정적 분석의 실행시간 전문화에 관한 연구

어현준, 이광근*
한국과학기술원, 전산학과

요 약

프로그램의 실행 도중 사용자의 입력 패턴에 따라 프로그램의 분석을 전문화(specialize) 시킬 수 있다. 기존의 정적 분석 방법으로부터 실행시간 입력에 따라 분석을 전문화 시킬수 있는 방법을 고안하는 것이 이 연구의 목표이다.

정적 분석에서 우리는 실행시간 입력에 대해서는 모든 값을 가질 수 있다고 전제하고 분석을 시작한다. 따라서, 입력에 의존하는 모든 식들은 그 결과가 헐렁하게 나올 수 밖에 없다. 만약, 입력에 의존하는 식들에 대해서는 분석을 진행하지 않고, 입력과 상관없이 컴파일 시간에 가능한 분석들을 모두 진행한 후, 입력을 보고 나머지 분석을 진행한다면 정적 분석보다 더 정확한 결과를 얻을 수 있을 것이다. 그러나, 이 방법은 실행시간 부담이 큰 단점이 있다.

본 연구에서서는 값을 자르는 분석(Static Value Slicing)을 이용해 실행시간 부담을 줄이면서 입력에 따른 전문화가 가능하도록 하는 분석 방법을 제안한다. 값을 자르는 분석은 주어진 식 e가 최종적으로 집합 V에 속한 값을 내려면 그 내부식들의 값은 무엇이 되어야 하는 지를 찾아내는 방법이다. 값을 자르는 분석을 이용하여 주어진 식 e가 어떤 집합 V에 속한 값을 내려면 프로그램의 입력 ω가 어떤 값에 속해야 하는 지를 찾아낼 수 있고, 이를 이용하여 우리는 입력 ω가 주어지자마자 프로그램의 정확한 (정적 분석에 비해) 분석 결과를 얻을 수 있다.

1 언어

우리가 대상으로 하는 언어는 다음과 같은 구문구조를 가지며 값에 의한 호출(call-by-value)을 이용하는 고차(higher-order) 언어이다.

$$
\begin{array}{rcll}
e & ::= & 1 & \text{unit} \\
 & \mid & x & \text{variable} \\
 & \mid & \omega & \text{free variable} \\
 & \mid & \lambda x.e & \text{abstraction} \\
 & \mid & e_1\ e_2 & \text{application} \\
 & \mid & \texttt{fix}\ f = \lambda x.e_1\ \texttt{in}\ e_2 & \text{recursive function binding} \\
 & \mid & \kappa(e) & \text{data construction} \\
 & \mid & \kappa^{-1}(e) & \text{data deconstruction} \\
 & \mid & \texttt{case}\ e_1\ \kappa\ e_2\ e_3 & \text{switch}
\end{array}
$$

이 언어가 가질 수 있는 값은 데이타 또는 함수이다. 데이타 값 $\kappa\, v$는 e가 v라는 값으로 계산되어질 때 "$\kappa(e)$"라는 식으로부터 만들어지며, 함수 값 $\lambda x.e$는 식 "$\lambda x.e$"로 부터 만들어진다. 식 "$\texttt{fix}\ f = \lambda x.e_1\ \texttt{in}\ e_2$"의 계산은 f가 함수 값 $\lambda x.e$를 가지는 환경 하에서 e_2의 값을 계산한다. "$\texttt{case}\ e_1\ \kappa\ e_2\ e_3$"의 계산은 e_1의 값이 $\kappa\, v$이면 e_2를 계산한 값이 되고, e_1의 값이 $\kappa\, v$가 아니면 e_3를 계산한 값이 된다. 프로그램 $\wp$의 입력은 "ω"이다.

프로그램 내의 식 $\texttt{fix}\ f = \lambda x.e_1\ \texttt{in}\ e_2 \in \wp$에 있는 모든 f들과, 식 $\lambda x.e$ 내부에 있는 모든 x들의 이름은 모두 다르다고 가정한다.

---

*E-mail: {poisson; kwang}@pllab.kaist.ac.kr, web: http://pllab.kaist.ac.kr/~{poisson; kwang}

2 집합 기반 분석(Set-Based Analysis)

집합 기반 분석 방법은 프로그램의 수행 시 각 변수가 가질 수 있는 값을 집합으로 생각해 그 값 사이의 집합 관계식(set constraints)을 도출해 내고, 그 관계식을 풀어 변수가 실행중 가질 수 있는 값들을 정적으로 구해내는 방법이다[Hei93].

집합관계식을 이용한 분석에서는 프로그램 변수의 값을 여러 값들의 집합으로 취급함으로써 변수간의 의존성(dependency)을 무시하고 분석을 수행하게 된다. 동작 중심의 의미구조에서 환경을 각각의 변수를 값들의 집합으로 대응시키는 집합 환경(set environment)으로 바꾸어서 집합을 이용한 의미 구조(set-based semantics)로 바꿀 수 있다. 이 집합을 사용한 의미구조를 이용하면 프로그램 수행시 각 변수가 가지는 값을 집합을 이용해 근사(approximate)할 수 있다. 주어진 프로그램 e_0에 대해 $\vdash e_0 \rightarrow v$ 인 경우를 근사하는 집합 환경$\mathcal{E}$는 그 집합 환경에서 각각의 변수에 대응하는 집합이 실제로 e_0를 동작 중심의 의미구조 상에서 실행한 값을 모두 포함할 경우에 안전(safe)하다고 한다. 집합을 이용한 근사(set-based approximation)는 프로그램에 대해 안전한 집합 환경 중, 최소 집합 환경하에서 프로그램이 가질 수 있는 모든 값을 모은 것이다. 이런 근사값을 구하는 것이 바로 집합 기반 분석이다.

