Research On Program Analysis System
National Creative Research Initiative Center
ROPAS Korea Advanced Institute of Science and Technology

ROPAS MEMO
2000-6
July 13, 2000

Verification of Machine Codes Using An Effect Type System

Jaeyoun Chung, Sukyoung Ryu and Kwangkeun Yi
{jay,puppy,kwang}@ropas.kaist.ac.kr

July 13, 2000

Abstract

We devise a mechanism to verify the safety of machine codes. We design a stack based machine etySECK whose code part is annotated with types and effects. And we propose an effect type system to verify properties of etySECK programs. Our system analyzes memory effects as a property of programs and we can extend our system to analyze other effects.

1 Effect-Typed Abstract Machine : etySECK

The etySECK machine is a variant of Landin's SECD [1, 2, 3, 4] machine with type and effect annotations. The syntax and the semantics of the machine is described in Figure 1 and 2, respectively. Since the machine supports functional values, compilation from functional languages to this machine is not difficult. Throughout this paper, we use dot(.) as a list constructing operator and use a single value for a list of length one.

1.1 Syntax

A machine configuration consists of six components – a stack, a heap, an environment, a sequence of commands, a continuation and a set of effects of memory behaviors. The stack and the environment are just lists of values. Each command may consume several values from the stack and produce a result on the stack. The environment represents a lexically enclosing environment and the commands access the values in the environment. The continuation is a list of pairs of an environment and a code. Each pair of the continuation is a frozen state of the machine when the function application appears by the `app` command. The set of effects is a trace of memory behaviors which have occurred until the machine configuration reaches at the current state.

A memory region is a set of memory locations which are assumed to be allocated to the same block of the memory. Our system analyzes observable side-effects of memory behaviors – init, read, write – for regions.

The values are either primitive values, a location, or function values. A function value is a triple of its type, the function's body, and an environment for static binding. A polymorphic value is either `fn` or `rfn` and it can be instantiated to a concrete monomorphic value.

The type is either an integer type, a boolean type, a unit type, a location type, a function type, a type variable, or a polymorphic type. A location type indicates that a value of this type is a location pointer to a value in the heap. A function type is annotated with a latent effect indicating a possible set of effects during the execution of the function's body.

[0]This work is supported by Creative Research Initiatives of the Korean Ministry of Science and Technology.

Machine Configuration: (S, H, E, C, K, φ)

$$
\begin{array}{lcll}
S \in \mathit{Stack} & = & \mathit{Value}\ \text{list} & \text{stack} \\
H \in \mathit{Heap} & = & \mathit{Loc} \rightarrow \mathit{Value} & \text{heap} \\
E \in \mathit{Env} & = & \mathit{Value}\ \text{list} & \text{environment} \\
K \in \mathit{Cont} & = & (\mathit{Env} \times \mathit{Cmds})\ \text{list} & \text{continuation}
\end{array}
$$

$$
\begin{array}{lcll}
\mathit{Region} \ni \rho & ::= & r_i & \text{region constant} \\
& \mid & \rho_i & \text{region variable} \\
\mathit{Effect} \ni \varphi & ::= & \epsilon & \text{no effect} \\
& \mid & \{\mathit{init}(\rho)\} & \text{memory allocated} \\
& \mid & \{\mathit{read}(\rho)\} & \text{memory read} \\
& \mid & \{\mathit{write}(\rho)\} & \text{memory write} \\
& \mid & \varphi \cup \varphi' & \text{effect composition} \\
\mathit{Type} \ni \tau & ::= & i \mid b & \text{integer, boolean} \\
& \mid & \mathit{unit} & \text{unit type} \\
& \mid & \mathit{ref}_{\rho}\tau & \text{location type} \\
& \mid & \tau \xrightarrow{\varphi} \tau & \text{function type} \\
& \mid & \alpha & \text{type variable} \\
& \mid & \forall(\vec{\alpha}, \vec{\rho}).\tau & \text{polymorphic type} \\
\\
\mathit{Prim.Value} \ni p & ::= & n \mid \texttt{true} \mid \texttt{false} \mid () & \\
\\
\mathit{Value} \ni v & ::= & p & \text{primitive values} \\
& \mid & l_{\rho} & \text{location} \\
& \mid & \mathsf{fn}(\tau, C, E) & \text{function} \\
& \mid & \mathsf{rfn}(\tau, C, E) & \text{recursive function} \\
& \mid & \Lambda(\vec{\alpha}, \vec{\rho}).v & \text{polymorphic value} \\
\mathit{Cmd} \ni C & ::= & \mathit{nil} & \\
& \mid & \texttt{frame}.C \mid \texttt{deframe}.C & \\
& \mid & \texttt{fetch}(n).C \mid \texttt{quote}(p).C & \\
& \mid & \texttt{add}.C \mid \texttt{cond}(C_1, C_2).C & \\
& \mid & \texttt{fn}(\forall\vec{\alpha}, \vec{\rho}.C_f : \tau_1 \xrightarrow{\varphi} \tau_2).C & \\
& \mid & \texttt{rfn}(\forall\vec{\alpha}, \vec{\rho}.C_f : \tau_1 \xrightarrow{\varphi} \tau_2).C & \\
& \mid & \texttt{app}.C & \\
& \mid & \texttt{tapp}(\vec{\tau}, \vec{\rho}).C & \\
& \mid & \texttt{ref}(\rho).C \mid \texttt{get}.C \mid \texttt{set}.C &
\end{array}
$$

