The tcoML Language

Kwangkeun Yi
http://cs.kaist.ac.kr/~kwang
KAIST

CS420/Spring 1999

1 Syntax

The syntax of our source language tcoML is:

e	::=	*const*	constant
	|	*id*	identifier
	|	`fn` *id* `=>` *e*	function
	|	*e e*	application
	|	`let` *bind* `in` *e* `end`	local block
	|	`if` *e* `then` *e* `else` *e*	branch
	|	*e op e*	infix binary operation
	|	`read`	input
	|	`write` *e*	output
	|	`(`*e*`)`	
bind	::=	*id* `=` *e*	value binding
	|	`rec` *id* `=` *e*	recursive value binding
op	::=	`+` | `-` | `*` | `/` | `=` | `<` | `>` | `and` | `or`	
const	::=	`true` | `false` | *string* | *num*	
id			alpha-numeric identifier
string			string
num			integer

1.1 Identifiers

Alpha-numeric identifiers are [`a-zA-Z`][`a-zA-Z0-9_'`]*. Identifiers are case sensitive: `z` and `Z` are different. The reserved words cannot be used as identifiers.

```
fn let in end if then else read write rec and or true false
```

1.2 Strings/Numbers/Comments

Strings begin and end with `"`. Inside the two double quotes, any sequence of characters excepting the new-line and the `"` characters can appear: [^"^\n]*. Null string `""` is possible.

Numbers are integers, optionally prefixed with `~`(for negative integer): ~?[0-9]$^+$.

A comment is any character sequence within the comment block `(* *)`. The comment block can be nested.

1.3 Precedence/Associativity

The binary operators (*op*) precedence in decreasing order is as follows. Operators in the same set have identical precedence.

$$\{\texttt{and}, *, /\}, \{+, -, \texttt{or}\}\}, \{=, <, >\}$$

All the above operators are left associative. Function applications are left associative.

1.4 Syntactic Restrictions

Each value binding *id* `=` *e* within `rec`, *e* must be of the form `fn` *x* `=>` *e*. That is, only function values can be recursively declared.

2 Dynamic Semantics

$$
\begin{array}{rclll}
x, f & \in & Id & & \text{identifiers} \\
v & \in & Val & = \quad Num \cup String \cup Bool \cup Fvals & \text{values} \\
n & \in & Num & & \\
\langle \rho, \sigma \rangle & \in & Fvals & = \quad Fexpr \times Env & \text{function values} \\
\sigma & \in & Env & = \quad Id \overset{\text{fin}}{\to} Val & \text{environments} \\
\rho & \in & Fexpr & ::= \quad \lambda x.e & \text{function defs} \\
 & & & \;\;\mid \quad\;\; f \lambda x.e & \text{rec function defs}
\end{array}
$$

Notations: we write $\{x_1 \mapsto v_1, \cdots, x_n \mapsto v_n\}$ for an environment σ. The domain $Dom(\sigma)$ is $\{x_1, \cdots, x_n\}$. We write $\sigma(x)$ for v if $x \mapsto v \in \sigma$. If $x \mapsto v \notin \sigma$ then $\sigma(x)$ is not defined. We write $\sigma[x \mapsto v]$ for

$$\sigma \cup \{x \mapsto v\} \quad \text{if } x \notin Dom(\sigma)$$

and

$$(\sigma \setminus \{\sigma(x)\}) \cup \{x \mapsto v\} \quad \text{if } x \in Dom(\sigma).$$

The semantics rules precisely defines how relations of the form

$$\sigma \vdash e \Rightarrow v$$

to be inferred. The relation is read "expression e computes value v under environment σ."

Definition 1 (Program's Semantics) *A program e's semantics is defined to be the inference tree of relation $\emptyset \vdash e \Rightarrow v$ for some v. If there is no such v, then the expression has no meaning.*

[Const] $$\sigma \vdash c \Rightarrow c$$

[Id] $$\frac{\sigma(x) = v}{\sigma \vdash x \Rightarrow v}$$

[Fun] $$\sigma \vdash \texttt{fn}\ x\ \texttt{=>}\ e \Rightarrow \langle \lambda x.e, \sigma \rangle$$

[App] $$\frac{\sigma \vdash e_1 \Rightarrow \langle \lambda x.e, \sigma' \rangle \qquad \sigma \vdash e_2 \Rightarrow v_2 \qquad \sigma'[x \mapsto v_2] \vdash e \Rightarrow v}{\sigma \vdash e_1\, e_2 \Rightarrow v}$$

[RecApp] $$\frac{\begin{array}{c}\sigma \vdash e_1 \Rightarrow \langle f \lambda x.e, \sigma' \rangle \qquad \sigma \vdash e_2 \Rightarrow v_2 \\ \sigma'[x \mapsto v_2][f \mapsto \langle f \lambda x.e, \sigma' \rangle] \vdash e \Rightarrow v\end{array}}{\sigma \vdash e_1\, e_2 \Rightarrow v}$$

[Let] $$\frac{\sigma \vdash e_1 \Rightarrow v_1 \qquad \sigma[x \mapsto v_1] \vdash e_2 \Rightarrow v}{\sigma \vdash \texttt{let}\ x\ \texttt{=}\ e_1\ \texttt{in}\ e_2\ \texttt{end} \Rightarrow v}$$

[RecLet] $$\frac{\sigma \vdash e_1 \Rightarrow \langle \lambda x.e, \sigma' \rangle \qquad \sigma[f \mapsto \langle f \lambda x.e, \sigma' \rangle] \vdash e_2 \Rightarrow v}{\sigma \vdash \texttt{let rec}\ f\ \texttt{=}\ e_1\ \texttt{in}\ e_2\ \texttt{end} \Rightarrow v}$$

[IfT] $$\frac{\sigma \vdash e_1 \Rightarrow \texttt{true} \qquad \sigma \vdash e_2 \Rightarrow v}{\sigma \vdash \texttt{if}\ e_1\ \texttt{then}\ e_2\ \texttt{else}\ e_3 \Rightarrow v}$$

[IfT] $$\frac{\sigma \vdash e_1 \Rightarrow \texttt{false} \qquad \sigma \vdash e_3 \Rightarrow v}{\sigma \vdash \texttt{if}\ e_1\ \texttt{then}\ e_2\ \texttt{else}\ e_3 \Rightarrow v}$$

[Op] $$\frac{\sigma \vdash e_1 \Rightarrow v_1 \qquad \sigma \vdash e_2 \Rightarrow v_2 \qquad op(v_1, v_2) = v}{\sigma \vdash e_1\ op\ e_2 \Rightarrow v}$$

[Read] $$\sigma \vdash \texttt{read} \Rightarrow n$$

[Write] $$\frac{\sigma \vdash e \Rightarrow v}{\sigma \vdash \texttt{write}\ e \Rightarrow v}$$

[Paren] $$\frac{\sigma \vdash e \Rightarrow v}{\sigma \vdash \texttt{(}e\texttt{)} \Rightarrow v}$$