집합 기반 분석은 두가지 단계로 이루어진다. 첫번째 단계는 주어진 프로그램 에 대해 집합 관계식(set constraints)을 구하는 단계이고, 두번째는 구해진 집합관계식을 만족하는 최소의 해(least model)를 가지는 간단한 집합 관계식을 구하는 것이다.

2.1 집합 관계식의 도출

먼저, 집합 관계식을 정의해 보자. 프로그램의 집합 관계식 C는 $\{\mathcal{X}_e \supseteq se \mid e \in \wp\}$의 형태를 가진다. $\mathcal{X}_e \supseteq se$의 의미는 식 e의 값이 집합 식se가 나타내는 값의 집합을 포함할 수 있다는 것이다.

기본 집합식 ae는 다음과 같이 정의된다. ae는 식이 가질 수 있는 값을 나타내며 우리가 최종적으로 구할려는 해는 $\mathcal{X} \supseteq ae$꼴로 표현 되어지는 값의 정규트리문법(regular tree grammar)이다.

$$\begin{array}{rcl} ae & ::= & \mathtt{1} \\ & | & \lambda x.e \\ & | & \kappa\,\mathcal{X} \end{array}$$

집합 식 se는 기본 집합식 ae를 포함한다. 집합 변수 X는 프로그램 내부 식마다 하나씩 유니크하게 결정되며, 각 식이 가질 수 있는 값들을 대표한다. $\kappa^{-1}\,\mathcal{X}$는 $\mathcal{X}$가 가지는 값들에 κ를 떼어낸 값들 이다. $app(\mathcal{X}_1, \mathcal{X}_2)$는 $\mathcal{X}_2$의 값들을 인수로 $\mathcal{X}_1$이 가지는 함수들을 호출한 결과 값들을 의미한다. $case(\mathcal{X}_1, \kappa, \mathcal{X}_2, \mathcal{X}_3)$는 case문이 가질 수 있는 값들을 의미한다.

$$\begin{array}{rcl} se & ::= & ae \\ & | & \mathcal{X} \\ & | & \kappa^{-1}\,\mathcal{X} \\ & | & app(\mathcal{X}_1, \mathcal{X}_2) \\ & | & case(\mathcal{X}_1, \kappa, \mathcal{X}_2, \mathcal{X}_3) \end{array}$$

그림 1은 각 집합식들의 정확한 의미와, 프로그램으로부터 집합 관계식을 도출해 내는 규칙을 나타낸다.

2.2 집합 관계식의 해

우리가 최종적으로 원하는 결과는 ae의 정규 트리 문법으로 표현된 내부식들의 값이다. 따라서 $\mathcal{X} \supseteq se$의 형태로 도출된 집합 관계식들은 그림 2의 규칙에 의해 $\mathcal{X} \supseteq ae$의 형태로 단순화(simplify)되어져야 한다. 그림 2의 규칙들은 그림 1의 의미를 제대로 반영하도록 정의된다. 그림 3은 프로그램으로부터 도출된 집합 관계식 $\mathcal{C}$를 입력으로 받아 더 이상 단순화될 관계식이 없을 때까지 그림 2의 규칙을 적용해 그 중 $\mathcal{X} \supseteq ae$ 형태의 단순화된 관계식만을 결과로 낸다.

3 목표

N. Heintz의 집합 기반 분석은 프로그램에 입력, 즉 자유 변수(free variable)이 없을 경우에 대해서만 정의되어진다. 만약, 프로그램에 입력 ω가 있다면, 그 입력을 어떻게 다루어야 될까?

$$
\begin{array}{rclll}
v & \in & Val & = & Closure + Constant + \mathtt{1} \quad \text{values} \\
\lambda x.e & \in & Closure & = & Expr \quad \text{lambda exprs in program } \wp \\
\kappa\, v & \in & Constant & = & Con \times Val \quad \text{constants} \\
\kappa & \in & Con & = & \{\kappa_1, \cdots, \kappa_N\} \quad \text{constructors in program } \wp
\end{array}
$$

$$
\begin{array}{lcl}
\mathcal{I}(\mathcal{X}_e) & \subseteq & Val \\
\mathcal{I}(\lambda x.e) & = & \{\lambda x.e\} \\
\mathcal{I}(\kappa^{-1}\, \mathcal{X}_1) & = & \{v \mid \kappa\, v \in \mathcal{I}(\mathcal{X}_1)\}
\end{array}
\qquad
\begin{array}{lcl}
\mathcal{I}(\mathtt{1}) & = & \{\mathtt{1}\} \\
\mathcal{I}(\kappa\, \mathcal{X}_1) & = & \{\kappa\, v \mid v \in \mathcal{I}(\mathcal{X}_1)\}
\end{array}
$$

$$
\begin{array}{lcl}
\mathcal{I}(app(\mathcal{X}_1, \mathcal{X}_2)) & = & \{v \mid \lambda x.e \in \mathcal{I}(\mathcal{X}_1), v \in \mathcal{I}(\mathcal{X}_e)\} \text{ provided } \lambda x.e \in \mathcal{I}(\mathcal{X}_1) \text{ implies } \mathcal{X}_x \supseteq \mathcal{X}_2 \\
\mathcal{I}(case(\mathcal{X}_1, \kappa, \mathcal{X}_2, \mathcal{X}_3)) & = & \{v \mid v \in \mathcal{I}(\mathcal{X}_2), \kappa\, v' \in \mathcal{I}(\mathcal{X}_1)\} \cup \{v \mid v \in \mathcal{I}(\mathcal{X}_3), \kappa' v' \in \mathcal{I}(\mathcal{X}_1), \kappa' \neq \kappa\}
\end{array}
$$