Figure 1: Machine Configuration of etySECK

$$(S,H,E,nil,(E',C).K,\varphi)\Rightarrow(S,H,E',C,K,\varphi)$$
$$(v.S,H,E,\texttt{frame}.C,K,\varphi)\Rightarrow(S,H,v.E,C,K,\varphi)$$
$$(S,H,v.E,\texttt{deframe}.C,K,\varphi)\Rightarrow(S,H,E,C,K,\varphi)$$
$$(S,H,v_0.\cdots.v_n.E,\texttt{fetch}(n).C,K,\varphi)\Rightarrow(v_n.S,H,v_0.\cdots.v_n.E,C,K,\varphi)$$
$$(S,H,E,\texttt{quote}(p).C,K,\varphi)\Rightarrow(p.S,H,E,C,K,\varphi)$$
$$(i_2.i_1.S,H,E,\texttt{add}.C,K,\varphi)\Rightarrow(i_1+i_2.S,H,E,C,K,\varphi)$$
$$(\texttt{true}.S,H,E,\texttt{cond}(C_1,C_2).C,K,\varphi)\Rightarrow(S,H,E,C_1.C,K,\varphi)$$
$$(\texttt{false}.S,H,E,\texttt{cond}(C_1,C_2).C,K,\varphi)\Rightarrow(S,H,E,C_2.C,K,\varphi)$$
$$(S,H,E,\texttt{fn}(\forall\vec{\alpha},\vec{\rho}.C':\tau_1\xrightarrow{\varphi_f}\tau_2).C,K,\varphi)\Rightarrow(\Lambda(\vec{\alpha},\vec{\rho}).\mathsf{fn}(\tau_1\xrightarrow{\varphi_f}\tau_2,C',E).S,H,E,C,K,\varphi)$$
$$(S,H,E,\texttt{rfn}(\forall\vec{\alpha},\vec{\rho}.C':\tau_1\xrightarrow{\varphi_f}\tau_2).C,K,\varphi)\Rightarrow(\Lambda(\vec{\alpha},\vec{\rho}).\mathsf{rfn}(\tau_1\xrightarrow{\varphi_f}\tau_2,C',E).S,H,E,C,K,\varphi)$$
$$(v.\mathsf{fn}(\tau_1\xrightarrow{\varphi_f}\tau_2,C_f,E_f).S,H,E,\mathsf{app}.C,K,\varphi)\Rightarrow(S,H,v.E_f,C_f,(E,C).K,\varphi)$$
$$(v.\mathsf{rfn}(\tau_1\xrightarrow{\varphi_f}\tau_2,C_f,E_f).S,H,E,\mathsf{app}.C,K,\varphi)\Rightarrow(S,H,v.\mathsf{rfn}(\tau_1\xrightarrow{\varphi_f}\tau_2,C_f,E_f).E_f,C_f,(E,C).K,\varphi)$$
$$(\Lambda(\vec{\alpha},\vec{\rho}).\mathsf{fn}(\tau_1\xrightarrow{\varphi_f}\tau_2,C_f,E_f).S,H,E,\texttt{tapp}(\vec{\tau},\vec{r}).C,K,\varphi)\Rightarrow$$
$$([\tau_i/\alpha_i,r_i/\rho_i]\mathsf{fn}(\tau_1\xrightarrow{\varphi_f}\tau_2,C_f,E_f).S,H,E,C,K,\varphi)$$
$$(\Lambda(\vec{\alpha},\vec{\rho}).\mathsf{rfn}(\tau_1\xrightarrow{\varphi_f}\tau_2,C_f,E_f).S,H,E,\texttt{tapp}(\vec{\tau},\vec{r}).C,K,\varphi)\Rightarrow$$
$$([\tau_i/\alpha_i,r_i/\rho_i]\mathsf{rfn}(\tau_1\xrightarrow{\varphi_f}\tau_2,C_f,E_f).S,H,E,C,K,\varphi)$$
$$(v.S,H,E,\texttt{ref}(\rho).C,K,\varphi)\Rightarrow(l_\rho.S,H[l_\rho\to v],E,C,K,\varphi\cup\{\mathit{init}(\rho)\})\quad l_\rho\notin\mathrm{Dom(H)}$$
$$(l_\rho.S,H,E,\texttt{get}.C,K,\varphi)\Rightarrow(H(l_\rho).S,H,E,C,K,\varphi\cup\{\mathit{read}(\rho)\})\quad l_\rho\in\mathrm{Dom(H)}$$
$$(v.l_\rho.S,H,E,\texttt{set}.C,K,\varphi)\Rightarrow(().S,H[l_\rho\to v],E,C,K,\varphi\cup\{\mathit{write}(\rho)\})\quad l_\rho\in\mathrm{Dom(H)}$$

Figure 2: Machine Transition Rules of etySECK

1.2 Semantics

The semantics of our language is described by the transition relations ($\Rightarrow$) in Figure 2.

If the current sequence of commands is empty (*nil*), the first element of the continuation replaces the current environment and the commands:

$$(S,H,E,nil,(E',C).K,\varphi)\Rightarrow(S,H,E',C,K,\varphi).$$

Each instruction may take some values from the top of the stack and put a result value on the stack. The `frame` moves the stack top value into the environment and the `deframe` removes the value on the top of the environment:

$$\begin{array}{rcl}(v.S,H,E,\texttt{frame}.C,K,\varphi) & \Rightarrow & (S,H,v.E,C,K,\varphi)\\ (S,H,v.E,\texttt{deframe}.C,K,\varphi) & \Rightarrow & (S,H,E,C,K,\varphi).\end{array}$$

The $\texttt{fetch}(n)$ gets the n-th value from the environment and puts the value on the stack. A new constant value p is put on the stack by the `quote`. The `add` gets two values from the stack and puts the sum on the stack. The $\texttt{cond}(C_1,C_2)$ branches to either C_1 or C_2 according to the truth value of the stack top.

