20-2-R03

다단계 프로그램에 대한 흐름을 고려한 분석[1]

Flow-sensitive Analysis of Multi-Staged Programs

김덕환 · 이광근
서울대학교 전기·컴퓨터공학부 프로그래밍 연구실
{dk; kwang}@ropas.snu.ac.kr

요 약

고차 함수를 지원하는 다단계 프로그래밍 언어에 대한 흐름을 고려한 집합 제약식 분석을 제안한다. 다단계 프로그램이란 프로그램 텍스트를 값처럼 다루는 프로그램이다. 최근에 다양한 언어 특징을 포함한 다단계 프로그램을 위한 let-다형 타입 체계와 그 추론 시스템이 제시되었다. 그러나, 타입 체계는 일반적으로 프로그램에서 실제로 실행되지 않는 부분에 대해서도 보수적으로 타입 검사를 수행하는 단점을 지니고 있다. 이 논문에서는 프로그램의 실행의 흐름을 고려하는 단순 타입 체계보다 정교한 프로그램 분석을 고안한다.

1 서론

다단계 프로그램(multi-staged program)이란 프로그램 텍스트를 전형적인(first-class) 값처럼 다루는 프로그램을 뜻한다. 즉, 실행 중에 프로그램의 원시 소스에 드러나지 않는 새로운 프로그램 텍스트를 조합하고 실행할 수 있다. 다단계 프로그램의 전형적인 예로는 부분 계산(partial evaluation) [2], 실시간 코드 생성(run-time code generation), 함수 펼치기(function inlining), 매크로 확장(macro expansion), 콰지-쿼트(quasi-quote) 시스템 [1]이 있다.

최근에 다단계 프로그램을 위한 let-다형 타입 체계(let-polymorphic type system)와 그 추론 시스템이 고안되었다 [3]. 그 타입 체계는 자유 변수를 포함한 코드 템플릿(open code template), 코드 템플릿에 대한 명령형 연산(imperative operation), 의도적으로 변수가 환경에 영향을 받도록 하는 치환(variable-capturing substitution), 환경에 영향을 받지 않도록 하는 치환(capture-avoiding substitution), 값을 코드로 변환(lifting)하는 연산을 모두 지원한다.

그러나, 타입 체계는 일반적으로 실제로 실행되지 않는 프로그램 부분에 대해서도 보수적으로 타입 검사를 수행하는 단점을 지니고 있다. 예를 들어, [3]의 타입 체계는 $\lambda x.(x\,x)$와 $\mathtt{box}_t\,(c\,c)$와 같은 프로그램을 받아들이지 못한다. 여기서, 함수 적용(function application)인 $(x\,x)$와 $(c\,c)$은 프로그램 실행 중에 실제 계산이 일어나지 않는 부분이다.

이 논문에서는 다단계 프로그램에 대한 흐름을 고려한 보다 정교한 분석을 제안한다. 안전할(sound) 뿐만 아니라, 단순 타입 체계보다 더 많은 올바른 프로그램을 받아들이는 것을 목표로 한다. 구체적으로, 위에서 말한 $\lambda x.(x\,x)$와 $\mathtt{box}_t\,(c\,c)$와 같은 프로그램도 올바른 프로그램으로 판단한다.

[1]이 연구는 정보통신부 선도기반기술개발사업과 교육인적자원부 두뇌한국21사업의 지원을 받았음을 밝힙니다.

$$
\begin{array}{rcl}
l & \in & \mathit{Label} \\
\\
e & \in & \mathit{Expr} \\
e & := & l : e' \\
e' & := & c \mid x \mid \lambda x.e \mid e\ e \mid \mathtt{box}_t\ e \mid \mathtt{unbox}_{k(>0)}\ e \mid \mathtt{eval}\ e
\end{array}
$$

[그림 1] 핵심 문법

분석은 집합 제약식(constraint-based) 분석의 형태로 표현한다. 단계가 없는(non-staged) 프로그램에 대한 Jens Palsbergs와 Michael I. Schwartzbach의 안전성 분석(safety analysis) [4]과 유사한 형태로 볼 수 있다.

2 언어

분석 대상 언어는 고차 함수(higher-order function)를 지원하는 값전달 호출(call-by-value) 방식의 전형적인 형태에 실행 중에 새로운 코드를 생성하고 실행할 수 있도록 하는 연산을 추가한 언어이다. 코드 템플릿 속에 포함된 자유 변수(free variable)가 그 코드 템플릿이 사용되는 환경에 따라 값이 달라지는 것을 허용한다.

2.1 핵심 문법

그림 1에 대상 언어의 핵심 문법(abstract syntax)을 나타내었다. 프로그램의 각 부분을 쉽게 지칭할 수 있도록, 프로그램의 모든 하부 수식에 표지(label)를 매달아 놓는다. 특별히 $\mathtt{box}_t$ 연산자들은 프로그램 어느 부분에서도 쉽게 지칭할 수 있도록 표지(t)를 매달아 놓는다. 이러한 것들은 논문의 가독성을 위한 것들로 프로그램의 의미 구조에는 영향을 미치지 않는다. 이후로 프로그램의 특정 부분을 지칭할 때, 해당 식으로 직접 표현하거나 표지를 사용해서 간접적으로 표현할 것이다.

2.2 의미 구조

그림 2는 대상 언어의 과정을 드러내는 의미 구조(operational semantics)를 보여준다. $\sigma \vdash e \xrightarrow{n} v$는 환경 σ을 사용하여 수식 e를 n 단계(stage)에서 계산하면 그 값이 v가 된다는 것을 의미한다.

다단계 프로그램에서는 값들을 단계 별로 분류할 수 있으며, 각각 v^0, v^1, $\cdots$로 표현한다.

$$
\begin{array}{rcl}
v & \in & \mathit{Val} = \mathit{Val}^0 + \mathit{Val}^1 + \cdots \\
v^n & \in & \mathit{Val}^n (n \geq 0) \\
v^0 & ::= & c \mid \langle \lambda x.e, \sigma \rangle \mid \mathtt{box}_t\ v^1 \\
v^n & ::= & c \mid x \mid \lambda x.v^n \mid v^n\ v^n \mid \mathtt{box}_t\ v^{n+1} \mid \mathtt{unbox}_k\ v^{n-k} \mid \mathtt{eval}\ v^n \quad (\text{단}, 0 < k < n)
\end{array}
$$

Var는 프로그램에 나타나는 모든 변수들의 유한 집합이다. 환경(environment)의 집합 Env는 변수들의 집합 Var에서 값의 집합 Val으로 가는 유한한 함수들의 집합이다.

$$\sigma \in \mathit{Env} = \mathit{Var} \xrightarrow{\text{fin}} \mathit{Val}$$

$$[\texttt{ECON}]\ \frac{}{\sigma \vdash c \xrightarrow{n} c}$$

$$[\texttt{EVAR}]\ \frac{\sigma(x) = v}{\sigma \vdash x \xrightarrow{0} v} \qquad [\texttt{EVAR}']\ \frac{}{\sigma \vdash x \xrightarrow{n+1} x}$$

$$[\texttt{EABS}]\ \frac{}{\sigma \vdash \lambda x.e \xrightarrow{0} \langle \lambda x.e, \sigma \rangle} \qquad [\texttt{EABS}']\ \frac{\sigma \vdash e \xrightarrow{n+1} v}{\sigma \vdash \lambda x.e \xrightarrow{n+1} \lambda x.v}$$

$$[\texttt{EAPP}]\ \frac{\sigma \vdash e_1 \xrightarrow{0} \langle \lambda x.e', \sigma' \rangle \qquad \sigma \vdash e_2 \xrightarrow{0} v_2 \qquad \sigma'\{x \mapsto v_2\} \vdash e' \xrightarrow{0} v}{\sigma \vdash e_1\ e_2 \xrightarrow{0} v}$$

$$[\texttt{EAPP}']\ \frac{\sigma \vdash e_1 \xrightarrow{n+1} v_1 \qquad \sigma \vdash e_2 \xrightarrow{n+1} v_2}{\sigma \vdash e_1\ e_2 \xrightarrow{n+1} v_1\ v_2}$$

$$[\texttt{EBOX}]\ \frac{\sigma \vdash e \xrightarrow{n+1} v}{\sigma \vdash \texttt{box}_t\ e \xrightarrow{n} \texttt{box}_t\ v}$$

$$[\texttt{EUNBOX}]\ \frac{\sigma \vdash e \xrightarrow{0} \texttt{box}_t\ v}{\sigma \vdash \texttt{unbox}_{n+1}\ e \xrightarrow{n+1} v} \qquad [\texttt{EUNBOX}']\ \frac{\sigma \vdash e \xrightarrow{n+1} v \qquad k > 0}{\sigma \vdash \texttt{unbox}_k\ e \xrightarrow{n+1+k} \texttt{unbox}_k\ v}$$

$$[\texttt{EEVAL}]\ \frac{\sigma \vdash e \xrightarrow{0} \texttt{box}_t\ v' \qquad \varnothing \vdash v' \xrightarrow{0} v}{\sigma \vdash \texttt{eval}\ e \xrightarrow{0} v} \qquad [\texttt{EEVAL}']\ \frac{\sigma \vdash e \xrightarrow{n+1} v}{\sigma \vdash \texttt{eval}\ e \xrightarrow{n+1} \texttt{eval}\ v}$$

[그림 2] 과정을 드러내는 의미 구조

대상 언어는 전통적인 람다 계산법(lambda calculus)에 실행 중에 코드 템플릿을 조합하고 계산할 수 있도록 하는 연산자들을 추가한 언어이다. 단계 0에서의 계산들 [ECON], [EVAR], [EABS], [EAPP]는 값전달 호출 방식의 람다 계산법의 그것과 동일하다.