$$
[\text{VAR}] \quad \rhd_1 x : \{\ \} \qquad [\text{C}] \quad \rhd_1 \mathtt{1} : \{\mathcal{X}_e \supseteq \mathtt{1}\} \qquad [\text{ABS}] \quad \frac{\rhd_1 e_1 : \mathcal{C}_1}{\rhd_1 \lambda x.e_1 : \{\mathcal{X}_e \supseteq \lambda x.e_1\} \cup \mathcal{C}_1}
$$

$$
[\text{FIX}] \quad \frac{\rhd_1 e_1 : \mathcal{C}_1 \quad \rhd_1 e_2 : \mathcal{C}_2}{\rhd_1 \mathtt{fix}\ f = \lambda x.e_1\ \mathtt{in}\ e_2 : \{\mathcal{X}_e \supseteq \mathcal{X}_2, \mathcal{X}_f \supseteq \lambda x.e_1\} \cup \mathcal{C}_1 \cup \mathcal{C}_2}
$$

$$
[\text{APP}] \quad \frac{\rhd_1 e_1 : \mathcal{C}_1 \quad \rhd_1 e_2 : \mathcal{C}_2}{\rhd_1 e_1\ e_2 : \{\mathcal{X}_e \supseteq app(\mathcal{X}_1, \mathcal{X}_2)\} \cup \mathcal{C}_1 \cup \mathcal{C}_2}
$$

$$
[\text{CON}] \quad \frac{\rhd_1 e_1 : \mathcal{C}_1}{\rhd_1 \kappa(e_1) : \{\mathcal{X}_e \supseteq \kappa\, \mathcal{X}_1\} \cup \mathcal{C}_1}
$$

$$
[\text{DECON}] \quad \frac{\rhd_1 e_1 : \mathcal{C}_1}{\rhd_1 \kappa^{-1}(e_1) : \{\mathcal{X}_e \supseteq \kappa^{-1}\, \mathcal{X}_1\} \cup \mathcal{C}_1}
$$

$$
[\text{CASE}] \quad \frac{\rhd_1 e_1 : \mathcal{C}_1 \quad \rhd_1 e_2 : \mathcal{C}_2 \quad \rhd_1 e_3 : \mathcal{C}_3}{\rhd_1 \mathtt{case}\ e_1\ \kappa\ e_2\ e_3 : \{\mathcal{X}_e \supseteq case(\mathcal{X}_1, \kappa, \mathcal{X}_2, \mathcal{X}_3)\} \cup \mathcal{C}_1 \cup \mathcal{C}_2 \cup \mathcal{C}_3}
$$

그림 1: constructing set constraints $\rhd_1$

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

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

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

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

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

그림 2: set constraints simplification rule

정적 분석의 틀에서 프로그램의 입력은 모든 값을 가질 수 있다고 전제하고 분석을 시작한다. ω의 값을 컴파일 시간에는 알 방법이 없기때문에 안전한 분석을 위해서는 프로그램이 실행중 발생할 수 있는 모든 값을 가지게 해야 하는 것이다. 프로그램이 실행 중 발생할 수 있는 모든 값을 가지는 집합을 $\mathcal{U}$라하고, $\rhd_1 \wp : \mathcal{C}$일 때, 입력 ω에 대한 집합 관계식은 $\{\mathcal{X}_\omega \supseteq \mathcal{U}\} \cup \mathcal{C}$가 된다. 이렇게 만들어진

input a collection $\mathcal{C}$ of set constraints
repeat
  add set constraints according to rules in figure 2
until no changes in $\mathcal{C}$
output $\{\mathcal{X} \supseteq ae \mid \mathcal{X} \supseteq ae \in \mathcal{C}\}$

그림 3: set constraints simplification algorithm, $\mathcal{S}$

집합 관계식들을 그림 2의 알고리즘에 의해 풀면, 식 e의 값이 입력 ω에 의해 결정되어지는 경우 실제로 수행 중에는 제한된 값을 가짐에도 불구하고, 안전한 분석을 위해 더 헐렁한 결과를 만들어내게 된다. 예를 들어, e가 case $\omega\ \kappa\ e_2\ e_3$일 경우, 실제 실행 시간에 ω가 $\kappa(...)$ 의 값만을 가지더라도 -컴파일 시간에는 ω가 실행시간에 어떤 값들을 가지는 지 모르므로- $\mathcal{X}_\omega \supseteq \mathcal{U}$가 되고, 따라서 $\mathcal{X}_e$는 $\mathcal{X}_2$와 $\mathcal{X}_3$를 모두 가지게 된다.

다른 방법으로 프로그램이 수행 중 입력 ω의 값을 알고 나서 분석을 시작할 수도 있다. 이 경우 집합 관계식 $\mathcal{C}$는 그림 1의 방법에 따라 컴파일시간에 만들 수 있다. 그러나, $\mathcal{X}_\omega$에 대한 관계식이 만들어져 있지 않기 때문에 실행시간 까지 기다렸다가, v라는 값이 결정 되면, 컴파일 시간에 만들어 두었던 관계식들과 ω에 대한 관계식을 합친 $S(\{\mathcal{X}_\omega \supseteq v\} \cup \mathcal{C})$를 이용하여 해를 구하게 된다. 이 방법의 장점은 정확한 v값을 이용하여 분석을 하기 때문에 매우 엄밀한 분석을 할 수 있다는 것이다. 반면에 프로그램의 수행 중에 모든 분석을 해야 하기 때문에 실행시간 부담이 매우 큰 단점이 있다.