Function values are produced by the `fn` or the `rfn`. Let's consider the `fn` rule:

$$(S, H, E, \texttt{fn}(\forall\vec{\alpha},\vec{\rho}.C' : \tau_1 \xrightarrow{\varphi_f} \tau_2).C, K, \varphi) \Rightarrow (\Lambda(\vec{\alpha},\vec{\rho}).\mathsf{fn}(\tau_1 \xrightarrow{\varphi_f} \tau_2, C', E).S, H, E, C, K, \varphi).$$

Note that produced values are polymorphic functions which cannot be directly used but should be instantiated to monomorphic ones by the `tapp`:

$$\begin{aligned}(\Lambda(\vec{\alpha},\vec{\rho}).\mathsf{fn}(\tau_1 \xrightarrow{\varphi_f} \tau_2, C_f, E_f).S, H, E, \texttt{tapp}(\vec{\tau},\vec{r}).C, K, \varphi) \Rightarrow \\ ([\tau_i/\alpha_i, r_i/\rho_i]\mathsf{fn}(\tau_1 \xrightarrow{\varphi_f} \tau_2, C_f, E_f).S, H, E, C, K, \varphi).\end{aligned}$$

When the `app` applies the argument on the stack top to the function on the second of the stack, the function value is already instantiated to a monomorphic one. The following rule is for applying non-recursive functions:

$$(v.\mathsf{fn}(\tau_1 \xrightarrow{\varphi_f} \tau_2, C_f, E_f).S, H, E, \mathsf{app}.C, K, \varphi) \Rightarrow (S, H, v.E_f, C_f, (E, C).K, \varphi).$$

In case of applying recursive functions, the recursive function value is also placed on the environment so that the function value is visible during the evaluation of the function's body.

Operations on the heap are `ref`, `get`, and `set`. The `ref` allocates a memory cell, stores the stack top value to it, and places the location of the cell on the stack:

$$(v.S, H, E, \texttt{ref}(\rho).C, K, \varphi) \Rightarrow (l_\rho.S, H[l_\rho \to v], E, C, K, \varphi \cup \{\mathit{init}(\rho)\}) \quad l_\rho \notin \mathrm{Dom(H)}.$$

The `get` fetches a value from the heap and the `set` updates a value in the heap. Note that each operation on the heap produces a corresponding effect.

The etySECK machine starts from the initial configuration with all components except the commands are empty. The machine stops when both the commands and the continuation are empty or there is no transition rule to apply. The former case is when the machine terminates normally and the latter case is when the machine is in a stuck configuration. We define these cases as follows:

Definition 1 (Normal Termination) *Program C terminates normally if*

$$(nil, \phi, nil, C, nil, \phi) \overset{\star}{\Rightarrow} (S, H, E, nil, nil, \varphi).$$

Definition 2 (Stuck Configuration) *The etySECK machine configuration (S, H, E, C, K, φ) ($C \neq nil$, $K \neq nil$) is in a stuck configuration if there is no $(S', H', E', C', K', \varphi')$ such that $(S, H, E, C, K, \varphi) \Rightarrow (S', H', E', C', K', \varphi')$.*

1.3 Type System

The type of a machine configuration is defined by the following judgement:

$$(S, H, E, C, K, \varphi) \rightsquigarrow (s_o, e_o, \varphi_o).$$

This judgement means that *if a given machine configuration continues its execution to a final configuration then the stack, the environment, and the effects of the final configuration have the types s_o, e_o, and the effects φ_o, respectively.* Figure 3 defines these typing rules.

In order to determine the type of a given machine configuration, we need to execute the configuration abstractly – that is, we need to execute it based on the type system. Typing

- $\boxed{(S, H, E, C, K, \varphi) \rightsquigarrow (s_o, e_o, \varphi_o)}$

$$\frac{H \models S : s_1 \quad H \models E : e_1 \quad H \models K : k_1 \quad \langle s_1, e_1, k_1, \varphi \rangle \vdash C \rightsquigarrow \langle s_o, e_o, \varphi_o \rangle}{(S, H, E, C, K, \varphi) \rightsquigarrow (s_o, e_o, \varphi_o)}$$

- $\boxed{H \models S : s, H \models E : e, H \models K : k}$

$$H \models nil : nil \qquad \frac{H \models v : \tau \quad H \models S : s}{H \models v.S : \tau.s}$$

$$\frac{H \models v : \tau \quad H \models E : e}{H \models v.E : \tau.e} \qquad \frac{H \models E : e \quad H \models K : k}{H \models (E, C).K : (e, C).k}$$

- $\boxed{H \models v : \tau}$

$$H \models p : \mathtt{type\_of}(p) \qquad \frac{l_\rho \in \mathrm{Dom}(H) \quad H \models H(l_\rho) : \tau}{H \models l_\rho : \mathit{ref}_\rho \tau}$$

$$\frac{H \models E_f : e \quad \langle \epsilon, \tau_1.e, \epsilon, \epsilon \rangle \vdash C_f \rightsquigarrow \langle \tau_2, \tau_1.e, \varphi'_f \rangle \qquad \varphi'_f \subseteq \varphi_f}{H \models \mathsf{fn}(\tau_1 \xrightarrow{\varphi_f} \tau_2, C_f, E_f) : \tau_1 \xrightarrow{\varphi_f} \tau_2}$$

$$\frac{H \models E_f : e \quad \langle \epsilon, \tau_1.\tau_1 \xrightarrow{\varphi_f} \tau_2.e, \epsilon, \epsilon \rangle \vdash C_f \rightsquigarrow \langle \tau_2, \tau_1.\tau_1 \xrightarrow{\varphi_f} \tau_2.e, \varphi'_f \rangle \qquad \varphi'_f \subseteq \varphi_f}{H \models \mathsf{rfn}(\tau_1 \xrightarrow{\varphi_f} \tau_2, C_f, E_f) : \tau_1 \xrightarrow{\varphi_f} \tau_2}$$

$$\frac{H \models \mathsf{fn}(\tau_1 \xrightarrow{\varphi_f} \tau_2, C_f, E_f) : \tau_1 \xrightarrow{\varphi_f} \tau_2 \quad H \models E_f : e \qquad \alpha_i, \rho_i \notin \mathrm{FTV}(e)}{H \models \Lambda(\vec{\alpha}, \vec{\rho}).\mathsf{fn}(\tau_1 \xrightarrow{\varphi_f} \tau_2, C_f, E_f) : \forall (\vec{\alpha}, \vec{\rho}).\tau_1 \xrightarrow{\varphi_f} \tau_2}$$

$$\frac{H \models \mathsf{rfn}(\tau_1 \xrightarrow{\varphi_f} \tau_2, C_f, E_f) : \tau_1 \xrightarrow{\varphi_f} \tau_2 \quad H \models E_f : e \qquad \alpha_i, \rho_i \notin \mathrm{FTV}(e)}{H \models \Lambda(\vec{\alpha}, \vec{\rho}).\mathsf{rfn}(\tau_1 \xrightarrow{\varphi_f} \tau_2, C_f, E_f) : \forall (\vec{\alpha}, \vec{\rho}).\tau_1 \xrightarrow{\varphi_f} \tau_2}$$