[EBOX]는 코드 템플릿을 생성하는 규칙이다. 닫힌(closed) 코드 템플릿 뿐만 아니라 코드 템플릿에 자유 변수가 포함된 열린(open) 형태의 코드 템플릿도 허용한다. 코드 템플릿 안의 하부 수식은 현재 단계보다 한 단계 높은 단계에서 계산한다. 현재의 단계가 0보다 크다면, 코드 템플릿 안을 계산 중인 상태를 의미하는데, 그런 단계에서는 β-줄이기(β-reduction)가 일어나지 않고, [EUNBOX]에 의한 코드 치환 작업만 발생할 수 있다. [EUNBOX]에 의해 $\texttt{unbox}_k\ e$는 단계 n에서의 현재 코드 템플릿에 단계 $n-k$에서의 코드 템플릿을 삽입하는 역할을 한다. [EEVAL]은 하부 수식을 계산한 결과인 코드 템플릿 $\texttt{box}_t\ v'$을 프로그램 v'으로 변환한 다음, 다시 v'을 실행한다. 이 때, v'은 자유 변수가 없는 닫힌 코드 템플릿이어야 한다. 단계 0에서 자유롭게 α-변환을 수행할 수 있도록 하기 위함이다. 일반적으로, 코드 템플릿 안에서는 α-변환을 수행할 수 없다. 코드를 조합할 때 삽입되는 곳의 환경에 코드 템플릿 속에 포함된 자유 변수가 바인딩되기 때문에, 자유 변수의 이름을 그대로 보존해야 한다. 코드 템플릿 속에서 α-변환을 수행하면 전체 프로그램의 의미가 달라질 수 있다. 반면, [EEVAL]에서 오직 닫힌 코드 템플릿만 허용하도록 제한하기 때문에, 코드 템플릿 속에 있던 자유 변수가 단계 0에서 환경에 바인딩되는 일은 일어나지 않는다. 따라서, 단계 0에서는 자유롭게 α-변환을 수행할 수 있다.

$$[\texttt{TCON}]\quad \frac{}{\Gamma_0 \cdots \Gamma_n \vdash c : \iota}$$

$$[\texttt{TVAR}]\quad \frac{\Gamma_n(x) = \tau}{\Gamma_0 \cdots \Gamma_n \vdash x : \tau}$$

$$[\texttt{TABS}]\quad \frac{\Gamma_0 \cdots \Gamma_n + x : \tau_1 \vdash e : \tau_2}{\Gamma_0 \cdots \Gamma_n \vdash \lambda x.e : \tau_1 \to \tau_2}$$

$$[\texttt{TAPP}]\quad \frac{\Gamma_0 \cdots \Gamma_n \vdash e_1 : \tau_1 \to \tau_2 \qquad \Gamma_0 \cdots \Gamma_n \vdash e_2 : \tau_1}{\Gamma_0 \cdots \Gamma_n \vdash e_1\ e_2 : \tau_2}$$

$$[\texttt{TBOX}]\quad \frac{\Gamma_0 \cdots \Gamma_n \Gamma \vdash e : \tau}{\Gamma_0 \cdots \Gamma_n \vdash \texttt{box}_t\ e : \Box(\Gamma \triangleright \tau)}$$

$$[\texttt{TUNBOX}]\quad \frac{\Gamma_0 \cdots \Gamma_n \vdash e : \Box(\Gamma_{n+k} \triangleright \tau)}{\Gamma_0 \cdots \Gamma_{n+k} \vdash \texttt{unbox}_k\ e : \tau}$$

$$[\texttt{TEVAL}]\quad \frac{\Gamma_0 \cdots \Gamma_n \vdash e : \Box(\varnothing \triangleright \tau)}{\Gamma_0 \cdots \Gamma_n \vdash \texttt{eval}\ e : \tau}$$

[그림 3] 단순 타입 체계

3 타입 체계

그림 3은 대상 언어에 대한 단순 타입 체계(simple type system)를 보여주고 있다. [3]의 단순 타입 체계를 우리 언어에 맞게 정리한 것이다. $\Gamma_0 \cdots \Gamma_n \vdash e : \tau$는 타입 환경들 $\Gamma_0 \cdots \Gamma_n$ 하에서 단계 n 상의 수식 e는 타입 τ를 가짐을 의미한다.

세 가지 종류의 타입이 존재한다. 기저 타입(ι), 함수 타입($\tau \to \tau$), 코드 템플릿 타입($\Box(\Gamma \triangleright \tau)$)이 그것들이다. 타입 환경 Γ는 변수에서 타입으로 가는 유한 함수이다.

$$\begin{aligned} \tau &\in \mathit{Type} \\ \tau &::= \iota \mid \tau \to \tau \mid \Box(\Gamma \triangleright \tau) \end{aligned}$$

$$\Gamma \in \mathit{TypeEnv} = \mathit{Var} \xrightarrow{\text{fin}} \mathit{Type}$$

[TBOX]는 닫힌 코드 템플릿 뿐만 아니라 자유 변수를 포함한 열린 코드 템플릿도 타입을 가질 수 있도록 한다. $\Gamma_0 \cdots \Gamma_n \Gamma \vdash e : \tau$가 자유 변수를 포함하고 있는 수식 e가 타입을 가질 수 있게 하기 때문이다. 이 때, 코드 템플릿 타입 $\Box(\Gamma \triangleright \tau)$에서 조건 Γ가 τ 타입의 코드 템플릿 속에 포함된 자유 변수들의 타입을 지정한다. [TUNBOX]는 현재 단계의 타입 환경이 하부 수식에서 만들어진 코드 템플릿이 요구하는 조건을 충족하는 지를 검사한다. [TEVAL]은 하부 수식을 계산해서 얻어진 코드 템플릿이 자유 변수를 포함하지 않은 닫힌 코드 템플릿인지 검사한다.

정리 1 (안전성) *$\varnothing \vdash e : \tau$이고 $\varnothing \vdash e \xrightarrow{0} v$이면, $\varnothing \vdash v : \tau$이다.*

$$
\begin{array}{rcl}
sv & \in & SetVar \\
sv & := & \mathcal{X}_l^{\mathcal{Y}_t} \mid \mathcal{Y}_t^x \mid \mathcal{Z}_l \\
\\
sc & \in & SetCstr \\
sc & := & \mathcal{X}_l^{\mathcal{Y}_t} \supseteq \iota \mid \mathcal{X}_l^{\mathcal{Y}_t} \supseteq \mathcal{Y}_{t'}^x \to \mathcal{X}_{l'}^{\mathcal{Y}_{t'}} \mid \mathcal{X}_l^{\mathcal{Y}_t} \supseteq \Box \mathcal{X}_{l'}^{\mathcal{Y}_{t'}} \mid \\
 & & \mathcal{X}_l^{\mathcal{Y}_t} \supseteq apply(\mathcal{X}_{l'}^{\mathcal{Y}_t}, \mathcal{X}_{l''}^{\mathcal{Y}_t}) \mid \mathcal{X}_l^{\mathcal{Y}_t} \supseteq unbox_k\ \mathcal{X}_{l'}^{\mathcal{Y}_{t'}} \mid \mathcal{X}_l^{\mathcal{Y}_t} \supseteq eval\ \mathcal{X}_{l'}^{\mathcal{Y}_t} \mid \\
 & & \mathcal{Z}_l \supseteq true \mid \\
 & & \mathcal{Y}_t^x \subseteq \emptyset \mid \mathcal{Z}_l \Rightarrow \mathcal{Y}_t^x \subseteq \emptyset \mid \mathcal{Y}_t^x \not\subseteq \emptyset \mid \mathcal{Z}_l \Rightarrow \mathcal{Y}_t^x \not\subseteq \emptyset \mid \\
 & & \mathcal{X}_l^{\mathcal{Y}_t} \subseteq \text{LAMBDA} \mid \mathcal{Z}_l \Rightarrow \mathcal{X}_{l'}^{\mathcal{Y}_t} \subseteq \text{LAMBDA} \mid \mathcal{X}_l^{\mathcal{Y}_t} \subseteq \text{BOX} \mid \mathcal{Z}_l \Rightarrow \mathcal{X}_{l'}^{\mathcal{Y}_t} \subseteq \text{BOX} \mid \\
 & & \mathcal{X}_l^{\mathcal{Y}_t} \supseteq \mathcal{X}_{l'}^{\mathcal{Y}_{t'}} \mid \mathcal{X}_l^{\mathcal{Y}_t} \supseteq \mathcal{Y}_{t'}^x \mid \mathcal{Y}_t^x \supseteq \mathcal{X}_l^{\mathcal{Y}_{t'}} \mid \mathcal{Y}_t^x \supseteq \mathcal{Y}_{t'}^x \mid \mathcal{Z}_l \supseteq \mathcal{Z}_{l'}
\end{array}
$$

[그림 4] 집합 제약식

4 집합 제약식 분석

집합 제약식 분석은 프로그램을 훑어 집합 제약식들을 도출하고, 도출된 집합 제약식을 푸는 과정으로 이루어진다.

