6.1 tySECK Syntax and Semantics

tySECK command is identical to SECK command except that first, `FTN` and `UNROLL` command has the type of its corresponding function and second, the `UNROLL` has as a sub-command the body code of the function to unroll:

$$
\begin{array}{rcl}
tyCmd & ::= & \texttt{FRAME} \mid \texttt{DEFRAME} \\
& \mid & \texttt{FETCH(}n\texttt{)} \\
& \mid & \texttt{QUOTE(}p\texttt{)} \\
& \mid & \texttt{ADD} \mid \texttt{SUB} \mid \texttt{MUL} \mid \texttt{DIV} \mid \texttt{EQ} \mid \texttt{LT} \mid \texttt{GT} \\
& \mid & \texttt{READ} \mid \texttt{WRITE} \\
& \mid & \texttt{COND(}tyCmd\texttt{,}tyCmd\texttt{)} \\
& \mid & \texttt{FTN(}\tau_1 \to \tau_2\texttt{, } tyCmd\texttt{)} \\
& \mid & \texttt{FTN(}rec\,\tau_1 \to \tau_2\texttt{,}tyCmd\texttt{)} \\
& \mid & \texttt{APP} \\
& \mid & \texttt{RETURN} \\
& \mid & \texttt{UNROLL(}rec\,\tau_1 \to \tau_2\texttt{,}tyCmd\texttt{)} \\
& \mid & tyCmd^+ \\
n & ::= & \text{non-negative integer} \\
p & ::= & \text{integer, boolean, or string}
\end{array}
$$

tySECK 코드 수행중에 만드는 값들의 타입 τ는 다음과 같다:

$$
\begin{array}{rcll}
\tau & ::= & i \mid b \mid r & \text{int, bool, string type} \\
& \mid & \tau \to \tau & \text{function type} \\
& \mid & rec\,\tau \to \tau & \text{recursive function type}
\end{array}
$$

tySECK machine's transition semantics is identical to the SECK cases except:

$$
\begin{array}{rcl}
(S,\ E,\ \texttt{FTN(}\tau_1 \to \tau_2, C'\texttt{)}.C,\ K) & \to & ((C',\ E).S,\ E,\ C,\ K) \\
(S,\ E,\ \texttt{FTN(}rec\,\tau_1 \to \tau_2, C'\texttt{)}.C,\ K) & \to & ((C',\ E).S,\ E,\ C,\ K) \\
(S,\ v.E',\ \underbrace{\texttt{UNROLL(}rec\,\tau_1 \to \tau_2\texttt{, } C'\texttt{)}.C}_{C''},\ K) & \to & (S,\ v.(C'', E').E',\ C,\ K)
\end{array}
$$

6.2 tcoML-to-tySECK translation

The tcoML-to-tySECK translation is the same as before except that first, we add types for function codes and second, the `UNROLL` has as a sub-command the body code of the function to unroll:

$$[\text{RLET}_\triangleright] \quad \frac{\begin{array}{c} L[f \mapsto \ell+1], \ell+1, e_2 \;\triangleright\; C_2 \\ L[f \mapsto \ell+1][x \mapsto \ell+2], \ell+2, e_1 \;\triangleright\; C_1 \end{array}}{\begin{array}{l} L, \ell, \texttt{let rec } f : \tau_1 \to \tau_2 \texttt{ = fn } x \texttt{ => } e_1 \texttt{ in } e_2 \texttt{ end} \\ \triangleright \\ \texttt{FTN(UNROLL(}\mathit{rec}\, \tau_1 \to \tau_2, C_1\texttt{)}.C_1\texttt{.RETURN).FRAME.}C_2\texttt{.DEFRAME} \end{array}}$$

$$[\text{FN}_\triangleright] \quad \frac{L[x \mapsto \ell+1], \ell+1, e \;\triangleright\; C}{L, \ell, \texttt{fn } x \texttt{ => } e : \tau_1 \to \tau_2 \;\triangleright\; \texttt{FTN(}\tau_1 \to \tau_2, C\texttt{.RETURN)}}$$

