(\overline{x}\ X \rhd e_1 : C_1)$ 이 $Post(se \rhd e : C)$ 보다 작다. 따라서 *Post(node)*에 대한 귀납법 가정에 의해 $\sigma^v \vdash e_1 \rightarrow v_1$ 이 $\lfloor \sigma^v \vdash e_0 \rightarrow o \rfloor$ 에 존재하고 $v_1 \in I(\overline{x}\ X)$이다. $e_0 \overset{k-1}{\sim} e'$ 이 $\lfloor \sigma^v \vdash e_0 \rightarrow o \rfloor$ 에 존재하므로 $\sigma^v \vdash e' \rightarrow o''$ 이 수행 과정에 존재하고, e_1의 값이 x로 시작하지 않으므로 $\sigma^v \vdash e \rightarrow o'$ 이 수행 과정에 존재한다. 따라서 $e_0 \overset{k}{\sim} e$가 $\lfloor \sigma^v \vdash e_0 \rightarrow o \rfloor$ 에 존재한다.

- e' 이 handle e_1 E e 인 경우

$\widehat{\mathrm{E}} \rhd e_1 : C_1$ 이 $se' \rhd e' : C'$ 의 왼쪽 전제이므로 $Post(\widehat{\mathrm{E}} \rhd e_1 : C_1)$ 이 $Post(se \rhd e : C)$ 보다 작다. 따라서 *Post(node)*에 대한 귀납법 가정에 의해 $\sigma^v \vdash e_1 \rightarrow o_1$ 이 $\lfloor \sigma^v \vdash e_0 \rightarrow o \rfloor$ 에 존재하고 $o_1 \in I(\widehat{\mathrm{E}})$이다. $e_0 \overset{k-1}{\sim} e'$ 이 $\lfloor \sigma^v \vdash e_0 \rightarrow o \rfloor$ 에 존재하므로 $\sigma^v \vdash e' \rightarrow o''$ 이 수행 과정에 존재하고, e_1을 수행하는 중에 예외상황 E가 발생하므로 $\sigma^v \vdash e \rightarrow o'$ 이 수행 과정에 존재한다. 따라서 $e_0 \overset{k}{\sim} e$가 $\lfloor \sigma^v \vdash e_0 \rightarrow o \rfloor$ 에 존재한다.

(2) $se \rhd e : C$ 에 대하여 $\sigma^v \vdash e \rightarrow o'$ 이 $\lfloor \sigma^v \vdash e_0 \rightarrow o \rfloor$ 에 존재함을 보였고, 이제 $o' \in gm(C_0)(se)$라는 것을 증명해야 한다. 몇 가지 대표적인 경우에 대하여 증명하고, 비슷하게 증명되는 나머지 경우에 대하여는 부록 A에서 증명하기로 한다.

[C-1a] $0 \rhd 0 : \{\mathrm{true}\}$

(1)에 의해 $\sigma^v \vdash 0 \rightarrow 0$ 이 성립하므로 $0 \in \{0\}$ 이다.

[NVN-a] $se \rhd x : \{X \subseteq se\}\quad se \notin \{\widehat{\mathrm{E}}\}$

① x가 프로그램의 입력 x_0일 때

보조정리 2에 의해 $\sigma^v(x_0)$는 프로그램 수행 중에 덮어씌워지지 않으므로 $\sigma^v(x_0) = v$이다. $v \in I(X_0)$이고 $\{X_0 \subseteq se\} \subseteq C_0$이므로 $v \in I(se)$이다.

② x가 가 함수 호출 표현식에서 만들어진 변수 x_n 일 때

$$\frac{X_n \rhd e_2 : C_1 \quad se_n \rhd [x_n/x]e_3 : C_2}{se_n \rhd e_1\ e_2 : C_3}$$ (e_1이 $\lambda x.e_3$이거나 fix $f\ \lambda x.e_3$ 또는 f일 때)이 x_n을 만들어내는 규칙이라 하자. $se \rhd e : C$는 오른쪽 전제인 $se_n \rhd [x_n/x]e_3 : C_2$ 의 분석 과정 중에 나타나므로, *Post(node)*의 정의에 의해 왼쪽 전제의 $Post(X_n \rhd e_2 : C_1)$은 k미만이다. 따라서 귀납법 가정에 의해, $\sigma^v \vdash e_2 \rightarrow v_2$가 $\lfloor \sigma^v \vdash e_0 \rightarrow o \rfloor$ 에 존재하고 $v_2 \in I(X_n)$이다. 함수 호출 표현식의 의미 구조에 의해 $\sigma^v[x_n \mapsto v_2] \vdash [x_n/x]e_3 \rightarrow o'$ 이 수행 과정에 존재하므로, σ^v의 정의에 의해 $\sigma^v(x_n) = v_2$이다. $v_2 \in I(X_n)$이고 $\{X_n \subseteq se\} \subseteq C_0$이므로 $v_2 \in I(se)$이다.

[RSX-1a] $\widehat{\mathrm{E}} \rhd$ raise E : {true}

(1)에 의해 $\sigma^v \vdash$ raise E $\rightarrow \widehat{\mathrm{E}}$ 이 성립하므로 $\widehat{\mathrm{E}} \in \{\widehat{\mathrm{E}}\}$ 이다.

$$[\mathrm{CONN\text{-}1a}]\ \frac{x^{-1}\ X \rhd e : C}{X \rhd x\ e : C}$$

*Post(node)*의 정의에 의해 전제의 $Post(x^{-1}\ X \rhd e : C)$는 k미만이므로, 가정에 의해서 $\sigma^v \vdash e \rightarrow v'$ 이 성립하고 $v' \in I(x^{-1}\ X)$이다. 따라서 $\frac{\sigma^v \vdash e \rightarrow v'}{\sigma^v \vdash x\ e \rightarrow x\ v'}$ 이 성립하고 $x\ v' \in I(X)$ 이다.