4.1 집합 제약식

그림 4에 분석에 사용하는 집합 제약식을 나타내었다.

집합 변수 $\mathcal{X}_l^{\mathcal{Y}_t}$, $\mathcal{Y}_t^x$는 각각 타입 체계의 타입 τ과 타입 환경 Γ에 대응하는 정보를 모으는 변수들이다. $\mathcal{X}_l^{\mathcal{Y}_t}$은 타입 환경 $\mathcal{Y}_t^x, \mathcal{Y}_t^y, \cdots$ 하에서 실행되는 하부 수식 $l : e$이 가지는 타입 정보를 모으고, $\mathcal{Y}_t^x$은 프로그램 내의 $\texttt{box}_t$ 내부의 변수 x에 바인딩되는 값들을 모은다. 이와 달리, $\mathcal{Z}_l$는 타입 체계에는 대응되는 개념이 존재하지 않는 것으로 하부 수식 $l : e$에 대해 타입 검사를 할 필요가 있는 지를 보수적으로 판단하는 변수이다.

다양한 형태의 집합 제약식이 존재하지만, 근본적으로는 $X \supseteq Y$, $X \subseteq Y$, $X \not\subseteq Y$, $X \Rightarrow \cdots$ 형태로 분류할 수 있다. $X \supseteq Y$ 형태의 제약식은 X가 의미하는 집합이 Y가 의미하는 집합을 포함한다는 것을 의미한다. 직관적으로, 실행 중에 Y의 정보가 X로 흘러들어 가는 것을 묘사한다. $X \subseteq Y$와 $X \not\subseteq Y$ 형태의 제약식은 X가 의미하는 집합이 Y가 의미하는 집합의 부분집합이어야 한다 혹은 그렇지 않아야 한다는 것을 나타내는 것들이다. 해당 수식이 어떤 타입을 가져야 하는 지를 제한한다. $X \Rightarrow \cdots$는 화살표 오른쪽에 있는 제약식이 X가 참일 경우에만 적용된다는 뜻으로, 화살표 오른쪽에 있는 제약식의 적용을 미루기 위한 용도로 사용된다. 각 집합 제약식의 구체적 의미는 그림 5에서 정의한다.

4.2 집합 제약식 도출

우선, 단계 0에서는 자유롭게 α-변환할 수 있으므로 단계 0에서 바운딩되는 변수들은 적절히 α-변환하여 모두 다르다고 가정한다. 이런 변환 없이 분석하는 것도 가능하나 분석의 정확도가 떨어질 수 있다.

그림 6은 프로그램으로부터 집합 제약식을 도출하는 방법을 정의한다. 여기서, $\mathcal{Y}_{t_0} \cdots \mathcal{Y}_{t_n} \vdash l : e \triangleright \mathcal{C}$는 수식 $l : e$를 단계 n에서 계산하게 되면 제약식 $\mathcal{C}$을 도출한다는 뜻이다. l_0는 전체 프로그램의 표지(label)를 의미한다.

단계 0에서 반드시 실행이 되는 수식이나 단계 $n+1$에서의 단계 0까지 내려가는 $\texttt{unbox}_{n+1}\ e$에 대해서는 타입을 검사하는 $X \subseteq Y$ 형태의 제약식을 도출한다. 특정 단계

$$
\begin{array}{rcl}
h_t & \in & Herb_T \\
h_t & := & \iota \mid x \to h_t \mid \Box\, h_t \\
h_b & \in & Herb_B \\
h_b & := & true
\end{array}
$$

$$
\begin{array}{rcl}
\Sigma & \in & SetVar \to 2^{Herb_T} \\
\Delta & \in & SetVar \to 2^{Herb_B}
\end{array}
$$

$$
\begin{array}{rcl}
[\![\cdot]\!] & \in & SetExpr \to (SetVar \to 2^{Herb_T}) \to (SetVar \to 2^{Herb_B}) \to 2^{Herb} \\
[\![\mathcal{X}_l^{\mathcal{Y}_t}]\!]\,\Sigma\,\Delta & = & \Sigma(\mathcal{X}_l^{\mathcal{Y}_t}) \\
[\![\mathcal{Y}_t^x]\!]\,\Sigma\,\Delta & = & \Sigma(\mathcal{Y}_t^x) \\
[\![\mathcal{Z}_l]\!]\,\Sigma\,\Delta & = & \Delta(\mathcal{Z}_l) \\
[\![\iota]\!]\,\Sigma\,\Delta & = & \{\iota\} \\
[\![\mathcal{Y}_t^x \to \mathcal{X}_l^{\mathcal{Y}_t}]\!]\,\Sigma\,\Delta & = & \{x \to h_t \mid h_t \in [\![\mathcal{X}_l^{\mathcal{Y}_t}]\!]\,\Sigma\,\Delta\} \\
[\![\Box\, \mathcal{X}_l^{\mathcal{Y}_t}]\!]\,\Sigma\,\Delta & = & \{\Box\, h_t \mid h_t \in [\![\mathcal{X}_l^{\mathcal{Y}_t}]\!]\,\Sigma\,\Delta\} \\
[\![apply(\mathcal{X}_l^{\mathcal{Y}_t}, \mathcal{X}_{l'}^{\mathcal{Y}_t})]\!]\,\Sigma\,\Delta & = & \{h_t \mid x \to h_t \in [\![\mathcal{X}_l^{\mathcal{Y}_t}]\!]\,\Sigma\,\Delta\} \\
[\![unbox_k\, \mathcal{X}_l^{\mathcal{Y}_t}]\!]\,\Sigma\,\Delta & = & \{h_t \mid \Box\, h_t \in [\![\mathcal{X}_l^{\mathcal{Y}_t}]\!]\,\Sigma\,\Delta\} \\
[\![eval\, \mathcal{X}_l^{\mathcal{Y}_t}]\!]\,\Sigma\,\Delta & = & \{h_t \mid \Box\, h_t \in [\![\mathcal{X}_l^{\mathcal{Y}_t}]\!]\,\Sigma\,\Delta\} \\
[\![true]\!]\,\Sigma\,\Delta & = & \{true\} \\
[\![\emptyset]\!]\,\Sigma\,\Delta & = & \emptyset \\
[\![\text{LAMBDA}]\!]\,\Sigma\,\Delta & = & \{x \to h_t \mid x \to h_t \in Herb_T\} \\
[\![\text{BOX}]\!]\,\Sigma\,\Delta & = & \{\Box\, h_t \mid \Box\, h_t \in Herb_T\}
\end{array}
$$

[그림 5] 집합 제약식의 의미

에서 실행될 경우 타입 오류가 발생할 수 있지만 실제로 그 단계에서 실행될 지를 알 지 못하는 수식에 대해서는 $X \subseteq Y$ 대신 타입 검사를 일단 보류하는 $Z \Rightarrow X \subseteq Y$ 형태의 제약식을 도출한다. 집합 제약식을 푸는 과정에서 해당 수식이 그 단계에서 실행될 가능성이 있다고 판단되면(즉, 조건 Z가 참이 되면) $X \subseteq Y$ 제약식을 추가함으로써 보류되었던 타입 검사를 수행한다.