6.3 tySECK's Type System

각 tySECK 명령어 C 의 타입은 다음의 관계로 결정한다:

$$s, e, k \vdash C \;:\; s', e', k'$$

위의 관계는 다음과 같이 읽자: "tySECK 의 기계상태 타입 s, e, k에서 *tyCmd* C 가 수행이 마쳐지면 새로운 기계상태 타입 s', e', k'가 된다." 여기서 s, e, k 는 각각 SECK 기계의 S (stack), E (environment), K (continuation)의 타입을 나타내고 다음과 같이 정의한다:

s	$::=$	ϵ	empty stack type
	$\mid$	$\tau.s$	stack type with τ on top
	$\mid$	α	type for arbitrary/abstract stack
e	$::=$	ϵ	empty environment type
	$\mid$	$\tau.e$	environment type with τ on top
k	$::=$	ϵ	empty continuation type
	$\mid$	$(e, C).k$	continuation type with e and *tyCmd* C on top
	$\mid$	β	type for arbitrary/abstract continuation

tyCmd 의 타입을 결정하는 규칙들이 Figure 5에 정의되어 있다.

Our tySECK's type system is sound:

Theorem 9 (Soundness) *Let $s, e, k \vdash C \;:\; s', e', k'$.*
For all $S, E, K \sim s, e, k$:

- *(Progress) transitions from (S, E, C, K) are either always possible or finish with (S', E', nil, K').*
- *(Subject Reduction) if the transitions finish $(S, E, C, K) \xrightarrow{*} (S', E', nil, K')$ then $S', E', K' \sim s', e', k'$.*

Proof. We prove the subject reduction part by the structure induction on C. (That is, the induction hypothesis is that the theorem holds for subcommands of C.) The progress part can be similarly proved.

• Base case: $i.i.s, e, k \vdash \texttt{ADD} : i.s, e, k$. Let $v_1.v_2.S, E, K \sim i.i.s, e, k$. By definition,

$$(v_1.v_2.S, E, \texttt{ADD}, K) \to (v_1 + v_2.S, E, nil, K).$$

Clearly, $v_1 + v_2.S, E, K \sim i.s, e, k$. Other base cases are similarly straightforward.

• Case $b.s, e, k \vdash \texttt{COND(}C_1\texttt{, } C_2\texttt{)} : \tau.s, e, k$. Let $S, E, K \sim s, e, k$. By rule [IF] in Figure 5, $s, e, k \vdash C_1 : \tau.s, e, k$ holds. Hence by induction hypothesis, if $(S, E, C_1, K) \stackrel{*}{\to} (S', E', nil, K')$ then $S', E', K' \sim \tau.s, e, k$. Thus, the same holds if we start from $(true.S, E, \texttt{COND(}C_1\texttt{,}C_2\texttt{)}, K)$ because

$$(true.S, E, \texttt{COND(}C_1\texttt{,}C_2\texttt{)}, K) \to (S, E, C_1, K).$$

• Case $s, e, k \vdash \texttt{FN(}\tau_1 \to \tau_2\texttt{, } C\texttt{)} :$. Let $S, E, K \sim s, e, k$. By definition,

$$(S, E, \texttt{FN(}\tau_1 \to \tau_2\texttt{, } C\texttt{)}, K) \to ((C, E).S, E, nil, K).$$

We need to show that $(C, E) \sim \tau_1 \to \tau_2$. By rule [FN] in Figure 5, $C = C'\texttt{.RETURN}$ and $\alpha, \tau_1.e, \beta \vdash C' : \tau_2.\alpha, \tau_1.e, \beta$. By induction hypothesis, for any $S \sim \alpha$, $K \sim \beta$, and $v_1 \sim \tau_1$, if $(S, v_1.E, C', K) \stackrel{*}{\to} (S', E', nil, K')$ then $S' \sim \tau_2.\alpha$, $E' \sim \tau_1.e$, and $K' \sim \beta$. That is, the conditions of rule [F1] holds, hence $(C, E) \sim \tau_1 \to \tau_2$.