[DCONN-1a] $\frac{x\ Y \rhd e : C}{x'\ X \rhd x^{-1} e : C \cup \{Y \subseteq x'\ X\}}$ 새 변수 Y

가정에 의해서 $\sigma^v \vdash e \rightarrow v'$ 이 성립하고 $v' \in I(x\ Y)$ 이다. $v' = x\ v''$ 이라 하면 $\frac{\sigma^v \vdash e \rightarrow x\ v''}{\sigma^v \vdash x^{-1} e \rightarrow v''}$ 이 성립한다. $v' \in I(x\ Y)$ 이면 $x^{-1} v' \in I(Y)$ 이고 $\{Y \subseteq x'\ X\} \subseteq C_0$ 이므로, $v'' \in I(x'\ X)$ 이다.

[APP-1a] $\frac{X_n \rhd e_2 : C_1 \quad se \rhd [x_n/x]e_1 : C_2}{se \rhd (\lambda x.e_1)\ e_2 : C_1 \cup C_2}$ 새 번호 n

가정에 의해서, $\sigma^v \vdash e_2 \rightarrow v_2$ 와 $\sigma^v \vdash [x_n/x]e_1 \rightarrow o'$ 이 성립하고 $v_2 \in I(X_n)$ 이며 $o' \in I(se)$ 이다. 보조정리 2에 의해 σ^v는 프로그램 수행 중에 변하지 않으므로 $\sigma^v[x_n \mapsto v_2] = \sigma^v$ 이다. 따라서 $\frac{\sigma^v \vdash e_2 \rightarrow v_2 \quad \sigma^v[x_n \mapsto v_2] \vdash [x_n/x]e \rightarrow o'}{\sigma^v \vdash (\lambda x.e_1)\ e_2 \rightarrow o'}$ 이 성립하고, $o' \in I(se)$이다.

[CASE-1a] $\frac{x\ X \rhd e_1 : C_1 \quad se \rhd e_2 : C_2}{se \rhd \text{case}\ e_1\ x\ e_2\ e_3 : C_1 \cup C_2}$ 새 변수 X

가정에 의해서 $\sigma^v \vdash e_1 \rightarrow v_1$, $\sigma^v \vdash e_2 \rightarrow o'$ 이 성립하고 $v_1 \in I(x\ X)$, $o' \in I(se)$ 이다. $v_1 = x\ v'$ 이라 하면, $\frac{\sigma^v \vdash e_1 \rightarrow x\ v' \quad \sigma^v \vdash e_2 \rightarrow o'}{\sigma^v \vdash \text{case}\ e_1\ x\ e_2\ e_3 \rightarrow o'}$ 이 성립하고 $o' \in I(se)$ 이다.

[HNDL-a] $\frac{\widehat{\mathrm{E}} \rhd e_1 : C_1 \quad se \rhd e_2 : C_2}{se \rhd \text{handle}\ e_1\ \mathrm{E}\ e_2 : C_1 \cup C_2}$

가정에 의해서 $\sigma^v \vdash e_1 \rightarrow o_1$, $\sigma^v \vdash e_2 \rightarrow o'$ 이 성립하고 $o_1 \in \{\widehat{\mathrm{E}}\}$ 이며 $o' \in I(se)$ 이다. 따라서 $\frac{\sigma^v \vdash e_1 \rightarrow \widehat{\mathrm{E}} \quad \sigma^v \vdash e_2 \rightarrow o'}{\sigma^v \vdash \text{handle}\ e_1\ \mathrm{E}\ e_2 \rightarrow o'}$ 이 성립하고 $o' \in I(se)$ 이다. □

정리 1 (보증성) *x_0를 입력으로 갖는 프로그램 e_0의 분석을 $\widehat{\mathrm{E}} \rhd e_0 : C_0$라고 하면, 임의의 $v \in gm(C_0)(X_0)$에 대해서 $[x_0 \mapsto v] \vdash e_0 \rightarrow \widehat{\mathrm{E}}$ 이 성립한다.*

Proof. 보조정리 3에 의해 (1) $\sigma^v \vdash e_0 \rightarrow o$가 $|\sigma^v \vdash e_0 \rightarrow o|$에 존재하고 (2) $o \in gm(C_0)(\widehat{\mathrm{E}}) = \{\widehat{\mathrm{E}}\}$이므로, $\sigma^v \vdash e_0 \rightarrow \widehat{\mathrm{E}}$이 성립한다. 보조정리 2에 의해 $\sigma^v(x_0)$는 프로그램 수행 중에 덮어씌워지지 않으므로 $\sigma^v(x_0) = v$이다. 프로그램 e_0의 입력이 x_0뿐이고 $[x_0 \mapsto v](x_0)$와 $\sigma^v(x_0)$는 같은 값을 가지므로, $[x_0 \mapsto v]$만을 가지고 e_0를 실행시켜도 σ^v를 가지고 실행시키는 것과 같은 결과를 만들어낸다. 그러므로 $[x_0 \mapsto v] \vdash e_0 \rightarrow \widehat{\mathrm{E}}$이 성립한다. □

5. 예

몇 가지 예를 사용하여, 주어진 프로그램에 대한 분석이 어떻게 이루어지는지를 설명한다. 언어 L에서는 데이타 타입 선언(datatype)을 제공하지 않지만, 예로 든 프로그램에서 다루는 데이타를 명확히 하기 위하여 데이타 타입 선언을 사용하기로 한다.