이 연구의 목표는 프로그램의 수행 중에 입력이 일정한 패턴을 가질 때, 정적 분석을 통해 얻어진 결과보다 더 정확한 결과를 내면서, 실행시간 부담이 적은 분석 방법을 고안하는 것이다.

4 컴파일시간 준비/ 실행시간 분석

이 절에서는 컴파일 시간에 분석할 수 있는 곳까지 분석을 하고 나머지는 실행시간으로 미루는 방법에 대해서 설명하려고 한다.

이 방법이 $\mathcal{U}$를 이용한 정적 분석과 다른 부분은 집합 관계식을 풀 때, $\mathcal{U}$가 들어감으로써 헐렁하게 할 수 밖에 없었던 부분을 실행시간으로 미루는 것이다. 이런 부분을 모달 연산자 $\Box$라 하고 다음과 같이 정의된다. 모달 논리(Modal Logic)에서 연산자 $\Box$는 '필연적으로(necessarily)" 의 의미를 가지며, 지금도 참이지만 앞으로도 계속 참이 되어야 함을 의미한다[AGM92]. 즉, $\mathcal{X} \supseteq \Box$이면, $\mathcal{X}$는 컴파일시간에도 $\mathcal{X} \supseteq \Box$이면서 실행시간에도 $\mathcal{X} \supseteq \Box$의 관계가 성립해야 함을 의미한다.

$$\begin{array}{lll} \Box & ::= & \mathcal{X}_\omega \\ & | & \kappa_\Box^{-1}\ \mathcal{X} \\ & | & app_\Box(\mathcal{X}_1, \mathcal{X}_2) \\ & | & case_\Box(\mathcal{X}_1, \kappa, \mathcal{X}_2, \mathcal{X}_3) \end{array}$$

$\mathcal{X}_\omega$는 입력 ω에 대한 집합 변수이고, $\kappa_\Box^{-1}\ \mathcal{X}$의 의미는 $\mathcal{X} \supseteq \Box$이 되어 더 이상 분석을 진행할 수 없을 때 전체 식의 값은 $\mathcal{X}$의 값이 실행시간에 $\kappa\, v$라는 값으로 결정되면 그 값이 인자인 v를 값으로 가진다는 의미이다. $app_\Box(\mathcal{X}_1, \mathcal{X}_2)$는 실행시간에 $\mathcal{X}_1$이 어떤 함수인지 결정되면 $\mathcal{X}_2$를 인자로 그 함수를 호출한 결과 값을 의미한다. $case_\Box(\mathcal{X}_1, \kappa, \mathcal{X}_2, \mathcal{X}_3)$는 $\mathcal{X}_1$이 실행시간에 $\kappa(...)$의 값을 가지면 $\mathcal{X}_2$이고, 아니면 $\mathcal{X}_3$라는 의미이다.

컴파일시간 분석 규칙은 그림 5와 같으며, 알고리즘 $\mathcal{S}_c$은 이 규칙을 더 이상 적용할 수 없을 때까지 계속 적용해서 나온 집합 관계식들 중, $\mathcal{X} \supseteq ae$또는 $\mathcal{X} \supseteq \Box$인 것들을 모은 것이다.

프로그램의 수행 중에 ω의 값이 v로 결정되면, 컴파일시간에 분석한 결과와 관계식 $\mathcal{X}_\omega \supseteq v$를 이용하여 $\mathcal{U}$를 이용한 분석보다 더 정확한 분석을 얻어낼 수 있다. 실행시간 알고리즘 $\mathcal{S}_r$은 $\mathcal{S}_c$의 결과와 $\mathcal{X}_\omega \supseteq v$를 입력으로 받아서, 그림 7의 규칙에 따라 관계식들을 풀고, $\mathcal{X} \supseteq ae$ 꼴의 해를 구해 내게 된다.

$$
\begin{array}{lcl@{\qquad}lcl}
\mathcal{I}(\mathcal{X}_e) & \subseteq & Val & \mathcal{I}(1) & = & \{1\} \\
\mathcal{I}(\lambda x.e) & = & \{\lambda x.e\} & \mathcal{I}(\kappa\,\mathcal{X}_1) & = & \{\kappa\, v \mid v \in \mathcal{I}(\mathcal{X}_1)\}
\end{array}
$$