Figure 3: Typing Rules for etySECK (1)

a machine configuration consists of typing a stack, an environment, a continuation, and a sequence of commands:

$$\frac{H \models S : s_1 \quad H \models E : e_1 \quad H \models K : k_1 \quad \langle s_1, e_1, k_1, \varphi\rangle \vdash C \leadsto \langle s_o, e_o, \varphi_o\rangle}{(S, H, E, C, K, \varphi) \leadsto (s_o, e_o, \varphi_o)}.$$

The judgements $H \models S : s$, $H \models E : e$, and $H \models K : k$ mean that for a given heap H, a stack S, an environment E, and a continuation K have types s, e, and k, respectively. The type of a stack is a list of the types whose values are in the stack:

$$H \models nil : nil \qquad \frac{H \models v : \tau \quad H \models S : s}{H \models v.S : \tau.s}.$$

Types of an environment and a continuation are also lists of the types whose values are in those components.

The judgement $H \models v : \tau$ means that for a given heap H a value v has a type τ. Since the type of a location l_ρ is determined by the value stored in the heap, we need H for the value typing:

$$\frac{l_\rho \in \mathrm{Dom}(H) \quad H \models H(l_\rho) : \tau}{H \models l_\rho : \mathit{ref}_\rho \tau}.$$

A location value l_ρ has a type $\mathit{ref}_\rho\tau$ if a value stored in the location has a type τ. We assume that there is a function `type_of` which is a mapping from primitive values to their types. The type of a function value is determined by checking whether the annotated type is correct:

$$\frac{H \models E_f : e \quad \langle \epsilon, \tau_1.e, \epsilon, \epsilon\rangle \vdash C_f \leadsto \langle \tau_2, \tau_1.e, \varphi'_f\rangle \qquad \varphi'_f \subseteq \varphi_f}{H \models \mathsf{fn}(\tau_1 \xrightarrow{\varphi_f} \tau_2, C_f, E_f) : \tau_1 \xrightarrow{\varphi_f} \tau_2}.$$

Note that the actual effects of executing the function's body may be less than the annotated effects.

We define the type of a sequence of commands as follows:

$$\langle s, e, k, \varphi\rangle \vdash C \leadsto \langle s_o, e_o, \varphi_o\rangle.$$

This judgement means that *if a given machine configuration which has a stack, an environment, a continuation of types s, e, k, respectively, effects φ, and a sequence of commands C continues its execution to a final configuration, then the types of the stack, the environment, and the effects of the final configuration are s_o, e_o, φ_o, respectively.* Figure 4 defines the typing rules.

If the sequence of commands is empty, there are two possible typing rules. Rule nil describes that if both the commands and the continuation are empty, this configuration is a final state. Rule nilk states that if only the commands are empty but the continuation is not empty, the machine replaces the environment and the commands by the first element of the continuation and keeps going on:

$$\frac{\langle s, e', k, \varphi\rangle \vdash C' \leadsto \langle s_o, e_o, \varphi_o\rangle}{\langle s, e, (e', C').k, \varphi\rangle \vdash nil \leadsto \langle s_o, e_o, \varphi_o\rangle}\ \mathsf{nilk}.$$

Each typing rule for commands is an abstraction of the corresponding machine transition rule. The frame moves the stack top type into the environment and the `deframe` removes the

- $\boxed{\langle s, e, k, \varphi\rangle \vdash C \leadsto \langle s_o, e_o, \varphi_o\rangle}$

$$\frac{}{\langle s, e, \epsilon, \varphi\rangle \vdash \mathit{nil} \leadsto \langle s, e, \varphi\rangle}\ \textsf{nil} \qquad \frac{\langle s, e', k, \varphi\rangle \vdash C' \leadsto \langle s_o, e_o, \varphi_o\rangle}{\langle s, e, (e', C').k, \varphi\rangle \vdash \mathit{nil} \leadsto \langle s_o, e_o, \varphi_o\rangle}\ \textsf{nilk}$$

$$\frac{\langle s, \tau.e, k, \varphi\rangle \vdash C \leadsto \langle s_o, e_o, \varphi_o\rangle}{\langle \tau.s, e, k, \varphi\rangle \vdash \texttt{frame}.C \leadsto \langle s_o, e_o, \varphi_o\rangle}\ \textsf{frame}$$

$$\frac{\langle s, e, k, \varphi\rangle \vdash C \leadsto \langle s_o, e_o, \varphi_o\rangle}{\langle s, \tau.e, k, \varphi\rangle \vdash \texttt{deframe}.C \leadsto \langle s_o, e_o, \varphi_o\rangle}\ \textsf{deframe}$$

$$\frac{\langle \tau_n.s, e, k, \varphi\rangle \vdash C \leadsto \langle s_o, e_o, \varphi_o\rangle}{\langle s, e \text{ as } \tau_0.\cdots.\tau_n.e', k, \varphi\rangle \vdash \texttt{fetch}(n).C \leadsto \langle s_o, e_o, \varphi_o\rangle}\ \textsf{fetch}$$

$$\frac{\tau = \texttt{type\_of}(p) \qquad \langle \tau.s, e, k, \varphi\rangle \vdash C \leadsto \langle s_o, e_o, \varphi_o\rangle}{\langle s, e, k, \varphi\rangle \vdash \texttt{quote}(p).C \leadsto \langle s_o, e_o, \varphi_o\rangle}\ \textsf{quote}$$