5.1 예 1

그림 6에는 주어진 표현식에 대해 적용 가능한 규칙이 여러 가지가 존재하므로, 주어진 프로그램에 대한 유도 과정도 여러 가지가 존재할 수 있다. 첫번째 예에서는 다음 프로그램에 대해 만들어질 수 있는 모든 유도 과정을 보여준다. 다음은 입력값 x가 A라는 이름의 데이타일 경우에 예외상황 E를 발생시키고, 그렇지 않을 때는 입력값을 결과값으로 하는 프로그램이다.

```
datatype t = A of t | B of t | C of 0
case x A (raise E) x
```

case 표현식이 예외상황 E를 발생시키기 위해서 적용 가능한 규칙은 [CASEX-a], [CASE-1a], [CASE-2a]이다.

• 가능한 유도 과정:

$$\frac{\widehat{\mathrm{E}} \rhd x : \{\text{false}\}}{\widehat{\mathrm{E}} \rhd \text{case}\ x\ \mathrm{A}\ (\text{raise E})\ x : \{\text{false}\}}$$

이 유도 과정은 [CASEX-a] 규칙을 적용하는 경우이다. x가 예외상황 E를 발생시킬 수 없으므로 {false}라는 제약식이 만들어진다.

• 또 다른 유도 과정:

$$\frac{\mathrm{A}\ Y \rhd x : \{X \subseteq \mathrm{A}\ Y\} \quad \widehat{\mathrm{E}} \rhd \text{raise E} : \{\text{true}\}}{\widehat{\mathrm{E}} \rhd \text{case}\ x\ \mathrm{A}\ (\text{raise E})\ x : \{X \subseteq \mathrm{A}\ Y, \text{true}\}}$$

이 유도 과정은 [CASE-1a] 규칙을 적용하는 경우이다. x가 A Y에 포함되기 위해서는 X가 A Y에 포함되어야 한다. raise E는 항상 예외상황 $\widehat{\mathrm{E}}$를 발생시키므로 {true}라는 제약식이 만들어진다. 전체 프로그램에 대한 제약식은 다음과 같고:

$$\{X \subseteq \mathrm{A}\ Y, \text{true}\},$$

아무런 제약이 없는 변수의 가장 큰 해는 전체집합 U이므로, 위 제약식의 가장 큰 해는 $X = \mathrm{A}\ U$이다. 따라서, 프로그램의 입력값 x에 A라는 이름의 데이타를 대입하여 프로그램을 수행시키면 예외상황 E가 발생하게 된다.

• 마지막으로 가능한 유도 과정:

$$\frac{\overline{\mathrm{A}\ Y} \rhd x : \{X \subseteq \overline{\mathrm{A}\ Y}\} \quad \widehat{\mathrm{E}} \rhd x : \{\text{false}\}}{\widehat{\mathrm{E}} \rhd \text{case}\ x\ \mathrm{A}\ (\text{raise E})\ x : \{X \subseteq \overline{\mathrm{A}\ Y}, \text{false}\}}$$

이 유도 과정은 [CASE-2a] 규칙을 적용하는 경우이다. x가 $\overline{\mathrm{A}}\ Y$에 포함되기 위해서는 X가 $\overline{\mathrm{A}}\ Y$에 포함되어야 한다. x는 예외상황을 E를 발생시킬 수 없으므로 {false}라는 제약식이 만들어진다.

5.2 예 2

다음 프로그램에서 함수 f는 입력값이 Zero라는 이름의 데이타인 경우에는 예외상황 E를 발생하고, 입력값이 Suc이라는 이름의 데이타인 경우에는 입력값보다 하나 작은 수를 입력값으로 하여 자기 자신을 다시 호출하고, 그 외의 경우에는 입력값을 결과값으로 한다.

이 예에서는 재귀함수 호출이 한번만 이루어지는 유도 과정을 살펴보기로 한다. (규칙 [APP-3a]에서의 N을 1로 정한다.)

```
datatype n = Suc of n | Zero of 0
                          e1
(fix f λx. case x Zero (raise E) (f (Suc⁻¹ x)) ) y
                                   e2
```

주어진 프로그램에 그림 6에 있는 규칙 [APP-2a]를 적용하면 다음과 같다:

$$\frac{X \rhd y : \{Y \subseteq X\} \quad \widehat{\mathrm{E}} \rhd e_1 : C_1}{\widehat{\mathrm{E}} \rhd (\mathrm{fix}\ f\ \lambda x.e_1)\ y : \{Y \subseteq X\} \cup C_1},$$

$\widehat{\mathrm{E}} \rhd e_1 : C_1$에 규칙 [CASE-2a]를 적용하면 다음과 같고:

$$\frac{\overline{\mathtt{Zero}}\ \mathcal{Z} \rhd x : \{\mathcal{X} \subseteq \overline{\mathtt{Zero}}\ \mathcal{Z}\} \quad \hat{\mathrm{E}} \rhd e_2 : C_2}{\hat{\mathrm{E}} \rhd \underbrace{\mathtt{case}\ x\ \mathtt{Zero}\ (\mathtt{raise\ E})\ e_2}_{e_1} : C_1 = \{\mathcal{X} \subseteq \overline{\mathtt{Zero}}\ \mathcal{Z}\} \cup C_2},$$