각 제약식 도출 규칙을 개별적으로 살펴보자.

$$\frac{}{\mathcal{Y}_{t_0} \cdots \mathcal{Y}_{t_n} \vdash l : c \triangleright \{\mathcal{X}_l^{\mathcal{Y}_{t_n}} \supseteq \iota\}}$$

상수 c는 현재 단계 n에 무관하게 ι의 타입을 가진다.

$$\frac{}{\mathcal{Y}_{t_0} \vdash l : x \triangleright \{\mathcal{X}_l^{\mathcal{Y}_{t_0}} \supseteq \mathcal{Y}_{t_0}^x, \mathcal{Y}_{t_0}^x \not\subseteq \emptyset, \mathcal{Z}_l \supseteq true\}}$$

단계 0에서 변수 x는 함수 적용을 통해 타입 환경($\mathcal{Y}_{t_0}^x$)에 바인딩된 타입을 가진다. 단계 0에서는 변수 x가 자유 변수일 수 없으며($\mathcal{Y}_{t_0}^x \not\subseteq \emptyset$), 단계 0에서 일어나는 계산이므로 타입을 검사한다($\mathcal{Z}_l \supseteq true$).

$$\frac{}{\mathcal{Y}_{t_0} \cdots \mathcal{Y}_{t_{n+1}} \vdash l : x \triangleright \{\mathcal{X}_l^{\mathcal{Y}_{t_{n+1}}} \supseteq \mathcal{Y}_{t_{n+1}}^x, \mathcal{Z}_l \Rightarrow \mathcal{Y}_{t_{n+1}}^x \not\subseteq \emptyset\}}$$

단계가 0보다 큰 경우에도 여전히 $\mathcal{Y}_{t_{n+1}}^x$에서 타입 정보를 가져온다. 그러나, 이 수식에 대해 타입 검사를 할 필요가 있다고 판단될 때만($\mathcal{Z}_l$이 참일 때만), 타입 검사를 수행하도록 한다($\mathcal{Z}_l \Rightarrow \mathcal{Y}_{l_{n+1}}^x \not\subseteq \emptyset$).

$$\frac{\mathcal{Y}_{t_0} \cdots \mathcal{Y}_{t_n} \vdash e \triangleright \mathcal{C}}{\mathcal{Y}_{t_0} \cdots \mathcal{Y}_{t_n} \vdash l : \lambda x.e \triangleright \{\mathcal{X}_l^{\mathcal{Y}_{t_n}} \supseteq \mathcal{Y}_{t_n}^x \to \mathcal{X}_e^{\mathcal{Y}_{t_n}}, \mathcal{Z}_e \supseteq \mathcal{Z}_l\} \cup \mathcal{C}}$$

$$\frac{}{\mathcal{Y}_{t_0}\cdots\mathcal{Y}_{t_n}\vdash l : c \triangleright \{\mathcal{X}_l^{\mathcal{Y}_{t_n}} \supseteq \iota\}}$$

$$\frac{}{\mathcal{Y}_{t_0}\vdash l : x \triangleright \{\mathcal{X}_l^{\mathcal{Y}_{t_0}} \supseteq \mathcal{Y}_{t_0}^x, \mathcal{Y}_{t_0}^x \not\subseteq \emptyset, \mathcal{Z}_l \supseteq \mathit{true}\}}$$

$$\frac{}{\mathcal{Y}_{t_0}\cdots\mathcal{Y}_{t_{n+1}}\vdash l : x \triangleright \{\mathcal{X}_l^{\mathcal{Y}_{t_{n+1}}} \supseteq \mathcal{Y}_{t_{n+1}}^x, \mathcal{Z}_l \Rightarrow \mathcal{Y}_{t_{n+1}}^x \not\subseteq \emptyset\}}$$

$$\frac{\mathcal{Y}_{t_0}\cdots\mathcal{Y}_{t_n}\vdash e \triangleright \mathcal{C}}{\mathcal{Y}_{t_0}\cdots\mathcal{Y}_{t_n}\vdash l : \lambda x.e \triangleright \{\mathcal{X}_l^{\mathcal{Y}_{t_n}} \supseteq \mathcal{Y}_{t_n}^x \rightarrow \mathcal{X}_e^{\mathcal{Y}_{t_n}}, \mathcal{Z}_e \supseteq \mathcal{Z}_l\} \cup \mathcal{C}}$$

$$\frac{\mathcal{Y}_{t_0}\vdash e_1 \triangleright \mathcal{C}_1 \qquad \mathcal{Y}_{t_0}\vdash e_2 \triangleright \mathcal{C}_2}{\mathcal{Y}_{t_0}\vdash l : e_1\ e_2 \triangleright \{\mathcal{X}_l^{\mathcal{Y}_{t_0}} \supseteq \mathit{apply}(\mathcal{X}_{e_1}^{\mathcal{Y}_{t_0}}, \mathcal{X}_{e_2}^{\mathcal{Y}_{t_0}}), \mathcal{X}_{e_1}^{\mathcal{Y}_{t_0}} \subseteq \mathrm{LAMBDA}, \mathcal{Z}_l \supseteq \mathit{true}\} \cup \mathcal{C}_1 \cup \mathcal{C}_2}$$

$$\frac{\mathcal{Y}_{t_0}\cdots\mathcal{Y}_{t_{n+1}}\vdash e_1 \triangleright \mathcal{C}_1 \qquad \mathcal{Y}_{t_0}\cdots\mathcal{Y}_{t_{n+1}}\vdash e_2 \triangleright \mathcal{C}_2}{\mathcal{Y}_{t_0}\cdots\mathcal{Y}_{t_{n+1}}\vdash l : e_1\ e_2 \triangleright \{\mathcal{X}_l^{\mathcal{Y}_{t_{n+1}}} \supseteq \mathit{apply}(\mathcal{X}_{e_1}^{\mathcal{Y}_{t_{n+1}}}, \mathcal{X}_{e_2}^{\mathcal{Y}_{t_{n+1}}}), \mathcal{Z}_l \Rightarrow \mathcal{X}_{e_1}^{\mathcal{Y}_{t_{n+1}}} \subseteq \mathrm{LAMBDA}, \mathcal{Z}_{e_1} \supseteq \mathcal{Z}_l, \mathcal{Z}_{e_2} \supseteq \mathcal{Z}_l\} \cup \mathcal{C}_1 \cup \mathcal{C}_2}$$

$$\frac{\mathcal{Y}_{t_0}\cdots\mathcal{Y}_{t_n}\mathcal{Y}_t\vdash e \triangleright \mathcal{C}}{\mathcal{Y}_{t_0}\cdots\mathcal{Y}_{t_n}\vdash l : \mathtt{box}_t\ e \triangleright \{\mathcal{X}_l^{\mathcal{Y}_{t_n}} \supseteq \Box\,\mathcal{X}_e^{\mathcal{Y}_t}, \mathcal{Z}_e \supseteq \mathcal{Z}_l\} \cup \mathcal{C}}$$

$$\frac{\mathcal{Y}_{t_0}\vdash e \triangleright \mathcal{C}}{\mathcal{Y}_{t_0}\cdots\mathcal{Y}_{t_{n+1}}\vdash l : \mathtt{unbox}_{n+1}\ e \triangleright \{\mathcal{X}_l^{\mathcal{Y}_{t_{n+1}}} \supseteq \mathit{unbox}_{n+1}\,\mathcal{X}_e^{\mathcal{Y}_{t_0}}, \mathcal{X}_e^{\mathcal{Y}_{t_0}} \subseteq \mathrm{BOX}, \mathcal{Z}_l \supseteq \mathit{true}\} \cup \mathcal{C}}$$

$$\frac{\mathcal{Y}_{t_0}\cdots\mathcal{Y}_{t_{n+1}}\vdash e \triangleright \mathcal{C} \qquad k > 0}{\mathcal{Y}_{t_0}\cdots\mathcal{Y}_{t_{n+1+k}}\vdash l : \mathtt{unbox}_k\ e \triangleright \{\mathcal{X}_l^{\mathcal{Y}_{t_{n+1+k}}} \supseteq \mathit{unbox}_k\,\mathcal{X}_e^{\mathcal{Y}_{t_{n+1}}}, \mathcal{Z}_l \Rightarrow \mathcal{X}_e^{\mathcal{Y}_{t_{n+1}}} \subseteq \mathrm{BOX}, \mathcal{Z}_e \supseteq \mathcal{Z}_l\} \cup \mathcal{C}}$$

$$\frac{\mathcal{Y}_{t_0}\vdash e \triangleright \mathcal{C}}{\mathcal{Y}_{t_0}\vdash l : \mathtt{eval}\ e \triangleright \{\mathcal{X}_l^{\mathcal{Y}_{t_0}} \supseteq \mathit{eval}\,\mathcal{X}_e^{\mathcal{Y}_{t_0}}, \mathcal{X}_e^{\mathcal{Y}_{t_0}} \subseteq \mathrm{BOX}, \mathcal{Z}_l \supseteq \mathit{true}\} \cup \mathcal{C}}$$

$$\frac{\mathcal{Y}_{t_0}\cdots\mathcal{Y}_{t_{n+1}}\vdash e \triangleright \mathcal{C}}{\mathcal{Y}_{t_0}\cdots\mathcal{Y}_{t_{n+1}}\vdash l : \mathtt{eval}\ e \triangleright \{\mathcal{X}_l^{\mathcal{Y}_{t_{n+1}}} \supseteq \mathit{eval}\,\mathcal{X}_e^{\mathcal{Y}_{t_{n+1}}}, \mathcal{Z}_l \Rightarrow \mathcal{X}_e^{\mathcal{Y}_{t_{n+1}}} \subseteq \mathrm{BOX}, \mathcal{Z}_e \supseteq \mathcal{Z}_l\} \cup \mathcal{C}}$$