$$\frac{\langle i.s, e, k, \varphi\rangle \vdash C \leadsto \langle s_o, e_o, \varphi_o\rangle}{\langle i.i.s, e, k, \varphi\rangle \vdash \texttt{add}.C \leadsto \langle s_o, e_o, \varphi_o\rangle}\ \textsf{add}$$

$$\frac{\begin{array}{c}\langle s, e, \epsilon, \epsilon\rangle \vdash C_i \leadsto \langle s_b, e_b, \varphi_{b_i}\rangle \quad i = 1, 2 \\ \langle s_b, e_b, k, \varphi_{b_1} \cup \varphi_{b_2} \cup \varphi\rangle \vdash C \leadsto \langle s_o, e_o, \varphi_o\rangle\end{array}}{\langle b.s, e, k, \varphi\rangle \vdash \texttt{cond}(C_1, C_2).C \leadsto \langle s_o, e_o, \varphi_o\rangle}\ \textsf{cond}$$

$$\frac{\begin{array}{c}\langle \epsilon, \tau_1.e, \epsilon, \epsilon\rangle \vdash C_f \leadsto \langle \tau_2, \tau_1.e, \varphi'_f\rangle \quad \varphi'_f \subseteq \varphi_f \qquad \alpha_i, \rho_i \notin \mathrm{FTV}(e) \\ \langle (\forall(\vec\alpha, \vec\rho).\tau_1 \xrightarrow{\varphi_f} \tau_2).s, e, k, \varphi\rangle \vdash C \leadsto \langle s_o, e_o, \varphi_o\rangle\end{array}}{\langle s, e, k, \varphi\rangle \vdash \texttt{fn}(\forall\vec\alpha, \vec\rho.C_f : \tau_1 \xrightarrow{\varphi_f} \tau_2).C \leadsto \langle s_o, e_o, \varphi_o\rangle}\ \textsf{fn}$$

$$\frac{\begin{array}{c}\langle \epsilon, \tau_1.\tau_1 \xrightarrow{\varphi_f} \tau_2.e, \epsilon, \epsilon\rangle \vdash C_f \leadsto \langle \tau_2, \tau_1.\tau_1 \xrightarrow{\varphi_f} \tau_2.e, \varphi'_f\rangle \quad \varphi'_f \subseteq \varphi_f \\ \langle (\forall(\vec\alpha, \vec\rho).\tau_1 \xrightarrow{\varphi_f} \tau_2).s, e, k, \varphi\rangle \vdash C \leadsto \langle s_o, e_o, \varphi_o\rangle \qquad \alpha_i, \rho_i \notin \mathrm{FTV}(e)\end{array}}{\langle s, e, k, \varphi\rangle \vdash \texttt{rfn}(\forall\vec\alpha, \vec\rho.C_f : \tau_1 \xrightarrow{\varphi_f} \tau_2).C \leadsto \langle s_o, e_o, \varphi_o\rangle}\ \textsf{rfn}$$

$$\frac{\langle \tau_2.s, e, k, \varphi \cup \varphi_f\rangle \vdash C \leadsto \langle s_o, e_o, \varphi_o\rangle}{\langle \tau_1.\tau_1 \xrightarrow{\varphi_f} \tau_2.s, e, k, \varphi\rangle \vdash \texttt{app}.C \leadsto \langle s_o, e_o, \varphi_o\rangle}\ \textsf{app}$$

$$\frac{\langle [\tau_i/\alpha_i, r_i/\rho_i]\tau.s, e, k, \varphi\rangle \vdash C \leadsto \langle s_o, e_o, \varphi_o\rangle}{\langle (\forall(\vec\alpha, \vec\rho).\tau).s, e, k, \varphi\rangle \vdash \texttt{tapp}(\vec\tau, \vec r).C \leadsto \langle s_o, e_o, \varphi_o\rangle}\ \textsf{tapp}$$

$$\frac{\langle \mathit{ref}_\rho \tau.s, e, k, \varphi \cup \{\mathit{init}(\rho)\}\rangle \vdash C \leadsto \langle s_o, e_o, \varphi_o\rangle}{\langle \tau.s, e, k, \varphi\rangle \vdash \texttt{ref}(\rho).C \leadsto \langle s_o, e_o, \varphi_o\rangle}\ \textsf{ref}$$

$$\frac{\langle \tau.s, e, k, \varphi \cup \{\mathit{read}(\rho)\}\rangle \vdash C \leadsto \langle s_o, e_o, \varphi_o\rangle}{\langle \mathit{ref}_\rho \tau.s, e, k, \varphi\rangle \vdash \texttt{get}.C \leadsto \langle s_o, e_o, \varphi_o\rangle}\ \textsf{get}$$

$$\frac{\langle \mathit{unit}.s, e, k, \varphi \cup \{\mathit{write}(\rho)\}\rangle \vdash C \leadsto \langle s_o, e_o, \varphi_o\rangle}{\langle \tau.\mathit{ref}_\rho \tau.s, e, k, \varphi\rangle \vdash \texttt{set}.C \leadsto \langle s_o, e_o, \varphi_o\rangle}\ \textsf{set}$$

Figure 4: Typing Rules for etySECK (2)

type on the top of the environment:

$$\frac{\langle s, \tau.e, k, \varphi\rangle \vdash C \rightsquigarrow \langle s_o, e_o, \varphi_o\rangle}{\langle \tau.s, e, k, \varphi\rangle \vdash \texttt{frame}.C \rightsquigarrow \langle s_o, e_o, \varphi_o\rangle}\ \textsf{frame}$$

$$\frac{\langle s, e, k, \varphi\rangle \vdash C \rightsquigarrow \langle s_o, e_o, \varphi_o\rangle}{\langle s, \tau.e, k, \varphi\rangle \vdash \texttt{deframe}.C \rightsquigarrow \langle s_o, e_o, \varphi_o\rangle}\ \textsf{deframe}.$$