$\widehat{\mathrm{E}} \rhd e_2 : C_2$에 규칙 [APP-3a]와 [DCONN-4a], [NVN-a]를 적용하면 다음과 같다:

$$\frac{\dfrac{\mathtt{Suc}\ \mathcal{V} \rhd x : \{\mathcal{X} \subseteq \mathtt{Suc}\ \mathcal{V}\}}{\mathcal{V} \rhd \mathtt{Suc}^{-1}\ x : \{\mathcal{X} \subseteq \mathtt{Suc}\ \mathcal{V}\}} \quad \hat{\mathrm{E}} \rhd [v/x]e_1 : C_3}{\hat{\mathrm{E}} \rhd \underbrace{f\ (\mathtt{Suc}^{-1}\ x)}_{e_2} : C_2 = \{\mathcal{X} \subseteq \mathtt{Suc}\ \mathcal{V}\} \cup C_3}.$$

$\widehat{\mathrm{E}} \rhd [v/x]\, e_1 : C_3$에 규칙 [CASE-1a]와 [NVN-a], [RSX-1a]를 적용한 유도 과정은 다음과 같다:

$$\frac{\mathtt{Zero}\ \mathcal{W} \rhd v : \{\mathcal{V} \subseteq \mathtt{Zero}\ \mathcal{W}\} \quad \hat{\mathrm{E}} \rhd \mathtt{raise\ E} : \{\mathtt{true}\}}{\hat{\mathrm{E}} \rhd \underbrace{\mathtt{case}\ v\ \mathtt{Zero}\ (\mathtt{raise\ E})\ [v/x]e_2}_{[v/x]e_1} : C_3 = \{\mathcal{V} \subseteq \mathtt{Zero}\ \mathcal{W}, \mathtt{true}\}}.$$

따라서 전체 프로그램에 대한 제약식은 다음과 같고:

$$\{Y \subseteq X,\ X \subseteq \overline{\mathtt{Zero}}\ Z,\ X \subseteq \mathtt{Suc}\ V,\ V \subseteq \mathtt{Zero}\ W,\ \mathtt{true}\},$$

위 제약식의 가장 큰 해는 다음과 같다:

$$Y = \mathtt{Suc}\,(\mathtt{Zero}\ U),\ X = \mathtt{Suc}\,(\mathtt{Zero}\ U),\ Z = U,\ W = U,\ V = \mathtt{Zero}\ U.$$

테스트 데이타 Suc (Zero U)를 입력으로 하여 프로그램을 수행시키면 재귀함수 f는 한 번만 호출되고 프로그램은 예외상황 E를 발생시킨다.

6. 결 론

본 논문에서는 프로그램 내의 예외상황을 모두 발생시키는 테스트 데이타를 자동으로 생성해주는 분석 방법을 제안하였다. 테스트 데이타는 집합 제약식을 이용하여 만들어진다. 프로그램의 구조를 나타내는 구문트리(syntax tree)와 프로그램에서 특정 예외상황이 발생한다는 초기 조건을 가지고, 구문트리의 루트노드(root)부터 잎노드(leaves)를 향하여 분석해가면서 프로그램 변수들에 대한 제약식을 만들어간다. 테스트 데이타 생성 분석에서 사용하는 집합 제약식은 이미 존재하는 알고리즘을 사용하여 그 해를 구할 수 있다[14].

집합 제약식을 만들어가는 것은 프로그램 수행의 한 경우를 거꾸로 추적해가면서 프로그램 내의 각 표현식에 대한 필요조건(a necessary-condition)을 만들어가는 것이다. 프로그램 수행을 따라가다가 여러 가지 수행이 가능한 경우에는, 단 하나의 수행만을 따라간다. 따라서 여러 가지 수행 가능성을 하나의 수행으로 뭉뚱그릴 때 어쩔 수 없이 포함되는 잘못된 값들(approximation) ("반드시(must)" 정보가 아니라 "아마도(may)" 정보)은 만들어내지 않는다.

집합 제약식을 만들어가는 것이 프로그램 수행의 각 경우를 따라가는 것이므로 가능한 모든 수행을 다 따라가야겠지만, 대부분의 수행은 미리 제외될 수 있을 것이다. 그 이유는 첫째로, 예외상황을 발생시키는 표현식이 프로그램 내에 많지 않아서, 대부분의 제약식들은 모순(contradiction)을 만들어내어 고려할 필요가 없어질 것이기 때문이다. 둘째로, 정해진 예외상황을 발생시키는 입력값을 하나만 찾으면 더 이상의 분석을 할 필요가 없기 때문이다. 마지막으로, 제약식을 만들어내는 데에는 입력으로 받은 프로그램의 크기에 비례하는 시간만을 필요로 하기 때문이다. 이것은 재귀함수 호출에 대해서 함수 내용을 따라가는 횟수를 제한하기 때문이다.