[그림 6] 집합 제약식 도출

람다 추상화(lambda abstraction)는 단계에 상관없이 인자를 받아서 몸체(body)를 계산해서 얻어지는 타입을 가진다($\mathcal{X}_l^{\mathcal{Y}_{t_n}} \supseteq \mathcal{Y}_{t_n}^x \rightarrow \mathcal{X}_e^{\mathcal{Y}_{t_n}}$). 이 수식에 대해 타입 검사를 할 필요가 있다면, 하부 수식에 대해서도 타입을 검사할 필요가 있다($\mathcal{Z}_e \supseteq \mathcal{Z}_l$).

$$\frac{\mathcal{Y}_{t_0}\vdash e_1 \triangleright \mathcal{C}_1 \qquad \mathcal{Y}_{t_0}\vdash e_2 \triangleright \mathcal{C}_2}{\mathcal{Y}_{t_0}\vdash l : e_1\ e_2 \triangleright \{\mathcal{X}_l^{\mathcal{Y}_{t_0}} \supseteq \mathit{apply}(\mathcal{X}_{e_1}^{\mathcal{Y}_{t_0}}, \mathcal{X}_{e_2}^{\mathcal{Y}_{t_0}}), \mathcal{X}_{e_1}^{\mathcal{Y}_{t_0}} \subseteq \mathrm{LAMBDA}, \mathcal{Z}_l \supseteq \mathit{true}\} \cup \mathcal{C}_1 \cup \mathcal{C}_2}$$

단계 0에서의 함수 적용의 타입은 먼저 e_1의 타입을 알아야 알 수 있으므로, 제약식 도출 과정에서는 함수 적용이다는 사실만을 기록하고 실제 타입 정보는 제약식을 푸는 과정에서 얻는다($\mathcal{X}_l^{\mathcal{Y}_{t_0}} \supseteq \mathit{apply}(\mathcal{X}_{e_1}^{\mathcal{Y}_{t_0}}, \mathcal{X}_{e_2}^{\mathcal{Y}_{t_0}})$). 단계 0에서의 함수 적용이므로 e_1의 타입은 반드시 함

수 타입이어야 한다($\mathcal{X}_{e_1}^{\mathcal{Y}_{t_0}} \subseteq \text{LAMBDA}$, $\mathcal{Z}_l \supseteq \mathit{true}$).

$$\frac{\mathcal{Y}_{t_0} \cdots \mathcal{Y}_{t_{n+1}} \vdash e_1 \triangleright \mathcal{C}_1 \qquad \mathcal{Y}_{t_0} \cdots \mathcal{Y}_{t_{n+1}} \vdash e_2 \triangleright \mathcal{C}_2}{\mathcal{Y}_{t_0} \cdots \mathcal{Y}_{t_{n+1}} \vdash l : e_1\ e_2 \triangleright \{\mathcal{X}_l^{\mathcal{Y}_{t_{n+1}}} \supseteq \mathit{apply}(\mathcal{X}_{e_1}^{\mathcal{Y}_{t_{n+1}}}, \mathcal{X}_{e_2}^{\mathcal{Y}_{t_{n+1}}}), \mathcal{Z}_l \Rightarrow \mathcal{X}_{e_1}^{\mathcal{Y}_{t_{n+1}}} \subseteq \text{LAMBDA}, \mathcal{Z}_{e_1} \supseteq \mathcal{Z}_l, \mathcal{Z}_{e_2} \supseteq \mathcal{Z}_l\} \cup \mathcal{C}_1 \cup \mathcal{C}_2}$$

현재 단계가 0보다 큰 경우에도 함수 적용에 관한 제약식을 도출하지만($\mathcal{X}_l^{\mathcal{Y}_{t_{n+1}}} \supseteq \mathit{apply}(\mathcal{X}_{e_1}^{\mathcal{Y}_{t_{n+1}}}, \mathcal{X}_{e_2}^{\mathcal{Y}_{t_{n+1}}})$), 타입을 제한하는 제약식은 해당 식이 타입을 검사할 필요가 있을 때만 수행되도록 한다($\mathcal{Z}_l \Rightarrow \mathcal{X}_{e_1}^{\mathcal{Y}_{t_{n+1}}} \subseteq \text{LAMBDA}$). 해당 수식에 대해 타입 검사를 수행해야 한다면, 하부 수식들에 대해서도 타입 검사를 수행해야 한다($\mathcal{Z}_{e_1} \supseteq \mathcal{Z}_l$, $\mathcal{Z}_{e_2} \supseteq \mathcal{Z}_l$).

$$\frac{\mathcal{Y}_{t_0} \cdots \mathcal{Y}_{t_n} \mathcal{Y}_t \vdash e \triangleright \mathcal{C}}{\mathcal{Y}_{t_0} \cdots \mathcal{Y}_{t_n} \vdash l : \texttt{box}_t\ e \triangleright \{\mathcal{X}_l^{\mathcal{Y}_{t_n}} \supseteq \Box\, \mathcal{X}_e^{\mathcal{Y}_t}, \mathcal{Z}_e \supseteq \mathcal{Z}_l\} \cup \mathcal{C}}$$

수식 $\texttt{box}_t\ e$의 타입은 하부 수식의 타입을 코드 템플릿으로 만든 타입이다($\mathcal{X}_l^{\mathcal{Y}_{t_n}} \supseteq \Box\, \mathcal{X}_e^{\mathcal{Y}_t}$). 해당 수식에 대해 타입을 검사할 필요가 있다면 하부 수식에 대해서도 타입을 검사할 필요가 있다는 보수적인(conservative) 입장을 취한다($\mathcal{Z}_e \supseteq \mathcal{Z}_l$)

$$\frac{\mathcal{Y}_{t_0} \vdash e \triangleright \mathcal{C}}{\mathcal{Y}_{t_0} \cdots \mathcal{Y}_{t_{n+1}} \vdash l : \texttt{unbox}_{n+1}\ e \triangleright \{\mathcal{X}_l^{\mathcal{Y}_{t_{n+1}}} \supseteq \mathit{unbox}_{n+1} \mathcal{X}_e^{\mathcal{Y}_{t_0}}, \mathcal{X}_e^{\mathcal{Y}_{t_0}} \subseteq \text{BOX}, \mathcal{Z}_l \supseteq \mathit{true}\} \cup \mathcal{C}}$$

$\texttt{unbox}_{n+1}\ e$는 함수 적용과 마찬가지로 하부 수식의 타입 정보를 알아야만 타입을 결정할 수 있다($\mathcal{X}_l^{\mathcal{Y}_{t_{n+1}}} \supseteq \mathit{unbox}_{n+1} \mathcal{X}_e^{\mathcal{Y}_{t_0}}$). 그러나, 단계 0까지 내려가므로 반드시 하부 수식의 타입은 코드 템플릿의 타입이어야 한다($\mathcal{X}_e^{\mathcal{Y}_{t_0}} \subseteq \text{BOX}$, $\mathcal{Z}_l \supseteq \mathit{true}$).

$$\frac{\mathcal{Y}_{t_0} \cdots \mathcal{Y}_{t_{n+1}} \vdash e \triangleright \mathcal{C} \qquad k > 0}{\mathcal{Y}_{t_0} \cdots \mathcal{Y}_{t_{n+1+k}} \vdash l : \texttt{unbox}_k\ e \triangleright \{\mathcal{X}_l^{\mathcal{Y}_{t_{n+1+k}}} \supseteq \mathit{unbox}_k\, \mathcal{X}_e^{\mathcal{Y}_{t_{n+1}}}, \mathcal{Z}_l \Rightarrow \mathcal{X}_e^{\mathcal{Y}_{t_{n+1}}} \subseteq \text{BOX}, \mathcal{Z}_e \supseteq \mathcal{Z}_l\} \cup \mathcal{C}}$$

단계 0까지 내려가지 못하는 $\texttt{unbox}_k\ e$도 하부 수식의 타입에 따라 타입이 결정되지만($\mathcal{X}_l^{\mathcal{Y}_{t_{n+1+k}}} \supseteq \mathit{unbox}_k\, \mathcal{X}_e^{\mathcal{Y}_{t_{n+1}}}$), 해당 수식에 대해 타입 검사를 반드시 수행해야 하는 것은 아니다($\mathcal{Z}_l \Rightarrow \mathcal{X}_e^{\mathcal{Y}_{t_{n+1}}} \subseteq \text{BOX}$). 그러나, 일단 타입 검사를 수행한다면 하부 수식에 대해서도 타입 검사를 수행해야 한다($\mathcal{Z}_e \supseteq \mathcal{Z}_l$).