Typing rules for functions, fn and rfn, check if the annotated types are correct and put polymorphic types on the stack. The following rule describes the typing of non-recursive functions:

$$\frac{\begin{array}{c}\langle \epsilon, \tau_1.e, \epsilon, \epsilon\rangle \vdash C_f \rightsquigarrow \langle \tau_2, \tau_1.e, \varphi'_f\rangle \quad \varphi'_f \subseteq \varphi_f \qquad \alpha_i, \rho_i \notin \mathrm{FTV}(e)\\ \langle (\forall(\vec{\alpha}, \vec{\rho}).\tau_1 \xrightarrow{\varphi_f} \tau_2).s, e, k, \varphi\rangle \vdash C \rightsquigarrow \langle s_o, e_o, \varphi_o\rangle\end{array}}{\langle s, e, k, \varphi\rangle \vdash \texttt{fn}(\forall \vec{\alpha}, \vec{\rho}.C_f : \tau_1 \xrightarrow{\varphi_f} \tau_2).C \rightsquigarrow \langle s_o, e_o, \varphi_o\rangle}\ \textsf{fn}.$$

Rule app does not distinguish applications of non-recursive functions and recursive functions:

$$\frac{\langle \tau_2.s, e, k, \varphi \cup \varphi_f\rangle \vdash C \rightsquigarrow \langle s_o, e_o, \varphi_o\rangle}{\langle \tau_1.\tau_1 \xrightarrow{\varphi_f} \tau_2.s, e, k, \varphi\rangle \vdash \texttt{app}.C \rightsquigarrow \langle s_o, e_o, \varphi_o\rangle}\ \textsf{app}.$$

Rule tapp also does not distinguish them and instantiates polymorphic types to monomorphic ones:

$$\frac{\langle [\tau_i/\alpha_i, r_i/\rho_i]\tau.s, e, k, \varphi\rangle \vdash C \rightsquigarrow \langle s_o, e_o, \varphi_o\rangle}{\langle (\forall(\vec{\alpha}, \vec{\rho}).\tau).s, e, k, \varphi\rangle \vdash \texttt{tapp}(\vec{\tau}, \vec{r}).C \rightsquigarrow \langle s_o, e_o, \varphi_o\rangle}\ \textsf{tapp}.$$

Rules `ref`, `get`, and `set` introduce corresponding memory effects. For example, rule ref introduces a $\{init(\rho)\}$ effect:

$$\frac{\langle ref_\rho \tau.s, e, k, \varphi \cup \{init(\rho)\}\rangle \vdash C \rightsquigarrow \langle s_o, e_o, \varphi_o\rangle}{\langle \tau.s, e, k, \varphi\rangle \vdash \texttt{ref}(\rho).C \rightsquigarrow \langle s_o, e_o, \varphi_o\rangle}\ \textsf{ref}.$$

2 Soundness

In this section, we prove the soundness of our typing rules by following the standard formalism of Wright and Felleisen [6]. Before presenting the main theorem, we need a few properties of our typing rules.

The first property is that the typing judgement $\langle s, e, k, \varphi\rangle \vdash C \rightsquigarrow \langle s_o, e_o, \varphi_o\rangle$ can be instantiated to a more concrete judgement by substituting the free variables occurring in the judgement. Lemma 1 formalizes this property.

Lemma 1 (Substitution) *If* $\langle s, e, k, \varphi\rangle \vdash C \rightsquigarrow \langle s_o, e_o, \varphi_o\rangle$*, then for any substitution* $\mathcal{S}$*,* $\langle \mathcal{S}s, \mathcal{S}e, \mathcal{S}k, \mathcal{S}\varphi\rangle \vdash \mathcal{S}C \rightsquigarrow \langle \mathcal{S}s_o, Se_o, S\varphi_o\rangle$.

Proof We prove by the case analysis of $\langle s, e, k, \varphi\rangle \vdash C \rightsquigarrow \langle s_o, e_o, \varphi_o\rangle$. Since most cases are obvious, we prove just two nontrivial cases and omit the rest of the proofs.

- **case** $C = \texttt{fn}(\forall \vec{\alpha}, \vec{\rho}.C_f : \tau_1 \xrightarrow{\varphi_f} \tau_2).C'$

  Suppose $\langle s, e, k, \varphi\rangle \vdash \texttt{fn}(\forall \vec{\alpha}, \vec{\rho}.C_f : \tau_1 \xrightarrow{\varphi_f} \tau_2).C' \rightsquigarrow \langle s_o, e_o, \varphi_o\rangle$. Then by fn,

$$\langle \epsilon, \tau_1.e, \epsilon, \epsilon\rangle \vdash C_f \rightsquigarrow \langle \tau_2, \tau_1.e, \varphi'_f\rangle \varphi'_f \subseteq \varphi_f \quad \alpha_i, \rho_i \notin \mathrm{FTV}(e) \tag{1}$$

$$\langle(\forall(\vec{\alpha},\vec{\rho}).\tau_1 \xrightarrow{\varphi_f} \tau_2).s, e, k, \varphi\rangle \vdash C' \leadsto \langle s_o, e_o, \varphi_o\rangle. \quad (2)$$

Let $\mathcal{S}' = \mathcal{S}[\beta_i/\alpha_i, \eta_i/\rho_i]$ for new β_i, η_i. Since $\alpha_i, \rho_i \notin \mathrm{FTV}(e)$, $\mathcal{S}'e = \mathcal{S}e$. Then by I.H., (1) implies

$$\begin{array}{c}\langle\epsilon, (\mathcal{S}'\tau_1).\mathcal{S}e, \epsilon, \epsilon\rangle \vdash \mathcal{S}'C_f \leadsto \langle\mathcal{S}'\tau_2, (\mathcal{S}'\tau_1).\mathcal{S}e, \mathcal{S}'\varphi'_f\rangle \\ \mathcal{S}'\varphi'_f \subseteq \mathcal{S}'\varphi_f \quad \beta_i, \eta_i \notin \mathrm{FTV}(\mathcal{S}e).\end{array} \quad (3)$$