• Case $\tau_1.(\tau_1 \to \tau_2).s, e, k \vdash \texttt{APP} : \tau_2.s, e, k$. Let $v_1.(C, E_0).S, E, K \sim \tau_1.(\tau_1 \to \tau_2).s, e, k$. By definition

$$(v_1.(C, E_0).S, E, \texttt{APP}, K) \to (S, v_1.E_0, C, (E, nil).K).$$

By rule [F1] in Figure 6, $C = C'.\texttt{RETURN}$ and if $(S, v_1.E_0, C', (E, nil).K) \stackrel{*}{\to} (S', E', nil, K')$ then $S' \sim \tau_2.s$, $E' \sim \tau_1.e$, $K' \sim (e, nil).k$ with $S, E, K \sim s, e, k$. Thus we can let $S' = v_2.S''$ and $K' = (E'', nil).K''$ with $v_2 \sim \tau_2$, $S'' \sim s$ and $K'' \sim k$. Hence if $(S, v_1.E_0, C'.\texttt{RETURN}, (E, nil).K)$ finishes, the state right before the terminal state is

$$(v_2.S'', E', \texttt{RETURN}, (E'', nil).K''),$$

which finishes in the next state

$$(v_2.S'', E'', nil, K'').$$

By definition, $v_2.S'' \sim \tau_2.s$, $E'' \sim e$, and $K'' \sim k$.

• Other cases are similary proved. $\Box$

6.4 Type Checking

Given a tySECK code C, we need to check if s, e, k exists such that

$$\epsilon, \epsilon, \epsilon \vdash C : s, e, k.$$

The tySECK code's type-checking is a straightforward implementation of the rules in Figure 5. For example, the type-checker

```
typeVerify:  stackTy * envTy * contTy -> code -> stackTy * envTy * contTy
```

would be defined as follows:[7]

```
exception SECKtyError
fun typeVerify (CONS(tau, s),e,k) FRAME = (s, CONS(tau,e), k)
  | typeVerify (CONS(INT, CONS(INT, s)), e, k) ADD = (CONS(INT,s), e, k)
  | typeVerify (CONS(BOOL,s), e, k) COND(C1, C2) =
      let
        val (tau::s', e', k') = typeVerify (s, e, k) C1
        val (tau'::s'', e'', k'') = typeVerify (s, e, k) C2
      in
        if equivT(tau,tau') andalso
           equivS(s, s') andalso equivS(s', s'') andalso
           equivE(e, e') andalso equivE(e', e'') andalso
           equivK(k, k') andalso equivK(k', k'')
        then (CONS(tau,s), e, k)
        else raise SECKtyError
      end
  | typeVerify (s, e, k) (FN(FNty(tau1,tau2), C)) =
      let
        val C'::RETURN = C
        val (tau2'::alpha', tau1'::e', beta') =
            typeVerify (alpha, tau1::e, beta) C'
      in
        if equivT(tau1,tau1') andalso equivT(tau2,tau2') andalso
           equivT(alpha,alpha') andalso equivT(beta,beta') andalso
           equivE(e,e')
        then (FNty(tau1,tau2)::s,e,k)
        else raise SECKtyError
  |
  ...
```

[7]You can use the unification as in type-checking the tcoML expressions.

[FR] $\tau.s, e, k \vdash \texttt{FRAME} : s, \tau.e, k \qquad s, \tau.e, k \vdash \texttt{DEFRAME} : s, e, k$

[OP] $i.i.s, e, k \vdash \texttt{ADD} : i.s, e, k \qquad b.b.s, e, k \vdash \texttt{AND} : b.s, e, k$

$i.i.s, e, k \vdash \texttt{EQ} : b.s, e, k \qquad b.b.s, e, k \vdash \texttt{EQ} : b.s, e, k$

[FE] $s, e \text{ as } \tau_0.\cdots.\tau_n.e', k \vdash \texttt{FETCH}(n) : \tau_n.s, e, k$

[QU] $$\frac{\mathit{type}(c) = \tau}{s, e, k \vdash \texttt{QUOTE}(c) : \tau.s, e, k}$$

[IO] $s, e, k \vdash \texttt{READ} : i.s, e, k \qquad s, e, k \vdash \texttt{WRITE} : s, e, k$

[IF] $$\frac{s, e, k \vdash C_1 : \tau.s, e, k \qquad s, e, k \vdash C_2 : \tau.s, e, k}{b.s, e, k \vdash \texttt{COND}(C_1,\ C_2) : \tau.s, e, k}$$