$$
\begin{array}{lcl}
\mathcal{I}(\kappa^{-1}\,\mathcal{X}_1) & = & \{v \mid \kappa\, v \in \mathcal{I}(\mathcal{X}_1)\} \cup \{v \mid \mathcal{I}(\mathcal{X}_1) \supseteq \mathcal{I}(\square), v \in \mathcal{I}(\kappa_\square^{-1}\,\mathcal{X}_1)\} \\
\mathcal{I}(app(\mathcal{X}_1, \mathcal{X}_2)) & = & \{v \mid \lambda x.e \in \mathcal{I}(\mathcal{X}_1), v \in \mathcal{I}(\mathcal{X}_e)\} \cup \{v \mid \mathcal{I}(\mathcal{X}_1) \supseteq \mathcal{I}(\square), v \in \mathcal{I}(app_\square(\mathcal{X}_1, \mathcal{X}_2))\} \\
 & & \text{provided } \lambda x.e \in \mathcal{I}(\mathcal{X}_1) \text{ implies } \mathcal{X}_x \supseteq \mathcal{X}_2 \\
\mathcal{I}(case(\mathcal{X}_1, \kappa, \mathcal{X}_2, \mathcal{X}_3)) & = & \{v \mid v \in \mathcal{I}(\mathcal{X}_2), \kappa\, v' \in \mathcal{I}(\mathcal{X}_1)\} \cup \{v \mid v \in \mathcal{I}(\mathcal{X}_3), \kappa' v' \in \mathcal{I}(\mathcal{X}_1), \kappa' \neq \kappa\} \\
 & & \cup\{v \mid \mathcal{I}(\mathcal{X}_1) \supseteq \mathcal{I}(\square), v \in \mathcal{I}(case_\square(\mathcal{X}_1, \kappa, \mathcal{X}_2, \mathcal{X}_3))\} \\
\mathcal{I}(\kappa_\square^{-1}\,\mathcal{X}_1) & = & \{v \mid \kappa\, v \in \mathcal{I}(\mathcal{X}_1) \text{ at runtime}\} \\
\mathcal{I}(app_\square(\mathcal{X}_1, \mathcal{X}_2)) & = & \{v \mid \lambda x.e \in \mathcal{I}(\mathcal{X}_1), v \in \mathcal{I}(\mathcal{X}_e) \text{ at runtime}\} \\
 & & \text{provided } \lambda x.e \in \mathcal{I}(\mathcal{X}_1) \text{ implies } \mathcal{X}_x \supseteq \mathcal{X}_2 \\
\mathcal{I}(case_\square(\mathcal{X}_1, \kappa, \mathcal{X}_2, \mathcal{X}_3)) & = & \{v \mid \kappa\, v \in \mathcal{I}(\mathcal{X}_2), v' \in \mathcal{I}(\mathcal{X}_1) \text{ at runtime}\} \\
 & & \cup\{v \mid \kappa'\, v \in \mathcal{I}(\mathcal{X}_3), v' \in \mathcal{I}(\mathcal{X}_1) \text{ at runtime}\}
\end{array}
$$

그림 4: Interpretation of set constraints in Semi-Static Analysis

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

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

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

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

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

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

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

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

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

그림 5: set constraints simplification rule (compile-time)

5 가능성(possiblity)을 이용한 분석

앞 절에서는 컴파일시간에 해를 구할 수 있는 데까지만 구하고, 분석할 수 없는 부분 - 입력에 의해 결정되어지는 부분 - 은 실행시간으로 미루는 방법에 대해 설명한 반면, 이 절에서는 값을 자르는 분석(Static Value Slicing)을 이용해 컴파일 시간에 분석할 수 없는 부분은 그 분석 가능성을 입력에

input a collection $\mathcal{C}$ of set constraints
repeat
  add set constraints according to rules in figure 5
until no changes in $\mathcal{C}$
output $\{\mathcal{X} \supseteq ae \mid \mathcal{X} \supseteq ae \in \mathcal{C}\} \cup \{\mathcal{X} \supseteq \Box \mid \mathcal{X} \supseteq \Box \in \mathcal{C}\}$

그림 6: set constraints simplification algorithm, $\mathcal{S}_c$

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

$$\frac{\mathcal{X} \supseteq case_{\Box}(\mathcal{X}_1, \kappa, \mathcal{X}_2, \mathcal{X}_3) \quad \mathcal{X}_1 \supseteq \kappa\, \mathcal{Y}}{\mathcal{X} \supseteq \mathcal{X}_2}$$

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

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

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

그림 7: set constraints simplification rule (runtime)

input $\mathcal{C}' = \mathcal{S}_c(\mathcal{C}) \cup \{\mathcal{X}_\omega \supseteq ae\}$
repeat
  add set constraints according to rules in figure 7
until no changes in $\mathcal{C}'$
output $\{\mathcal{X} \supseteq ae \mid \mathcal{X} \supseteq ae \in \mathcal{C}'\}$

그림 8: set constraints simplification algorithm, $\mathcal{S}_r$

관한 식으로 표현한 다음 프로그램의 수행 중 실제 입력이 결정되면 컴파일 시간에 준비된 가능성을 이용하여 빠르게 구해 내는 방법을 제안한다.

예를 들어 $e_1\ e_2$의 값을 분석한다고 하자. 그러면, 이 식이 가질 수 있는 모든 값들에 대해서 그 값이 결정되어 질 때의 입력의 조건을 구해 실제 입력과 그 조건들을 비교해 봄으로써 e_1의 값을 계산하지 않고도 $e_1\ e_2$의 값을 분석할 수 있게 된다.

5.1 값을 자르는 분석(Static Value Slicing)

값을 자르는 분석이란, 주어진 프로그램 $\wp$가 최종적으로 집합 V에 속한 값을 계산해 내려면 그 내부식들의 값은 무엇이되어야 할까를 분석하는 것이다[Yi98]. 여기서는 Yi의 SVS를 다음과 같이 확장하였다.