By I.H. and applying $\mathcal{S}$ to (2),

$$\langle\mathcal{S}(\forall(\vec{\alpha},\vec{\rho}).\tau_1 \xrightarrow{\varphi_f} \tau_2).\mathcal{S}s, \mathcal{S}e, \mathcal{S}k, \mathcal{S}\varphi\rangle \vdash \mathcal{S}C' \leadsto \langle\mathcal{S}s_o, \mathcal{S}e_o, \mathcal{S}\varphi_o\rangle.$$

Since $\mathcal{S}(\forall(\vec{\alpha},\vec{\rho}).\tau_1 \xrightarrow{\varphi_f} \tau_2) \equiv \forall(\vec{\beta},\vec{\eta}).\mathcal{S}'\tau_1 \xrightarrow{\mathcal{S}'\varphi_f} \mathcal{S}'\tau_2$,

$$\langle\forall(\vec{\beta},\vec{\eta}).\mathcal{S}'\tau_1 \xrightarrow{\mathcal{S}'\varphi_f} \mathcal{S}'\tau_2.\mathcal{S}s, \mathcal{S}e, \mathcal{S}k, \mathcal{S}\varphi\rangle \vdash \mathcal{S}C' \leadsto \langle\mathcal{S}s_o, \mathcal{S}e_o, \mathcal{S}\varphi_o\rangle. \quad (4)$$

Now, by applying fn to (3) and (4),

$$\langle\mathcal{S}s, \mathcal{S}e, \mathcal{S}k, \mathcal{S}\varphi\rangle \vdash \mathtt{fn}(\forall\vec{\beta},\vec{\eta}.\mathcal{S}'C_f : \mathcal{S}'\tau_1 \xrightarrow{\mathcal{S}'\varphi_f} \mathcal{S}'\tau_2).\mathcal{S}C' \leadsto \langle\mathcal{S}s_o, \mathcal{S}e_o, \mathcal{S}\varphi_o\rangle$$

and this means

$$\langle\mathcal{S}s, \mathcal{S}e, \mathcal{S}k, \mathcal{S}\varphi\rangle \vdash \mathcal{S}(\mathtt{fn}(\forall\vec{\alpha},\vec{\rho}.C_f : \tau_1 \xrightarrow{\varphi_f} \tau_2).C') \leadsto \langle\mathcal{S}s_o, \mathcal{S}e_o, \mathcal{S}\varphi_o\rangle.$$

- **case** $C = \mathtt{tapp}(\vec{\tau},\vec{r}).C'$

  Suppose $\langle(\forall(\vec{\alpha},\vec{\rho}).\tau).s, e, k, \varphi\rangle \vdash \mathtt{tapp}(\vec{\tau},\vec{r}).C' \leadsto \langle s_o, e_o, \varphi_o\rangle$ then for new β_i, η_i,

$$\langle(\forall(\vec{\beta},\vec{\eta}).[\beta_i/\alpha_i, \eta_i/\rho_i]\tau).s, e, k, \varphi\rangle \vdash \mathtt{tapp}(\vec{\tau},\vec{r}).C' \leadsto \langle s_o, e_o, \varphi_o\rangle.$$

Then by tapp,

$$\langle[\tau_i/\beta_i, r_i/\eta_i]([\beta_i/\alpha_i, \eta_i/\rho_i]\tau).s, e, k, \varphi\rangle \vdash C' \leadsto \langle s_o, e_o, \varphi_o\rangle. \quad (5)$$

By I.H. and applying $\mathcal{S}$ to (5),

$$\langle\mathcal{S}([\tau_i/\beta_i, r_i/\eta_i]([\beta_i/\alpha_i, \eta_i/\rho_i]\tau).s), \mathcal{S}e, \mathcal{S}k, \mathcal{S}\varphi\rangle \vdash \mathcal{S}C' \leadsto \langle\mathcal{S}s_o, \mathcal{S}e_o, \mathcal{S}\varphi_o\rangle.$$

Since β_i, η_i's are new type variables,

$$\begin{array}{c}\mathcal{S}([\tau_i/\beta_i, r_i/\eta_i]([\beta_i/\alpha_i, \eta_i/\rho_i]\tau).s) \\ \equiv \\ [(\mathcal{S}\tau_i)/\beta_i, (\mathcal{S}r_i)/\eta_i](S[\beta_i/\alpha_i, \eta_i/\rho_i]\tau).\mathcal{S}s\ .\end{array}$$

Thus we have

$$\langle[(\mathcal{S}\tau_i)/\beta_i, (\mathcal{S}r_i)/\eta_i](S[\beta_i/\alpha_i, \eta_i/\rho_i]\tau).\mathcal{S}s, \mathcal{S}e, \mathcal{S}k, \mathcal{S}\varphi\rangle \vdash \mathcal{S}C' \leadsto \langle\mathcal{S}s_o, \mathcal{S}e_o, \mathcal{S}\varphi_o\rangle. \quad (6)$$

By applying tapp to (6),

$$\langle(\forall(\vec{\beta},\vec{\eta}).\mathcal{S}[\beta_i/\alpha_i, \eta_i/\rho_i]\tau).\mathcal{S}s, \mathcal{S}e, \mathcal{S}k, \mathcal{S}\varphi\rangle \vdash \mathtt{tapp}(\mathcal{S}\vec{\tau},\mathcal{S}\vec{r}).\mathcal{S}C' \leadsto \langle\mathcal{S}s_o, \mathcal{S}e_o, \mathcal{S}\varphi_o\rangle$$

and this implies

$$\langle(\mathcal{S}(\forall(\vec{\alpha},\vec{\rho}).\tau).s), \mathcal{S}e, \mathcal{S}k, \mathcal{S}\varphi\rangle \vdash \mathcal{S}(\mathtt{tapp}(\vec{\tau},\vec{r}).\mathcal{S}C') \leadsto \langle\mathcal{S}s_o, \mathcal{S}e_o, \mathcal{S}\varphi_o\rangle.$$

$\square$

Lemma 2 states that the quantified type variables or region variables can be instantiated to arbitrary types or regions, respectively.