분석 결과로 얻은 값을 입력으로 하여 프로그램을 수

행시키면 지정한 예외상황이 꼭 발생한다는 것을 엄밀하게 증명하였다. 재귀함수 호출 횟수를 임의로 제한함으로 해서 가능한 테스트 데이타를 찾아내지 못할 수도 있게 된다. 하지만, 테스트 데이타를 찾지 못하더라도 분석의 보증성 (분석으로 얻은 테스트 데이타를 입력으로 하여 프로그램을 수행시키면 정해진 예외상황이 발생한다)은 여전히 성립한다.

6.1 테스트 데이타 생성 분석을 실제적인 함수형 언어에 적용

본 논문에서 제안한 분석 방법을 실제적인 함수형 언어에 적용하려면, 다음과 같은 문제점들을 고려해야 한다:

- 실제적인 함수형 언어에서는 함수가 다른 함수의 인수나 결과값으로 전달될 수 있고 메모리에 저장될 수도 있는 등, 다른 값들과 차이없이 사용된다. 그러므로 테스트 데이타 생성 분석을 하기 전에 프로그램의 함수 흐름 분석이 먼저 이루어져야 한다. 테스트 데이타 생성 분석이 각 표현식에 대해 꼭 필요한 제약식만을 만들어내기 위해서는, 함수 흐름 분석도 꼭 필요한 정보만을 모아야 한다. 즉, 함수값이 확실한 것만을 모아야 한다.
- 실제적인 함수형 언어에서는 발생되기 전의 예외상황이 함수의 인수나 결과값으로 전달될 수 있고 메모리에 저장되었다가 꺼내서 사용될 수 있는 등, 다른 값들과 차이없이 사용된다. 또한 예외상황은 함수나 다른 예외상황을 비롯한 값을 인수로 가질 수도 있다. 그러므로 테스트 데이타 생성 분석은 예외상황 값의 흐름을 다른 값의 흐름과 함께 표현할 수 있어야 한다. 이러한 확장은 예외상황 분석[6]에서와 마찬가지로 이루어질 수 있을 것이다.

참 고 문 헌

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

[2] Luca Cardelli, J. Donahue, Michael Jordan, B. Kalsow, and Greg Nelson. The modula-3 type system. In *ACM Symposium on Principles of Programming Languages*, pages 202-212, Austin, TX, January 1989.

[3] James Gosling, Bill Joy, and Guy L. Jr. Steele. *The Java Language Specification (Java Series).* Addison-Wesley, September 1996.

[4] A. Nico Habermann and Dewayne E. Perry. *Ada for Experienced Programmers.* Addison-Wesley, 1983.

[5] Ariane 5: Flight 501 Failure. http://www.esrin.esa.it/htdocs/tidc/Press/Press96/ariane5rep.html, July 1996.

[6] Kwangkeun Yi and Sukyoung Ryu. Towards a cost-effective estimation of uncaught exceptions in SML programs. In *Lecture Notes in Computer Science*, volume 1302, pages 98-113. Springer-Verlag, proceedings of the 4th international static analysis symposium edition, 1997.

[7] Kwangkeun Yi and Sukyoung Ryu. A cost-effective esimtation of uncaught exceptions in Standard ML programs. *Theoretical Computer Science*, (invited submission).

[8] François Pessaux and Xavier Leroy. Type-based analysis of uncaught exceptions. In *ACM Symposium on Principles of Programming Languages*, pages 276-290, January 1999.

[9] Kwangkeun Yi and Sukyoung Ryu. SML/NJ Exception Analyzer 0.98. http://compiler.kaist.ac.kr/pub/exna/exna-README.html, December 1998.

[10] G. Kahn. Natural semantics. In K. Fuchi and M. Nivaat, editors, *Programming of Future Generation Computers*, pages 237-257. Elsevier Science Publishers (North-Holland), 1988.

[11] Nevin Heintze and Joxan Jaffar. A finite presentation theorem for approximating logic programs. Technical Report IBM Technical Report RC 16089 (# 71415), IBM, August 1990.

[12] Nevin Heintze and Joxan Jaffar. A decision procedure for a class of set constraints. Technical Report CMU-CS-91-110, Carnegie-Mellon University, February 1991.

[13] Witold Charatonik and Andreas Podelski. Solving set constraints for greatest models. Technical Report MPI-I-97-2-004, Max-Planck-Institut fur Informatik, April 1997.

[14] 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.

[15] P. Devienne, J. M. Talbot, and S. Tison. Solving classes of set constraints with tree automata. Technical Report IT-303, Laboratoire d'Informatique Fondamentale de Lille, May 1997.

[16] Bokdan Korel. Automated software test data generation. *IEEE Transactions on Software Engineering*, 16(8):870-879, August 1990.

[17] Richard A. DeMillo and A. Jefferson Offutt. Constraint-based automatic test data generation. *IEEE Transactions on Software Engineering*, 17(9):900-910, September 1991.

[18] A. Jefferson Offutt, Zhenyi Jin, and Jie Pan. The dynamic domain reduction approach to test data

generation. *Software Practice and Experience*, 1999.

[19] Jānis Bičevskis, Juris Borzovs, Uldis Straujums, Andris Zarinš, and JR. Edward F. Miller. SMOTL - a system to construct samples for data processing program debugging. *IEEE Transactions on Software Engineering*, 5:60-66, January 1979.

[20] Lori A. Clarke. A system to generate test data and symbolically execute programs. *IEEE Transactions on Software Engineering*, 2(3):215-222, September 1976.

[21] William E. Howden. Symbolic testing and the DISSECT symbolic evaluation system. *IEEE Transactions on Software Engineering*, 4(4):266-278, 1977.