- 모든 내부 식 $e \in \wp$에 대하여 분석이 가능하도록 하였다.

$$\frac{\mathcal{X} \supseteq app(\mathcal{X}_1, \mathcal{X}_2) \quad \mathcal{X}_1 \supseteq \Diamond \quad \lambda x.e \in \wp}{\mathcal{X} \supseteq app_\Diamond(\mathcal{X}_\omega, SVS(\mathcal{X}_1 \subseteq \lambda x.e)(\omega), \mathcal{X}_e, \mathcal{X}_x, \mathcal{X}_2)}$$

$$\frac{\mathcal{X} \supseteq case(\mathcal{X}_1, \kappa, \mathcal{X}_2, \mathcal{X}_3) \quad \mathcal{X}_1 \supseteq \Diamond}{\mathcal{X} \supseteq case_\Diamond(\mathcal{X}_\omega, SVS(\mathcal{X}_1 \subseteq \kappa\,\mathcal{U})(\omega), \mathcal{X}_2, SVS(\mathcal{X}_1 \subseteq \overline{\kappa}\,\mathcal{U})(\omega), \mathcal{X}_3)}$$

$$\frac{\mathcal{X} \supseteq \kappa^{-1}\,\mathcal{Y} \quad \mathcal{Y} \supseteq \Diamond \quad S_1 = SVS(\mathcal{Y} \subseteq \kappa\,\mathcal{U})(\omega) \quad S_2 = \{v \mid \kappa\, v \in \mathcal{S}(\{\mathcal{X}_\omega \supseteq S_1\} \cup \mathcal{C}_1)(\mathcal{Y})\}}{\mathcal{X} \supseteq \kappa_\Diamond^{-1}(\mathcal{X}_\omega, S_1, S_2)}$$

그림 9: simplification rule with SVS (compile-time)

- Yi의 SVS는 표준 동적 의미구조(standard operational semantics)위에서 정의된 반면, 여기서는 집합기반 동적 의미구조(Set-Based operational semantics)위에서 정의되었다.

주어진 프로그램 $\wp$, 식 $e \in \wp$, 값을 자르는 기준 V에 대한 $SVS(e, V)(\omega)$는 입력 ω가 v'이라는 값을 가진다는 가정하에서 집합기반 동적 의미구조에 의해 e가 $v \in V$의 값을 가진다면 그러한 v'들을 모두 모은 것이다. 즉 $\mathcal{E}_{min} \vdash [v'/\omega]e \rightsquigarrow v \in V$ 인 모든 v'은 $SVS(e, V)(\omega)$에 포함되어야 한다.

5.2 가능성을 이용한 분석

$\Box$에서 $app_\Box(\mathcal{X}_1, \mathcal{X}_2)$, $case_\Box(\mathcal{X}_1, \kappa, \mathcal{X}_2, \mathcal{X}_3)$, $\kappa_\Box^{-1}(\mathcal{X}_1, \kappa, \mathcal{X}_2, \mathcal{X}_3)$은 모두 $\mathcal{X}_1 \supseteq ae$꼴로 $\mathcal{X}_1$의 값이 결정되어야만 simplify되어진다. 따라서, $\mathcal{X}_1 \supseteq ae$의 constraint가 만들어질 때까지 기다려야만 한다. $\Box$를 subexpression에 대한 조건이 아니라, $\mathcal{X}_\omega$에 대한 조건으로 바꾼 것이 $\Diamond$이다. $\Diamond$는 모달 논리에서 '아마도(possibliy)'의 의미를 가지며[AGM92] 여기서는 컴파일시간에 준비된 $\mathcal{X} \supseteq \Diamond$의 관계식들 중 실행시간에 $\mathcal{X}$가 아마도 $\Diamond$를 가지게 될 가능성이 있음을 의미한다. $\Diamond$가 앞에서 설명한 $\Box$와 다른 점은 컴파일 시간에 $\mathcal{X} \supseteq \Box$이면 실행시간에 $\mathcal{X}$는 반드시 $\Box$를 가지게 되고, 컴파일 시간에 $\mathcal{X} \supseteq \Diamond$이라고 해서 실행시간에 $\mathcal{X}$가 반드시 $\Diamond$를 가지지는 않는다는 것이다.

$$\begin{array}{lcl} \Diamond & ::= & \mathcal{X}_\omega \\ & | & \kappa_\Diamond^{-1}(\mathcal{X}_\omega, S_1, S_2) \\ & | & app_\Diamond(\mathcal{X}_\omega, S, \mathcal{X}_1, \mathcal{X}_2, \mathcal{X}_3) \\ & | & case_\Diamond(\mathcal{X}_\omega, S_1, \mathcal{X}_1, S_2, \mathcal{X}_2) \end{array}$$

$\Diamond$의 의미는 다음과 같다. $\kappa_\Diamond^{-1}(\mathcal{X}_\omega, S_1, S_2)$의 의미는 입력이 S_1과 관계가 있으면 S_2의 값을 가질 수 있다는 것이고, $app_\Diamond(\mathcal{X}_\omega, S, \mathcal{X}_1, \mathcal{X}_2, \mathcal{X}_3)$의 의미는 입력이 집합 S와 관계가 있으면 전체 식은 $\mathcal{X}_1$의 값을, $\mathcal{X}_2$는 $\mathcal{X}_3$의 값을 가질 수 있다는 의미이다. $case_\Diamond(\mathcal{X}_\omega, S_1, \mathcal{X}_1, S_2, \mathcal{X}_2)$의 의미는 입력이 S_1과 관계가 있으면 $\mathcal{X}_1$의 값을, S_2와 관계가 있으면 $\mathcal{X}_2$의 값을 가질 수 있다는 의미이다.

$$\begin{array}{lcl} \mathcal{I}(\kappa_\Diamond^{-1}(\mathcal{X}_\omega, S_1, S_2)) & = & \{v \mid v \in S_2, \mathcal{X}_\omega \cap S_1 \neq \emptyset\} \\ \mathcal{I}(app_\Diamond(\mathcal{X}_\omega, S, \mathcal{X}_1, \mathcal{X}_2, \mathcal{X}_3)) & = & \{v \mid v \in \mathcal{X}_1, \mathcal{X}_\omega \cap S_1 \neq \emptyset\} \text{ provided } \mathcal{X}_\omega \cap S_1 \neq \emptyset \text{ implies } \mathcal{X}_2 \supseteq \mathcal{X}_3 \\ \mathcal{I}(case_\Diamond(\mathcal{X}_\omega, S_1, \mathcal{X}_1, S_2, \mathcal{X}_2)) & = & \{v \mid v \in \mathcal{X}_1, \mathcal{X}_\omega \cap S_1 \neq \emptyset\} \cup \{v \mid v \in \mathcal{X}_2, \mathcal{X}_\omega \cap S_2 \neq \emptyset\} \end{array}$$

컴파일 시간에 가능성 $\Diamond$을 준비하는 규칙은 그림 9와 같다.