$$\frac{\mathcal{Y}_{t_0} \vdash e \triangleright \mathcal{C}}{\mathcal{Y}_{t_0} \vdash l : \texttt{eval}\ e \triangleright \{\mathcal{X}_l^{\mathcal{Y}_{t_0}} \supseteq \mathit{eval}\, \mathcal{X}_e^{\mathcal{Y}_{t_0}}, \mathcal{X}_e^{\mathcal{Y}_{t_0}} \subseteq \text{BOX}, \mathcal{Z}_l \supseteq \mathit{true}\} \cup \mathcal{C}}$$

단계 0에서 $\texttt{eval}\ e$을 수행한 결과값의 타입 정보는 하부 수식의 값의 타입에 따라 달라진다($\mathcal{X}_l^{\mathcal{Y}_{t_0}} \supseteq \mathit{eval}\, \mathcal{X}_e^{\mathcal{Y}_{t_0}}$). 이 때, 하부 수식은 반드시 코드 템플릿의 타입을 가져야 한다($\mathcal{X}_e^{\mathcal{Y}_{t_0}} \subseteq \text{BOX}, \mathcal{Z}_l \supseteq \mathit{true}$).

$$\frac{\mathcal{Y}_{t_0} \cdots \mathcal{Y}_{t_{n+1}} \vdash e \triangleright \mathcal{C}}{\mathcal{Y}_{t_0} \cdots \mathcal{Y}_{t_{n+1}} \vdash l : \texttt{eval}\ e \triangleright \{\mathcal{X}_l^{\mathcal{Y}_{t_{n+1}}} \supseteq \mathit{eval}\, \mathcal{X}_e^{\mathcal{Y}_{t_{n+1}}}, \mathcal{Z}_l \Rightarrow \mathcal{X}_e^{\mathcal{Y}_{t_{n+1}}} \subseteq \text{BOX}, \mathcal{Z}_e \supseteq \mathcal{Z}_l\} \cup \mathcal{C}}$$

단계 0보다 높은 단계에서 수행한 $\texttt{eval}\ e$도 하부 수식의 타입에 따라 그 타입이 결정된다($\mathcal{X}_l^{\mathcal{Y}_{t_{n+1}}} \supseteq \mathit{eval}\, \mathcal{X}_e^{\mathcal{Y}_{t_{n+1}}}$). 해당 수식이 결코 단계 0에서 수행되지 않을 수도 있으므로, 필요하다고 판단될 때만 타입을 검사하는 형태의 제약식을 도출한다($\mathcal{Z}_l \Rightarrow \mathcal{X}_e^{\mathcal{Y}_{t_{n+1}}} \subseteq \text{BOX}$).

$$\frac{\mathcal{X}_l^{\mathcal{Y}_t} \supseteq apply(\mathcal{X}_{l'}^{\mathcal{Y}_t}, \mathcal{X}_{l''}^{\mathcal{Y}_t}) \qquad \mathcal{X}_{l'}^{\mathcal{Y}_t} \supseteq \mathcal{Y}_{t'}^{x} \to \mathcal{X}_{l'''}^{\mathcal{Y}_{t'}}}{\{\mathcal{Y}_{t'}^{x} \supseteq \mathcal{X}_{l''}^{\mathcal{Y}_t}, \mathcal{X}_l^{\mathcal{Y}_t} \supseteq \mathcal{X}_{l'''}^{\mathcal{Y}_{t'}}, \mathcal{Z}_{l'''} \supseteq \mathcal{Z}_l\}}$$

$$\frac{\mathcal{X}_l^{\mathcal{Y}_t} \supseteq unbox_k\, \mathcal{X}_{l'}^{\mathcal{Y}_{t'}} \qquad \mathcal{X}_{l'}^{\mathcal{Y}_{t'}} \supseteq \Box\, \mathcal{X}_{l''}^{\mathcal{Y}_{t''}}}{\{\mathcal{X}_l^{\mathcal{Y}_t} \supseteq \mathcal{X}_{l'''}^{\mathcal{Y}_{t''}}, \mathcal{Z}_{l''} \supseteq \mathcal{Z}_l\}} \qquad \frac{\mathcal{X}_l^{\mathcal{Y}_t} \supseteq unbox_k\, \mathcal{X}_{l'}^{\mathcal{Y}_{t'}} \qquad \mathcal{X}_{l'}^{\mathcal{Y}_{t'}} \supseteq \Box\, \mathcal{X}_{l''}^{\mathcal{Y}_{t''}} \qquad x \in FV^0(l'')}{\{\mathcal{Y}_{t''}^{x} \supseteq \mathcal{Y}_t^{x}\}}$$

$$\frac{\mathcal{X}_l^{\mathcal{Y}_t} \supseteq eval\, \mathcal{X}_{l'}^{\mathcal{Y}_t} \qquad \mathcal{X}_{l'}^{\mathcal{Y}_t} \supseteq \Box\, \mathcal{X}_{l'''}^{\mathcal{Y}_{t'}}}{\{\mathcal{X}_l^{\mathcal{Y}_t} \supseteq \mathcal{X}_{l''}^{\mathcal{Y}_{t'}}, \mathcal{Z}_{l''} \supseteq \mathcal{Z}_l\}} \qquad \frac{\mathcal{X}_l^{\mathcal{Y}_t} \supseteq eval\, \mathcal{X}_{l'}^{\mathcal{Y}_t} \qquad \mathcal{X}_{l'}^{\mathcal{Y}_t} \supseteq \Box\, \mathcal{X}_{l'''}^{\mathcal{Y}_{t'}} \qquad x \in FV^0(l'')}{\{\mathcal{Z}_l \Rightarrow \mathcal{Y}_{t'}^{x} \subseteq \emptyset, \mathcal{Z}_l \Rightarrow \mathcal{Y}_{t'}^{x} \not\subseteq \emptyset\}}$$

$$\frac{\mathcal{Z}_l \Rightarrow \mathcal{Y}_t^{x} \subseteq \emptyset \qquad \mathcal{Z}_l \supseteq true}{\mathcal{Y}_t^{x} \subseteq \emptyset} \qquad \frac{\mathcal{Z}_l \Rightarrow \mathcal{Y}_t^{x} \not\subseteq \emptyset \qquad \mathcal{Z}_l \supseteq true}{\mathcal{Y}_t^{x} \not\subseteq \emptyset}$$

$$\frac{\mathcal{Z}_l \Rightarrow \mathcal{X}_e^{\mathcal{Y}_t} \subseteq \text{LAMBDA} \qquad \mathcal{Z}_l \supseteq true}{\mathcal{X}_e^{\mathcal{Y}_t} \subseteq \text{LAMBDA}} \qquad \frac{\mathcal{Z}_l \Rightarrow \mathcal{X}_e^{\mathcal{Y}_t} \subseteq \text{BOX} \qquad \mathcal{Z}_l \supseteq true}{\mathcal{X}_e^{\mathcal{Y}_t} \subseteq \text{BOX}}$$

$$\frac{sv_1 \supseteq sv_2 \qquad sv_2 \supseteq \iota}{sv_1 \supseteq \iota} \qquad \frac{sv_1 \supseteq sv_2 \qquad sv_2 \supseteq sv_3 \to sv_4}{sv_1 \supseteq sv_3 \to sv_4} \qquad \frac{sv_1 \supseteq sv_2 \qquad sv_2 \supseteq \Box\, sv_3}{sv_1 \supseteq \Box\, sv_3}$$

$$\frac{sv_1 \supseteq sv_2 \qquad sv_2 \supseteq true}{sv_1 \supseteq true}$$

[그림 7] 집합 제약식 풀기

$$\begin{array}{rcll}
FV^n(c) & = & \emptyset & \\
FV^n(x) & = & \{x\} & \text{if } n = 0 \\
 & = & \emptyset & \text{otherwise} \\
FV^n(\lambda x.e) & = & FV^n(e) - \{x\} & \text{if } n = 0 \\
 & = & FV^n(e) & \text{otherwise} \\
FV^n(e_1\, e_2) & = & FV^n(e_1) \cup FV^n(e_2) & \\
FV^n(\texttt{box}_t\, e) & = & FV^{n+1}(e) & \\
FV^n(\texttt{unbox}_k\, e) & = & FV^{n-k}(e) & \text{if } n - k \geq 0 \\
 & = & \emptyset & \text{otherwise} \\
FV^n(\texttt{eval}\, e) & = & FV^n(e) &
\end{array}$$

[그림 8] 자유 변수

4.3 집합 제약식 풀기

집합 제약식을 푸는 방법은 그림 7과 같다. 기본적으로 기존의 제약식에 위의 제약식들이 존재하면 아래의 새로운 제약식을 추가하는 형태이다. 제약식 집합의 최대 크기가 분석할 프로그램에 의해 한정되므로 푸는 과정은 유한한 시간 내에 끝난다.