[22] C. V. Ramamoorthy, Siu bun F. Ho, and W. T. Chen. On the automated generation of program test data. *IEEE Transactions on Software Engineering*, 2(4):293-300, December 1976.

[23] Juris Borzovs, Audris Kalninš, and Inga Medvedis. Automatic construction of test sets: Practical approach. volume 502 of *Lecture Notes in Computer Science*, pages 360-432. 1991.

[24] Peter M. Maurer. Generating testing data with enhanced context-free grammars. IEEE Software, 7(4), July 1990.

[25] Jeffrey Voas, Larry Morell, and Keith Miller. Predicting where faults can hide from testing. *IEEE Software*, 8(2):41-58, March 1991.

A 부록: 보조정리 3의 증명

4장에 있는 보조정리 3의 증명에서 빠진 부분이다.

[C-2a] $X \rhd 0 : \{0 \subseteq X\}$

(1)에 의해 $\sigma^v \vdash 0 \rightarrow 0$ 이 성립하고 $\{0 \subseteq X\} \subseteq C_0$ 이므로 $0 \in I(X)$ 이다.

[C-3a] $x^{-1}X \rhd 0 : \{x\ 0 \subseteq X\}$

(1)에 의해 $\sigma^v \vdash 0 \rightarrow 0$ 이 성립하고 $\{x\ 0 \subseteq X\} \subseteq C_0$ 이므로 $x\ 0 \in I(X)$ 이다. 따라서 $0 \in I(x^{-1}X)$ 이다.

[CONN-2a] $\dfrac{X \rhd e : C}{x\ X \rhd x\ e : C}$

가정에 의해서, $\sigma^v \vdash e \rightarrow v'$ 이 성립하고 $v' \in I(X)$ 이다. 따라서 $\dfrac{\sigma^v \vdash e \rightarrow v'}{\sigma^v \vdash x\ e \rightarrow x\ v'}$ 이 성립하고 $x\ v' \in I(x\ X)$ 이다.

[CONN-3a] $\dfrac{x^{-1}\ Y \rhd e : C}{x'^{-1}X \rhd x\ e : C \cup \{Y \subseteq x'^{-1}X\}}$ 새 변수 Y

가정에 의해서, $\sigma^v \vdash e \rightarrow v'$ 이 성립하고 $v' \in I(x^{-1}\ Y)$ 이다. 따라서 $\dfrac{\sigma^v \vdash e \rightarrow v'}{\sigma^v \vdash x\ e \rightarrow x\ v'}$ 이 성립한다. $v' \in I(x^{-1}\ Y)$ 이면 $x\ v' \in I(Y)$ 이고 $\{Y \subseteq x'^{-1}X\} \subseteq C_0$ 이므로, $x\ v' \in I(x'^{-1}X)$ 이다.

[CONN-4a] $\dfrac{a \rhd e : C}{x\ a \rhd x\ e : C}$

가정에 의해서, $\sigma^v \vdash e \rightarrow v'$ 이 성립하고 $v' \in I(a)$ 이다. 따라서 $\dfrac{\sigma^v \vdash e \rightarrow v'}{\sigma^v \vdash x\ e \rightarrow x\ v'}$ 이 성립하고 $x\ v' \in I(x\ a)$ 이다.

[CONX-a] $\dfrac{\widehat{E} \rhd e : C}{\widehat{E} \rhd x\ e : C}$

가정에 의해서, $\sigma^v \vdash e \rightarrow o'$ 이 성립하고 $o' \in \{\widehat{E}\}$ 이다. 따라서 $\dfrac{\sigma^v \vdash e \rightarrow \widehat{E}}{\sigma^v \vdash x\ e \rightarrow \widehat{E}}$ 이 성립하고 $\widehat{E} \in \{\widehat{E}\}$ 이다.

[DCONN-2a] $\dfrac{x\ Y \rhd e : C}{x'^{-1}X \rhd x^{-1}e : C \cup \{Y \subseteq x'^{-1}X\}}$ 새 변수 Y

가정에 의해서, $\sigma^v \vdash e \rightarrow v'$ 이 성립하고 $v' \in I(x\ Y)$ 이다. $v' = x\ v''$ 이라 하면 $\dfrac{\sigma^v \vdash e \rightarrow x\ v''}{\sigma^v \vdash x^{-1}e \rightarrow v''}$ 이 성립한다. $v' \in I(x\ Y)$ 이면 $x^{-1}v' \in I(Y)$ 이고 $\{Y \subseteq x'^{-1}X\} \subseteq C_0$ 이므로, $v'' \in I(x'^{-1}X)$ 이다.

[DCONN-3a] $\dfrac{x\ Y \rhd e : C}{\overline{x'}\ X \rhd x^{-1}e : C \cup \{Y \subseteq \overline{x'}\ X\}}$ 새 변수 Y

가정에 의해서, $\sigma^v \vdash e \rightarrow v'$ 이 성립하고 $v' \in I(x\ Y)$ 이다. $v' = x\ v''$ 이라 하면 $\dfrac{\sigma^v \vdash e \rightarrow x\ v''}{\sigma^v \vdash x^{-1}e \rightarrow v''}$ 이 성립한다. $v' \in I(x\ Y)$ 이면 $x^{-1}v' \in I(Y)$ 이고 $\{Y \subseteq \overline{x'}\ X\} \subseteq C_0$ 이므로, $v'' \in I(\overline{x'}\ X)$ 이다.