만약, $\mathcal{X} \supseteq app(\mathcal{X}_1, \mathcal{X}_2)$인데, $\mathcal{X}_1$이 컴파일시간에 결정될 수 없으면 규칙 9는 그 위치에서 호출될 수 있는 모든 함수 $\lambda x.e$들에 대해서 $\mathcal{X}_1 \supseteq \lambda x.e$가 될 입력의 조건 $SVS(\mathcal{X}_1 \subseteq \lambda x.e)(\omega)$을 구하고, 그 조건을 이용하여 관계식들을 만들어 낸다. 따라서, 규칙 9는 하나의 식 $e_1\ e_2$에 대해서 여러개의 가능성($\Diamond$을 포함한 관계식)들을 만들어 내고, 프로그램의 입력에 따라 그 중 몇개는 해를 구할 수 있고, 몇개는 해를 구할 수 없게 된다.

마찬가지 방법으로 $\mathcal{X} \supseteq case(\mathcal{X}_1, \kappa, \mathcal{X}_2, \mathcal{X}_3)$인데 $\mathcal{X}_1$을 컴파일시간에 결정할 수 없다면 규칙 9은 $\mathcal{X} \supseteq case_\Diamond(\mathcal{X}_\omega, SVS(\mathcal{X}_1 \subseteq \kappa\,\mathcal{U})(\omega), \mathcal{X}_2, SVS(\mathcal{X}_1 \subseteq \overline{\kappa}\,\mathcal{U})(\omega), \mathcal{X}_3)$ 의 관계식을 만들어 낸다.

자료(data)로 부터 인자를 분해(deconstruction)하는 경우는 위의 두 경우보다 더 복잡하다. $\mathcal{X} \supseteq \kappa^{-1}\,\mathcal{Y}$이 있을 때 $\mathcal{Y}$를 컴파일 시간에 계산할 수 없다고 하자. 어떤 경우에 관계식을 풀 것인가에 대

$$\frac{\mathcal{X} \supseteq app_\diamond(\mathcal{X}_\omega, S, \mathcal{X}_1, \mathcal{X}_2, \mathcal{X}_3) \quad \mathcal{X}_\omega \cap S \neq \emptyset}{\mathcal{X} \supseteq \mathcal{X}_1 \quad \mathcal{X}_2 \supseteq \mathcal{X}_3}$$

$$\frac{\mathcal{X} \supseteq case_\diamond(\mathcal{X}_\omega, S_1, \mathcal{X}_1, S_2, \mathcal{X}_2) \quad \mathcal{X}_\omega \cap S_1 \neq \emptyset}{\mathcal{X} \supseteq \mathcal{X}_1}$$

$$\frac{\mathcal{X} \supseteq case_\diamond(\mathcal{X}_\omega, S_1, \mathcal{X}_1, S_2, \mathcal{X}_2) \quad \mathcal{X}_\omega \cap S_2 \neq \emptyset}{\mathcal{X} \supseteq \mathcal{X}_2}$$

$$\frac{\mathcal{X} \supseteq \kappa_\diamond^{-1}(\mathcal{X}_\omega, S_1, S_2) \quad \mathcal{X}_\omega \cap S_1 \neq \emptyset}{\mathcal{X} \supseteq S_2}$$

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

그림 10: simplification rule with SVS (runtime)

한 조건 S_1은 $SVS(\mathcal{Y} \subseteq \kappa\,\mathcal{U})(\omega)$는 쉽게 생각할 수 있다. 단순히 $\mathcal{Y}$가 $\kappa\ldots$의 값을 가질때의 ω의 조건을 구하기만 하면 된다. 그러나, 그 경우 $\mathcal{X}$가 어떤 값을 가질 것인가에 대해서는 $\mathcal{Y}$가 실제로 어떤 값을 가지는지 알기 전에는 아무런 말을 해 줄 수 없다. 이를 해결하기 위해, 여기서는 $\mathcal{X}$의 값을 $\mathcal{Y}$가 실제로 위의 조건 S_1에 해당하는 값을 가진다는 가정하에 프로그램을 분석하고 그 분석이 말해주는 $\mathcal{X}$의 모든 값을 가질 수 있다고 근사(approximate)하였다.

이 방법을 사용하면 실제로 $\mathcal{Y}$의 값을 알고 분석을 진행한 것 보다 정확도는 떨어지지만 안전하면서 $\mathcal{Y}$의 값을 전체집합 $\mathcal{U}$라고 한 것보다는 더 정확한 결과를 프로그램의 입력 ω만 보고서도 얻을 수 있다.

