3 Static Semantics: Type System

위의 장에서는 tcoML 프로그램이 실행된다는 것이 무엇을 뜻하는 지를 정확하게 정의하였다. 위의 규칙들을 이용해서 주어진 프로그램 e가 어떤 값 v에 대해서

$$\emptyset \vdash e \Rightarrow v$$

의 관계를 가진다는 것이 증명되면 "프로그램 e는 값 v를 계산해 낸다"고 한다. 그러한 관계가 증명되는 과정(proof tree 혹은 inference tree)은 프로그램이 실행되는 과정에 그대로 대응된다.

이 장에서는 프로그램이 실행되는 과정을 **미리** 검증해 볼 수 있는 규칙들을 생각해 본다. 이러한 규칙에 의해서 주어진 프로그램이 검증될 때에만 우리는 그 프로그램을 목적코드로 컴파일하고 싶은것이다. 컴파일러는 이제 문법이 맞는 프로그램뿐 아니라 의미있는 (문제없이 실행될 수 있는) 프로그램만 컴파일해야 하는 것이다.

이러한 규칙이 갖추어야 할 조건을 정리해보자:

- 검증된 프로그램은 실행중에 잘못되지 않는다. 즉, 임의의 프로그램 e에 대해서 검증이 성공하면 $\emptyset \vdash e \Rightarrow v$를 만족하는 v가 존재한다.
- 검증은 항상 유한시간내에 마쳐져야 한다.
- 가능한한 많은 의미있는 프로그램들이 검증될 수 있어야 한다.

이러한 규칙을 만드는 가장 훌륭한 방법이 타입이론(type theory)으로 부터 나왔다. [1]

여러분들은 본 강의를 통해서 타입이론이 프로그래밍 언어의 디자인과 구현에 어떻게 어디까지 쓰이고 있고 미래가 어떻게 될 것인지를 부분적으로나마 엿보게 될 것이다. 미리 몇 마디 덧붙이면, "첨단 프로그래밍 이론과 실제는 타입이론의 주석(footnote)에 불과한 것이고 현재의 첨단은 항상 고전을 정확하고 철저하게 이해하는 사람들에 의해서만 만들어진다는 사실을 여러분들은 깊이 이해하기 바란다."

[1] 타입이론은 1908년 B. Russel 이 집합에서의 모순(Russel's paradox)을 피하는 방안으로 고안되기 시작했으며, A. Church에 의해 1940년도에 Lambda Calculus에 적용되서 "typed lambda terms always reduce into their normal forms"라는 정리(Normalization Theorem)가 만들어 진것이 프로그래밍에 적용되는 다리가 되었다. 타입이론이 프로그래밍에 적용되는 역사를 부분적으로 일별하면 다음과 같다: simply-typed lambda calculus (Church 1940), type inference (Hindley 1969), plymorphic lambda calculus (Reynodls 1972), subtypes (Reynolds 1980), existential abstract types (Mitchell and Plotkin 1985), dependent type (MacQueen 1986), linear type (Girard 1987), SML (Milner 1990), proof-carrying code (Necula and Lee 1997), typed assembly language (Morriset 1998), etc. 각각의 수박은 수업시간에 핥는다.

우리의 tcoML에서 타입은 상당히 단순하다:

$$
\begin{array}{lll}
\multicolumn{3}{l}{\textit{Type}} \\
\tau \;\; ::= & i & \text{integer type} \\
\quad\;\; | & b & \text{boolean type} \\
\quad\;\; | & s & \text{string type} \\
\quad\;\; | & \tau \to \tau & \text{function type}
\end{array}
$$

타입규칙은 다음의 관계를 결정해주는 규칙들이다:

$$\Gamma \vdash e : \tau$$

위의 관계를 다음과 같이 읽자 "프로그램 식 e는 Γ라는 환경에서 타입 τ를 가진다." 타입 환경 Γ는 다음과 같은 테이블이다:

$$\Gamma \in \mathit{Id} \xrightarrow{\text{fin}} \mathit{Type} \qquad \text{type environment}$$

Definition 2 (Program's Type) *A program e has type τ if and only if relation $\emptyset \vdash e : \tau$ is proved. If there is no such τ, then the expression has no type.*

다음이 프로그램 식 e 의 타입을 결정하는 규칙들이다:

[Num] $$\Gamma \vdash n : i$$

[Bool] $$\Gamma \vdash \texttt{true} : \texttt{b} \qquad \Gamma \vdash \texttt{false} : \texttt{b}$$

[String] $$\Gamma \vdash \mathit{string} : s$$

[Fun] $$\frac{\Gamma[x : \tau_1] \vdash e : \tau_2}{\Gamma \vdash \texttt{fn}\ x\ \texttt{=>}\ e : \tau_1 \to \tau_2}$$

[App] $$\frac{\Gamma \vdash e_1 : \tau_1 \to \tau_2 \qquad \Gamma \vdash e_2 : \tau_1}{\Gamma \vdash e_1\, e_2 : \tau_2}$$

[Let] $$\frac{\Gamma \vdash e_1 : \tau_1 \qquad \Gamma[x : \tau_1] \vdash e_2 : \tau_2}{\Gamma \vdash \texttt{let}\ x\ \texttt{=}\ e_1\ \texttt{in}\ e_2\ \texttt{end} : \tau_2}$$

[RecLet] $$\frac{\Gamma[x : \tau_1] \vdash e_1 : \tau_1 \qquad \Gamma[x : \tau_1] \vdash e_2 : \tau_2}{\Gamma \vdash \texttt{let rec}\ x\ \texttt{=}\ e_1\ \texttt{in}\ e_2\ \texttt{end} : \tau_2}$$

[If] $$\frac{\Gamma \vdash e_1 : b \qquad \Gamma \vdash e_2 : \tau \qquad \Gamma \vdash e_3 : \tau}{\Gamma \vdash \texttt{if}\ e_1\ \texttt{then}\ e_2\ \texttt{else}\ e_3 : \tau}$$

[Op] $$\frac{\Gamma \vdash e_1 : i/b \qquad \Gamma \vdash e_2 : i/b}{\Gamma \vdash e_1\ \mathit{op}\ e_2 : i/b}$$

[Read] $$\Gamma \vdash \texttt{read} : i$$

[Write] $$\frac{\Gamma \vdash e : \tau}{\Gamma \vdash \texttt{write}\ e : \tau}$$

[Paren] $$\frac{\Gamma \vdash e : \tau}{\Gamma \vdash \texttt{(}e\texttt{)} : \tau}$$