해는 도출한 제약식들을 끝까지 풀어낸 각 집합 변수의 제약식들 중에서 최소 알갱이로 풀려진 제약식(atomic constraint)로 구성된다. 그림 9에서 최소 알갱이로 풀려진 제약식을 정의한다.

제약식을 푸는 규칙을 좀더 자세히 살펴보자.

$$\frac{\mathcal{X}_l^{\mathcal{Y}_t} \supseteq apply(\mathcal{X}_{l'}^{\mathcal{Y}_t}, \mathcal{X}_{l''}^{\mathcal{Y}_t}) \qquad \mathcal{X}_{l'}^{\mathcal{Y}_t} \supseteq \mathcal{Y}_{t'}^{x} \to \mathcal{X}_{l'''}^{\mathcal{Y}_{t'}}}{\{\mathcal{Y}_{t'}^{x} \supseteq \mathcal{X}_{l''}^{\mathcal{Y}_t}, \mathcal{X}_l^{\mathcal{Y}_t} \supseteq \mathcal{X}_{l'''}^{\mathcal{Y}_{t'}}, \mathcal{Z}_{l'''} \supseteq \mathcal{Z}_l\}}$$

$$
\begin{array}{rcl}
ac & \in & AtomSetCstr \\
ac & := & \mathcal{X}_l^{\mathcal{Y}_t} \supseteq \iota \mid \mathcal{X}_l^{\mathcal{Y}_t} \supseteq \mathcal{Y}_{t'}^{x} \to \mathcal{X}_{l'}^{\mathcal{Y}_{t'}} \mid \mathcal{X}_l^{\mathcal{Y}_t} \supseteq \Box\, \mathcal{X}_{l'}^{\mathcal{Y}_{t'}} \mid \\
& & \mathcal{Y}_t^{x} \supseteq \iota \mid \mathcal{Y}_t^{x} \supseteq \mathcal{Y}_{t'}^{y} \to \mathcal{X}_l^{\mathcal{Y}_{t'}} \mid \mathcal{Y}_t^{x} \supseteq \Box\, \mathcal{X}_l^{\mathcal{Y}_{t'}} \mid \\
& & \mathcal{Z}_l \supseteq true \mid \\
& & \mathcal{Y}_t^{x} \subseteq \emptyset \mid \mathcal{Y}_t^{x} \not\subseteq \emptyset \mid \mathcal{X}_l^{\mathcal{Y}_t} \subseteq \text{LAMBDA} \mid \mathcal{X}_l^{\mathcal{Y}_t} \subseteq \text{BOX}
\end{array}
$$

[그림 9] 최소 알갱이로 풀려진 집합 제약식

함수 적용 $l : (l' : e')(l'' : e'')$에서 $(l' : e')$의 타입 정보가 구체적으로 얻어지면($\mathcal{X}_{l''}^{\mathcal{Y}_{t'}} \supseteq \mathcal{Y}_{t''}^{x} \to \mathcal{X}_{l'''}^{\mathcal{Y}_{t''}}$), 타입 환경에서 x에 바인딩되는 타입 정보를 알 수 있다($\mathcal{Y}_{t''}^{x} \supseteq \mathcal{X}_{l''}^{\mathcal{Y}_{t'}}$). 함수 적용을 실행한 결과의 타입 정보는 함수 몸체의 타입 정보로부터 유도할 수 있다($\mathcal{X}_l^{\mathcal{Y}_t} \supseteq \mathcal{X}_{l'''}^{\mathcal{Y}_{t''}}$). 그리고, 함수 적용에 대해서 타입 정보를 검사할 필요가 있다면 그곳으로 흘러들어오는 함수들에 대해서도 타입 정보를 검사할 필요가 있다($\mathcal{Z}_{l'''} \supseteq \mathcal{Z}_l$).

$$
\frac{\mathcal{X}_l^{\mathcal{Y}_t} \supseteq unbox_k\, \mathcal{X}_{l'}^{\mathcal{Y}_{t'}} \qquad \mathcal{X}_{l'}^{\mathcal{Y}_{t'}} \supseteq \Box\, \mathcal{X}_{l''}^{\mathcal{Y}_{t''}}}{\{\mathcal{X}_l^{\mathcal{Y}_t} \supseteq \mathcal{X}_{l''}^{\mathcal{Y}_{t''}}, \mathcal{Z}_{l''} \supseteq \mathcal{Z}_l\}}
$$

$$
\frac{\mathcal{X}_l^{\mathcal{Y}_t} \supseteq unbox_k\, \mathcal{X}_{l'}^{\mathcal{Y}_{t'}} \qquad \mathcal{X}_{l'}^{\mathcal{Y}_{t'}} \supseteq \Box\, \mathcal{X}_{l''}^{\mathcal{Y}_{t''}} \qquad x \in FV^0(l'')}{\{\mathcal{Y}_{t''}^{x} \supseteq \mathcal{Y}_t^{x}\}}
$$

수식 $\texttt{unbox}_k\ e$의 하부 수식 e의 타입 정보가 코드 템플릿 타입을 포함하면, 전체 수식의 타입 정보도 그에 대응하는 타입을 포함한다($\mathcal{X}_l^{\mathcal{Y}_t} \supseteq \mathcal{X}_{l''}^{\mathcal{Y}_{t''}}$). 함수 적용과 유사하게, 해당 수식이 타입 검사를 할 필요가 있다면 흘러들어오는 코드 템플릿에 대해서도 타입 검사를 수행한다($\mathcal{Z}_{l''} \supseteq \mathcal{Z}_l$). 만약 흘러들어온 코드 템플릿 내에 자유 변수가 포함되어 있다면 타입 환경의 정보가 흘러가는데($\mathcal{Y}_{t''}^{x} \supseteq \mathcal{Y}_t^{x}$), 이는 타입 체계의 규칙 [TUNBOX]에서 현재 타입 환경(Γ_{n+k})이 하부 수식의 코드 템플릿의 타입 속에 포함된 조건 환경과 동일해야 한다는 규칙을 반영한다.

$$
\frac{\mathcal{X}_l^{\mathcal{Y}_t} \supseteq eval\, \mathcal{X}_{l'}^{\mathcal{Y}_t} \qquad \mathcal{X}_{l'}^{\mathcal{Y}_t} \supseteq \Box\, \mathcal{X}_{l''}^{\mathcal{Y}_{t'}}}{\{\mathcal{X}_l^{\mathcal{Y}_t} \supseteq \mathcal{X}_{l''}^{\mathcal{Y}_{t'}}, \mathcal{Z}_{l''} \supseteq \mathcal{Z}_l\}} \qquad \frac{\mathcal{X}_l^{\mathcal{Y}_t} \supseteq eval\, \mathcal{X}_{l'}^{\mathcal{Y}_t} \qquad \mathcal{X}_{l'}^{\mathcal{Y}_t} \supseteq \Box\, \mathcal{X}_{l''}^{\mathcal{Y}_{t'}} \qquad x \in FV^0(l'')}{\{\mathcal{Z}_l \Rightarrow \mathcal{Y}_{t'}^{x} \subseteq \emptyset, \mathcal{Z}_l \Rightarrow \mathcal{Y}_{t'}^{x} \not\subseteq \emptyset\}}
$$

수식 $\texttt{eval}\ e$의 경우는 수식 $\texttt{unbox}_k\ e$의 경우와 흡사하다. 단, 흘러들어온 코드 템플릿에는 자유 변수가 포함되어 있질 않아야 한다($\mathcal{Z}_l \Rightarrow \mathcal{Y}_{t''}^{x} \not\subseteq \emptyset$). 만약 자유 변수가 포함되어 있다면($\mathcal{Z}_l \Rightarrow \mathcal{Y}_{t''}^{x} \subseteq \emptyset$) 모순이 발생하여 제약식의 해가 존재하지 않는다. 즉, 해당 프로그램이 실행 중에 오류가 발생할 수 있다고 판단한다.

$$
\frac{\mathcal{Z}_l \Rightarrow \mathcal{Y}_t^{x} \subseteq \emptyset \qquad \mathcal{Z}_l \supseteq true}{\mathcal{Y}_t^{x} \subseteq \emptyset} \qquad \frac{\mathcal{Z}_l \Rightarrow \mathcal{Y}_t^{x} \not\subseteq \emptyset \qquad \mathcal{Z}_l \supseteq true}{\mathcal{Y}_t^{x} \not\subseteq \emptyset}
$$

$$
\frac{\mathcal{Z}_l \Rightarrow \mathcal{X}_e^{\mathcal{Y}_t} \subseteq \text{LAMBDA} \qquad \mathcal{Z}_l \supseteq true}{\mathcal{X}_e^{\mathcal{Y}_t} \subseteq \text{LAMBDA}} \qquad \frac{\mathcal{Z}_l \Rightarrow \mathcal{X}_e^{\mathcal{Y}_t} \subseteq \text{BOX} \qquad \mathcal{Z}_l \supseteq true}{\mathcal{X}_e^{\mathcal{Y}_t} \subseteq \text{BOX}}
$$