5.3 정확성(correctness)

프로그램 $\wp$으로부터 도출된 집합 관계식이 $\mathcal{C}$라고 할때, 다음과 같이 정적 분석을 $\mathcal{F}_\mathcal{U}$, 실행시간에 입력ω의 값이 v임을 알고 시작한 분석을 $\mathcal{F}_v$, 가능성을 이용한 분석을 $\mathcal{F}_\diamond$으로 정의할 수 있다.

- $\mathcal{F}_v \stackrel{\text{def}}{=}$ let
  $\mathcal{C}' = \mathcal{C} \cup \{\mathcal{X}_\omega \supseteq \underline{v}\}$
  in
  $\lambda\Phi.\mathcal{C}' \cup \{\mathcal{X} \supseteq se \mid \frac{\psi}{\mathcal{X} \supseteq se} \in Rule_v, \psi \in \Phi\}$

- $\mathcal{F}_\diamond \stackrel{\text{def}}{=}$ let
  $\mathcal{F}_c = \lambda\Phi.\mathcal{C} \cup \{\mathcal{X} \supseteq se \mid \frac{\psi}{\mathcal{X} \supseteq se} \in Rule_{\diamond_c}, \psi \in \Phi\}$
  $\mathcal{C}_\diamond = \text{lfp } \mathcal{F}_c \cup \{\mathcal{X}_\omega \supseteq \underline{v}\}$
  in
  $\lambda\Phi.\mathcal{C}_\diamond \cup \{\mathcal{X} \supseteq se \mid \frac{\psi}{\mathcal{X} \supseteq se} \in Rule_{\diamond_r}, \psi \in \Phi\}$

- $\mathcal{F}_\mathcal{U} \stackrel{\text{def}}{=}$ let
  $\mathcal{C}_\mathcal{U} = \mathcal{C} \cup \{\mathcal{X}_\omega \supseteq \mathcal{U}\}$
  in
  $\lambda\Phi.\mathcal{C}_\mathcal{U} \cup \{\mathcal{X} \supseteq se \mid \frac{\psi}{\mathcal{X} \supseteq se} \in Rule_\mathcal{U}, \psi \in \Phi\}$

명제 1과 명제 2는 우리의 분석이 안전하면서, 즉 실행시간 분석의 결과를 모두 포함하면서, 정적 분석보다 더 정확한 결과를 이끌어 냄을 보여준다. 여기서 lfp는 함수의 최소 고정점(least fixpoint)을 의미하며, $|\cdot|$는 집합 관계식들의 정규트리문법으로 부터 그 값의 집합을 이끌어내는 연산자이다.

Proposition 1

ω를 입력으로 갖는 프로그램 $\wp$에 대해, $\triangleright_1 \wp : \mathcal{C}$ 의 실행시간 분석, 가능성을 이용한 분석의 해가 각각 $|\text{lfp } \mathcal{F}_v|$ 와 $|\text{lfp } \mathcal{F}_\diamond|$ 라면 프로그램의 모든 내부식 $e \in \wp$에 대해 다음의 관계가 성립한다. $|\text{lfp } \mathcal{F}_\diamond|(\mathcal{X}_e) \supseteq |\text{lfp } \mathcal{F}_v|(\mathcal{X}_e)$.

Proof. $\mathcal{F}_v$와 $\mathcal{F}_\diamond$을 [YRer]식으로 다시 정의한 다음 고정점 귀납법(fixpoint induction)을 사용해 증명 □

Proposition 2

ω를 입력으로 갖는 프로그램 $\wp$에 대해, $\triangleright_1 \wp : \mathcal{C}$ 의 가능성을 이용한 분석, 정적 분석의 해가 각각 $|\text{lfp } \mathcal{F}_\diamond|$ 와 $|\text{lfp } \mathcal{F}_\mathcal{U}|$ 라면 프로그램의 모든 내부식 $e \in \wp$에 대해 다음의 관계가 성립한다. $|\text{lfp } \mathcal{F}_\mathcal{U}|(\mathcal{X}_e) \supseteq |\text{lfp } \mathcal{F}_\diamond|(\mathcal{X}_e)$.

Proof. $\mathcal{F}_\mathcal{U}$를 [YRer]식으로 다시 정의한 다음 고정점 귀납법(fixpoint induction)을 사용해 증명 □

6 결론 및 고찰

본 연구에서는 값을 자르는 분석(Static Value Slicing)을 이용해 실행시간 부담을 줄이면서 입력에 따른 전문화가 가능하도록 하는 분석 방법을 제안했다. 값을 자르는 분석을 이용하여 주어진 식 e가 어떤 집합 V에 속한 값을 내려면 프로그램의 입력 ω가 어떤 값에 속해야 하는 지를 찾아낼 수 있고, 이를 이용하여 우리는 입력 ω가 주어지자마자 정적분석에 비해 더 정확한(accurate) 분석 결과를 얻을 수 있다.

지금 현재 분석의 정확성(correctness)을 증명했으며, 이 분석 방법에 적합한 실례를 찾고 있는 중이다. 앞으로 우리 분석이 정말로 잘 적용될 수 있는 실례를 찾아 분석을 실제로 구현하고 그 결과를 관찰하여 분석의 유용성을 보이는 일이 남아 있다.

참고 자료

[AGM92] S. Abramsky, Dov M. Gabby, and T.S.E Maibaum. *Handbook of Logic in Computer Science*, volume 1. Clarendon Press, Oxford, 1992.

[Hei93] Nevin Heintze. Set based analysis of ML programs. Technical Report CMU-CS-93-193, Carnegie Mellon University, July 1993.

[Yi98] Kwangkuen Yi. 값을 자르는 분석 (static value slicing). In *정보과학회 프로그래밍 분과 학술 발표회*, September 1998.

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