[FN] $$\frac{\begin{array}{c} C = C'.\texttt{RETURN} \\ \alpha, \tau_1.e, \beta \vdash C' : \tau_2.\alpha, \tau_1.e, \beta \end{array}}{s, e, k \vdash \texttt{FN}(\tau_1 \to \tau_2,\ C) : (\tau_1 \to \tau_2).s, e, k}$$

[RF] $$\frac{\begin{array}{c} C = \texttt{UNROLL}(\mathit{rec}\,\tau_1 \to \tau_2, C').C'.\texttt{RETURN} \\ \alpha, \tau_1.\mathit{rec}\,\tau_1 \to \tau_2.e, \beta \vdash C' : \tau_2.\alpha, \tau_1.\mathit{rec}\,\tau_1 \to \tau_2.e, \beta \end{array}}{s, e, k \vdash \texttt{FN}(\mathit{rec}\,\tau_1 \to \tau_2,\ C) : (\mathit{rec}\,\tau_1 \to \tau_2).s, e, k}$$

[AP] $\tau_1.(\tau_1 \to \tau_2).s, e, k \vdash \texttt{APP} : \tau_2.s, e, k \qquad \tau_1.(\mathit{rec}\,\tau_1 \to \tau_2).s, e, k \vdash \texttt{APP} : \tau_2.s, e, k$

[RE] $s, e, e'.k \vdash \texttt{RETURN} : s, e', k$

[UN] $$\frac{\alpha, \tau_1.\mathit{rec}\,\tau_1 \to \tau_2.e, \beta \vdash C : \tau_2.\alpha, \tau_1.\mathit{rec}\,\tau_1 \to \tau_2.e, \beta}{s, \tau_1.e, k \vdash \texttt{UNROLL}(\mathit{rec}\,\tau_1 \to \tau_2,\ C) : s, \tau_1.(\mathit{rec}\,\tau_1 \to \tau_2).e, k}$$

[SE] $$\frac{s, e, k \vdash \mathit{cmd} : s', e', k' \qquad s', e', k' \vdash C : s'', e'', k''}{s, e, k \vdash \mathit{cmd}.C : s'', e'', k''}$$

[NI] $s, e, k \vdash \mathit{nil} : s, e, k$

그림 5: tySECK's Type System

[NT] $$nil \sim \epsilon$$

[VT] $$S \sim \alpha \qquad K \sim \beta$$

[ST] $$\frac{v \sim \tau \qquad S \sim s}{v.S \sim \tau.s}$$

[ET] $$\frac{v \sim \tau \qquad E \sim e}{v.E \sim \tau.e}$$

[KT] $$\frac{E \sim e \qquad K \sim k}{(E,C).K \sim (e,C).k}$$

[P1] $$n \sim i \qquad string \sim r$$

[P2] $$true \sim b \qquad false \sim b$$

[F1] $$\frac{\begin{array}{ll} & C = C'.\texttt{RETURN} \\ \forall S, K, v_1 \sim \tau : & (S, v_1.E, C', K) \text{ does not stuck and} \\ & \text{if } (S, v_1.E, C', K) \xrightarrow{*} (S', E', nil, K') \\ & \text{then } S' \sim \tau_2.s, E' \sim \tau_1.e, K' \sim k \\ & \text{where } S \sim s, E \sim e, \text{ and } K \sim k \end{array}}{(C,E) \sim \tau_1 \rightarrow \tau_2}$$

[F2] $$\frac{\begin{array}{ll} & C = \texttt{UNROLL(}rec\,\tau_1 \rightarrow \tau_2, C'\texttt{)}.C'.\texttt{RETURN} \\ (C,E) \sim rec\,\tau_1 \rightarrow \tau_2 & \\ \Rightarrow\ \forall S, K, v_1 \sim \tau : & (S, v_1.(C,E).E, C', K) \text{ does not stuck and} \\ & \text{if } (S, v_1.(C,E).E, C', K) \xrightarrow{*} (S', E', nil, K') \\ & \text{then } S' \sim \tau_2.s, E' \sim \tau_1.rec\,\tau_1 \rightarrow \tau_2.e, K' \sim k \\ & \text{where } S \sim s, E \sim e, \text{ and } K \sim k \end{array}}{(C,E) \sim rec\,\tau_1 \rightarrow \tau_2}$$

그림 6: tySECK's Value Type