[DCONX-a] $\dfrac{\widehat{E} \rhd e : C}{\widehat{E} \rhd x\ e : C}$

가정에 의해서, $\sigma^v \vdash e \rightarrow o'$ 이 성립하고 $o' \in \{\widehat{E}\}$ 이다. 따라서 $\dfrac{\sigma^v \vdash e \rightarrow \widehat{E}}{\sigma^v \vdash x^{-1}e \rightarrow \widehat{E}}$ 이 성립하고 $\widehat{E} \in \{\widehat{E}\}$ 이다.

[DCONN-4a]

$$\frac{x\ se \rhd e : C}{se \rhd x^{-1}e : C} \quad se \notin \{x'\ X, x'^{-1}\ X, \overline{x'}\ X, \widehat{E}\}$$

가정에 의해서, $\sigma^v \vdash e \rightarrow v'$ 이 성립하고 $v' \in I(x\ se)$ 이다. $v' = x\ v''$ 이라 하면 $\dfrac{\sigma^v \vdash e \rightarrow x\ v''}{\sigma^v \vdash x^{-1}e \rightarrow v''}$ 이 성립한다. $x\ v'' \in I(x\ se)$ 이므로 $v'' \in I(se)$ 이다.

[APPX-1a] $\dfrac{\widehat{E} \rhd e_2 : C}{\widehat{E} \rhd (\lambda x.e_1)\ e_2 : C}$

가정에 의해서, $\sigma^v \vdash e_2 \rightarrow o'$ 이 성립하고 $o' \in \{\widehat{E}\}$ 이다. 따라서 $\dfrac{\sigma^v \vdash e_2 \rightarrow \widehat{E}}{\sigma^v \vdash (\lambda x.e_1)\ e_2 \rightarrow \widehat{E}}$ 이 성립하고 $\widehat{E} \in \{\widehat{E}\}$ 이다.

[APPX-2a] $\dfrac{\widehat{E} \rhd e_2 : C}{\widehat{E} \rhd (\text{ fix } f\lambda x.e_1)\ e_2 : C \cup \{F \subseteq \lambda x.e_1\}}$

가정에 의해서, $\sigma^v \vdash e_2 \rightarrow o'$ 이 성립하고 $o' \in \{\widehat{E}\}$ 이다. 따라서 $\dfrac{\sigma^v \vdash e_2 \rightarrow \widehat{E}}{\sigma^v \vdash (\text{ fix } f\lambda x.e_1)\ e_2 \rightarrow \widehat{E}}$ 이 성립하고 $\widehat{E} \in \{\widehat{E}\}$ 이다.

[APPX-3a] $\dfrac{\widehat{E} \rhd e_2 : C}{\widehat{E} \rhd f\ e_2 : C}$ 프로그램 내의 fix $f\lambda x.e_1$

가정에 의해서, $\sigma^v \vdash e_2 \rightarrow o'$ 이 성립하고 $o' \in \{\widehat{E}\}$ 이다. 따라서 $\dfrac{\sigma^v \vdash e_2 \rightarrow \widehat{E}}{\sigma^v \vdash f\ e_2 \rightarrow \widehat{E}}$ 이 성립하고 $\widehat{E} \in \{\widehat{E}\}$ 이다.

[APP-1a] $\dfrac{X_n \rhd e_2 : C_1 \quad se \rhd [x_n/x]e_1 : C_2}{se \rhd (\lambda x.e_1)\ e_2 : C_1 \cup C_2}$ 새 번호 n

가정에 의해서, $\sigma^v \vdash e_2 \rightarrow v_2$ 와 $\sigma^v \vdash [x_n/x]e_1 \rightarrow o'$ 이 성립하고 $v_2 \in I(X_n)$ 이며 $o' \in I(se)$ 이다. 보조정리 2에 의해 σ^v 는 프로그램 수행 중에 변하지 않으므로 $\sigma^v[x_n \mapsto v_2] = \sigma^v$ 이다. 따라서

$\dfrac{\sigma^v \vdash e_2 \rightarrow v_2 \quad \sigma^v[x_n \mapsto v_2] \vdash [x_n/x]e_1 \rightarrow o'}{\sigma^v \vdash (\lambda x.e_1)\ e_2 \rightarrow o'}$ 이 성립하고,

$o' \in I(se)$ 이다.

[APP-2a] $\dfrac{X_n \rhd e_2 : C_1 \quad se \rhd [x_n/x]e_1 : C_2}{se \rhd (\ \mathrm{fiix}\, f\lambda x.e_1)\ e_2 : C_1 \cup C_2\{F \subseteq \lambda x.e_1\}}$ 새 번호 n

가정에 의해서, $\sigma^v \vdash e_2 \rightarrow v_2$ 와 $\sigma^v \vdash [x_n/x]e_1 \rightarrow o'$ 이 성립하고 $v_2 \in I(X_n)$ 이며 $o' \in I(se)$ 이다. 보조정리 2에 의해 σ^v 는 프로그램 수행 중에 변하지 않으므로

$\sigma^v[f \mapsto \langle \sigma^v,\ \mathrm{fix}\, f\lambda x.e_1 \rangle][x_n \mapsto v_2] = \sigma^v$ 이다. 따라서

$\dfrac{\sigma^v \vdash e_2 \rightarrow v_2}{\sigma^v[f \mapsto \langle \sigma^v,\ \mathrm{fix}\, f\lambda x.e_1 \rangle][x_n \mapsto v_2] \vdash [x_n/x]e_1 \rightarrow o' \quad \sigma^v \vdash (\ \mathrm{fix}\, f\lambda x.e_1)\ e_2 \rightarrow o'}$ 이 성립하고,

$o' \in I(se)$ 이다.