이 규칙들은 집합 제약식을 처음 도출할 때나 푸는 과정에서 추가된 타입 검사를 보류시킨 제약식 중에서 실제로 검사를 수행할 필요가 있다고 판단된 것들에 대해 타입 검사를 수행

하도록 변환된 규칙들을 추가한다.

$$\frac{sv_1 \supseteq sv_2 \qquad sv_2 \supseteq \iota}{sv_1 \supseteq \iota} \qquad \frac{sv_1 \supseteq sv_2 \qquad sv_2 \supseteq sv_3 \to sv_4}{sv_1 \supseteq sv_3 \to sv_4} \qquad \frac{sv_1 \supseteq sv_2 \qquad sv_2 \supseteq \Box\, sv_3}{sv_1 \supseteq \Box\, sv_3}$$

$$\frac{sv_1 \supseteq sv_2 \qquad sv_2 \supseteq true}{sv_1 \supseteq true}$$

이 규칙들은 최소 알갱이로 풀려진 제약식들의 해가 전체 집합 제약식의 해가 될 수 있도록 제약식들을 변환하는 역할을 수행한다.

이렇게 얻어진 최소 알갱이로 풀려진 제약식과 단계 0의 자유 변수에 대한 제약식($\forall x \in FV^0(l_0).\mathcal{Y}^x_{t_0} \subseteq \emptyset$)을 함께 만족하는 해가 존재하면 프로그램은 실행 중에 타입 에러가 발생하지 않는다.

4.4 예제

실제 예로서, $\lambda x.(x\,x)$와 $\mathtt{box}_t\ (c\,c)$에 대해 집합 제약식을 도출하고 푸는 과정을 살펴본다. 이 프로그램들은 올바른 프로그램임에도 불구하고 단순 타입 체계가 거절하는 프로그램들이다.

$\lambda x.(x\,x)$의 각 하부 수식에 $l_0 : \lambda x.(l_1 : (l_2 : x)\ (l_3 : x))$처럼 표지를 붙이면 프로그램의 각 부분에 대하여 다음과 같은 제약식이 도출된다.

$$\begin{array}{l} l_0 \ : \ \mathcal{X}^{\mathcal{Y}_{t_0}}_{l_0} \supseteq \mathcal{Y}^x_{t_0} \to \mathcal{X}^{\mathcal{Y}_{t_0}}_{l_1},\ \mathcal{Z}_{l_1} \supseteq \mathcal{Z}_{l_0} \\ l_1 \ : \ \mathcal{X}^{\mathcal{Y}_{t_0}}_{l_1} \supseteq apply(\mathcal{X}^{\mathcal{Y}_{t_0}}_{l_2}, \mathcal{X}^{\mathcal{Y}_{t_0}}_{l_3}),\ \mathcal{X}^{\mathcal{Y}_{t_0}}_{l_2} \subseteq \mathrm{LAMBDA},\ \mathcal{Z}_{l_1} \supseteq true \\ l_2 \ : \ \mathcal{X}^{\mathcal{Y}_{t_0}}_{l_2} \supseteq \mathcal{Y}^x_{t_0},\ \mathcal{Y}^x_{t_0} \not\subseteq \emptyset,\ \mathcal{Z}_{l_2} \supseteq true \\ l_3 \ : \ \mathcal{X}^{\mathcal{Y}_{t_0}}_{l_3} \supseteq \mathcal{Y}^x_{t_0},\ \mathcal{Y}^x_{t_0} \not\subseteq \emptyset,\ \mathcal{Z}_{l_3} \supseteq true \end{array}$$

도출된 제약식을 최대한 풀면, 최소 알갱이로 풀려진 제약식들은 다음과 같다.

$$\mathcal{X}^{\mathcal{Y}_{t_0}}_{l_0} \supseteq \mathcal{Y}^x_{t_0} \to \mathcal{X}^{\mathcal{Y}_{t_0}}_{l_1},\ \mathcal{X}^{\mathcal{Y}_{t_0}}_{l_2} \subseteq \mathrm{LAMBDA},\ \mathcal{Y}^x_{t_0} \not\subseteq \emptyset,\ \mathcal{Z}_{l_1} \supseteq true,\ \mathcal{Z}_{l_2} \supseteq true,\ \mathcal{Z}_{l_3} \supseteq true$$

타입을 제한하는 $\subseteq$, $\not\subseteq$ 형태의 제약식의 왼쪽에 등장하는 집합 변수가 $\supseteq$ 형태의 제약식의 왼쪽에 등장하지 않으므로, 이 제약식들을 만족하는 해가 존재한다.

마찬가지로, $\mathtt{box}_t\ (c\,c)$의 각 하부 수식에 $l_0 : \mathtt{box}_t\ (l_1 : (l_2 : c)\ (l_3 : c))$처럼 표지를 붙이면 프로그램에 각 부분에 대하여 아래의 제약식이 도출된다.

$$\begin{array}{l} l_0 \ : \ \mathcal{X}^{\mathcal{Y}_{t_0}}_{l_0} \supseteq \Box\, \mathcal{X}^{\mathcal{Y}_t}_{l_1},\ \mathcal{Z}_{l_1} \supseteq \mathcal{Z}_{l_0} \\ l_1 \ : \ \mathcal{X}^{\mathcal{Y}_t}_{l_1} \supseteq apply(\mathcal{X}^{\mathcal{Y}_t}_{l_2}, \mathcal{X}^{\mathcal{Y}_t}_{l_3}),\ \mathcal{Z}_{l_1} \Rightarrow \mathcal{X}^{\mathcal{Y}_t}_{l_2} \subseteq \mathrm{LAMBDA},\ \mathcal{Z}_{l_2} \supseteq \mathcal{Z}_{l_1},\ \mathcal{Z}_{l_3} \supseteq \mathcal{Z}_{l_1} \\ l_2 \ : \ \mathcal{X}^{\mathcal{Y}_t}_{l_2} \supseteq \iota \\ l_3 \ : \ \mathcal{X}^{\mathcal{Y}_t}_{l_3} \supseteq \iota \end{array}$$

이렇게 도출된 제약식을 최대한 풀면, 아래와 같은 최소 알갱이 제약식들로 풀려진다.

$$\mathcal{X}^{\mathcal{Y}_{t_0}}_{l_0} \supseteq \Box\, \mathcal{X}^{\mathcal{Y}_t}_{l_1},\ \mathcal{X}^{\mathcal{Y}_t}_{l_2} \supseteq \iota,\ \mathcal{X}^{\mathcal{Y}_t}_{l_3} \supseteq \iota$$

타입을 제한하는 $\subseteq$ 형태의 제약식이 존재하지 않으므로, 제약식들을 만족하는 해가 존재한다.

5 결론 및 향후 연구

지금까지 다단계 프로그램에 대한 흐름을 고려한 정교한 분석을 고안하였다. 이 분석은 안전할 뿐만 아니라, 단순 타입 체계보다 더 많은 올바른 프로그램을 받아들일 것이라고 추측된다. 이에 대한 형식적인(formal) 증명이 진행 중이다.

참고문헌

[1] Alan Bawden. Quasiquotation in lisp. In *Partial Evaluation and Semantic-Based Program Manipulation*, pages 4–12, 1999.

[2] Neil D. Jones. The essence of program transformation by partial evaluation and driving. In *PSI '99: Proceedings of the Third International Andrei Ershov Memorial Conference on Perspectives of System Informatics*, pages 62–79, London, UK, 2000. Springer-Verlag.

[3] Ik-Soon Kim, Kwangkeun Yi, and Cristiano Calcagno. A polymorphic modal type system for lisp-like multi-staged languages. In *POPL '06: Conference record of the 33rd ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 257–268, New York, NY, USA, 2006. ACM Press.

[4] Jens Palsberg and Michael I. Schwartzbach. Safety analysis versus type inference. *Inf. Comput.*, 118(1):128–141, 1995.

김 덕 환

– 1997-2005 서울대학교 학사
– 2005-현재 서울대학교 석사과정

<관심분야> 프로그램 분석 및 검증

이 광 근

– 1983-1987 서울대학교 학사
– 1988-1990 Univ. of Illinois at Urbana-Champaign 석사
– 1990-1993 Univ. of Illinois at Urbana-Champaign 박사
– 1993-1995 Bell Labs., Murray Hill 연구원
– 1995-2003 한국과학기술원 교수
– 2003-현재 서울대학교 교수

<관심분야> 프로그램 분석 및 검증, 프로그래밍 언어