[APP-3a]

$\dfrac{X_n \rhd e_2 : C_1 \quad se \rhd [x_n/x]e_1 : C_2}{se \rhd f\ e_2 : C_1 \cup C_2}$ 프로그램 내의 $\mathrm{fix}\, f\lambda x.e_1$, 새 번호 n, $Used_f() \le N$

가정에 의해서, $\sigma^v \vdash e_2 \rightarrow v_2$ 와 $\sigma^v \vdash [x_n/x]e_1 \rightarrow o'$ 이 성립하고 $v_2 \in I(X_n)$ 이며 $o' \in I(se)$ 이다. 보조정리 2에 의해 σ^v 는 프로그램 수행 중에 변하지 않으므로

$\sigma^v[f \mapsto \langle \sigma^v,\ \mathrm{fix}\, f\lambda x.e_1 \rangle][x_n \mapsto v_2] = \sigma^v$ 이다. 따라서

$\dfrac{\sigma^v(f) = \langle \sigma^v,\ \mathrm{fix}\, f\lambda x.e_1 \rangle \quad \sigma^v \vdash e_2 \rightarrow v_2}{\sigma^v[f \mapsto \langle \sigma^v,\ \mathrm{fix}\, f\lambda x.e_1 \rangle][x_n \mapsto v_2] \vdash [x_n/x]e_1 \rightarrow o' \quad \sigma^v \vdash (\ \mathrm{fix}\, f\lambda x.e_1)\ e_2 \rightarrow o'}$ 이 성립하고,

$o' \in I(se)$ 이다.

[CASEX-a] $\dfrac{\widehat{\mathrm{E}} \rhd e_1 : C}{\widehat{\mathrm{E}} \rhd \mathrm{case}\ e_1\ x\ e_2\ e_3 : C}$

가정에 의해서, $\sigma^v \vdash e_2 \rightarrow o'$ 이 성립하고 $o' \in \{\widehat{\mathrm{E}}\}$ 이다. 따라서 $\dfrac{\sigma^v \vdash e_1 \rightarrow \widehat{\mathrm{E}}}{\sigma^v \vdash \mathrm{case}\, e_1\, x\, e_2\, e_3 \rightarrow \widehat{\mathrm{E}}}$ 이 성립하고 $\widehat{\mathrm{E}} \in \{\widehat{\mathrm{E}}\}$ 이다.

[CASE-2a] $\dfrac{\overline{x}\,X \rhd e_1 : C_1 \quad se \rhd e_3 : C_2}{se \rhd \mathrm{case}\, e_1\, x\, e_2\, e_3 : C_1 \cup C_2}$ 새 변수 X

가정에 의해서 $\sigma^v \vdash e_1 \rightarrow v_1$, $\sigma^v \vdash e_3 \rightarrow o'$ 이 성립하고 $v_1 \in I(\overline{x}\,X)$ 이며 $o' \in I(se)$ 이다. $v_1 = x'\ v'$ 이고 $x' \neq x$ 라 하면, $\dfrac{\sigma^v \vdash e_1 \rightarrow x'\ v' \quad \sigma^v \vdash e_3 \rightarrow o'}{\sigma^v \vdash \mathrm{case}\ e_1\ x\ e_2\ e_3 \rightarrow o'}$ 이 성립하고 $o' \in I(se)$ 이다.

[HNDLN-a] $\dfrac{se \rhd e_1 : C}{se \rhd \mathrm{handle}\ e_1\ \mathrm{E}\ e_2 : C}$ $se \neq \{\widehat{\mathrm{E}}\}$

가정에 의해서, $\sigma^v \vdash e_1 \rightarrow v'$ 이 성립하고 $v' \in I(se)$ 이다. 따라서 $\dfrac{\sigma^v \vdash e_1 \rightarrow v'}{\sigma^v \vdash \mathrm{handle}\ e_1\ \mathrm{E}\ e_2 \rightarrow v'}$ 이 성립하고 $v' \in I(se)$ 이다.

[HNDX-a] $\dfrac{\widehat{\mathrm{E}'} \rhd e_1 : C}{\widehat{\mathrm{E}'} \rhd \mathrm{handle}\ e_1\ \mathrm{E}\ e_2 : C}$ $\mathrm{E}' \neq \mathrm{E}$

가정에 의해서, $\sigma^v \vdash e_1 \rightarrow o'$ 이 성립하고 $o' \in \{\widehat{\mathrm{E}'}\}$ 이다. 따라서 $\dfrac{\sigma^v \vdash e_1 \rightarrow \widehat{\mathrm{E}'}}{\sigma^v \vdash \mathrm{handle}\ e_1\ \mathrm{E}\ e_2 \rightarrow \widehat{\mathrm{E}'}}$ 이 성립하고 $\widehat{\mathrm{E}'} \in \{\widehat{\mathrm{E}'}\}$ 이다. □

류 석 영

1995년 한국과학기술원 전산학과 학사(B.S.). 1996년 한국과학기술원 전산학과 석사(M.S.). 1996년 ~ 현재 한국과학기술원 전산학과 박사과정.

이 광 근

정보과학회논문지 : 소프트웨어 및 응용 제 27권 제 3 호 참조