Lemma 2 (Instantiation) *If* $H \models \Lambda(\vec{\alpha}, \vec{\rho}).\mathsf{fn}(\tau_1 \xrightarrow{\varphi_f} \tau_2, C_f, E_f) : \forall(\vec{\alpha}, \vec{\rho}).\tau_1 \xrightarrow{\varphi_f} \tau_2$, *then* $H \models [\tau_i/\alpha_i, r_i/\rho_i]\mathsf{fn}(\tau_1 \xrightarrow{\varphi_f} \tau_2, C_f, E_f) : [\tau_i/\alpha_i, r_i/\rho_i](\tau_1 \xrightarrow{\varphi_f} \tau_2)$.

Proof By $H \models \Lambda(\vec{\alpha}, \vec{\rho}).\mathsf{fn}(\tau_1 \xrightarrow{\varphi_f} \tau_2, C_f, E_f) : \forall(\vec{\alpha}, \vec{\rho}).\tau_1 \xrightarrow{\varphi_f} \tau_2$, we have $H \models E_f : e$ and $H \models \mathsf{fn}(\tau_1 \xrightarrow{\varphi_f} \tau_2, C_f, E_f) : \tau_1 \xrightarrow{\varphi_f} \tau_2$ $\alpha_i, \rho_i \notin \mathrm{FTV}(e)$; that is,

$$\langle \epsilon, \tau_1.e, \epsilon, \epsilon \rangle \vdash C_f \rightsquigarrow \langle \tau_2, \tau_1.e, \varphi'_f \rangle \quad \varphi'_f \subseteq \varphi_f. \tag{7}$$

Let $\mathcal{S} = [\tau_i/\alpha_i, r_i/\rho_i]$. Since $\alpha_i, \rho_i \notin \mathrm{FTV}(e)$, we have $\mathcal{S}e = e$ and $H \models \mathcal{S}E_f : e$. By (7) and Lemma 1, $\langle \epsilon, \mathcal{S}(\tau_1.e), \epsilon, \epsilon \rangle \vdash \mathcal{S}C_f \rightsquigarrow \langle \mathcal{S}\tau_2, \mathcal{S}(\tau_1.e), \mathcal{S}\varphi'_f \rangle \quad \mathcal{S}\varphi'_f \subseteq \mathcal{S}\varphi_f$.

Since $H \models \mathsf{fn}(\mathcal{S}(\tau_1 \xrightarrow{\varphi_f} \tau_2), \mathcal{S}C_f, \mathcal{S}E_f) : \mathcal{S}(\tau_1 \xrightarrow{\varphi_f} \tau_2)$, $H \models \mathcal{S}\ \mathsf{fn}(\tau_1 \xrightarrow{\varphi_f} \tau_2, C_f, E_f) : \mathcal{S}(\tau_1 \xrightarrow{\varphi_f} \tau_2)$. □

Lemma 3 states that any extension of the stack bottom does not affect the typability of a given machine configuration and Lemma 4 states the monotonicity of effects. Since the proofs of Lemma 3 and 4 are trivial we omit them.

Lemma 3 (Stack Bottom Extensibility) *If* $\langle s, e, k, \varphi \rangle \vdash C \rightsquigarrow \langle s_o, e_o, \varphi_o \rangle$, *then for any* s', $\langle s.s', e, k, \varphi \rangle \vdash C \rightsquigarrow \langle s_o.s', e_o, \varphi_o \rangle$.

Lemma 4 (Effect monotonicity)
(1) If $\langle s, e, k, \varphi \rangle \vdash C \rightsquigarrow \langle s_o, e_o, \varphi_o \rangle$, *then for any* φ', $\langle s, e, k, \varphi \cup \varphi' \rangle \vdash C \rightsquigarrow \langle s_o, e_o, \varphi_o \cup \varphi' \rangle$.
(2) If $\langle s, e, k, \varphi \rangle \vdash C \rightsquigarrow \langle s_o, e_o, \varphi_o \rangle$ *and* $\varphi' \subseteq \varphi$, *then* $\langle s, e, k, \varphi' \rangle \vdash C \rightsquigarrow \langle s_o, e_o, \varphi'_o \rangle$ *and* $\varphi'_o \subseteq \varphi_o$.

Lemma 5 describes the typing of commands under the configuration with a non-*nil* continuation: after typing the current code, our typing mechanism gets a pair of an environment and commands from the continuation and keeps typing the new commands under the new environment.

Lemma 5 (Continuation)
If $\langle s_1, e_1, \epsilon, \varphi_1 \rangle \vdash C_1 \rightsquigarrow \langle s_2, e', \varphi_2 \rangle$ *and* $\langle s_2.s_3, e_2, k_2, \varphi_2 \cup \varphi' \rangle \vdash C_2 \rightsquigarrow \langle s, e, \varphi \rangle$,
then $\langle s_1.s_3, e_1, (e_2, C_2).k_2, \varphi_1 \cup \varphi' \rangle \vdash C_1 \rightsquigarrow \langle s, e, \varphi \rangle$.

Proof We prove by induction on the length of C_1.

- **case** $C_1 = nil$

  By assumption, $s_2 = s_1, e' = e_1, \varphi_2 = \varphi_1$, and $\langle s_2.s_3, e_2, k_2, \varphi_2 \cup \varphi' \rangle \vdash C_2 \rightsquigarrow \langle s, e, \varphi \rangle$. Then by nilk,

  $$\langle s_1.s_3, e_1, (e_2, C_2).k_2, \varphi' \cup \varphi_1 \rangle \vdash nil \rightsquigarrow \langle s, e, \varphi \rangle.$$

- **case** $C_1 = \mathtt{frame}.C'$

  By assumption, $s_1 = \tau.s'_1$ for some s'_1 such that

  $$\langle \tau.s'_1, e_1, \epsilon, \varphi_1 \rangle \vdash \mathtt{frame}.C' \rightsquigarrow \langle s_2, e', \varphi_2 \rangle \tag{8}$$

  $$\langle s_2.s_3, e_2, k_2, \varphi_2 \cup \varphi' \rangle \vdash C_2 \rightsquigarrow \langle s, e, \varphi \rangle. \tag{9}$$

  Then by (8) and frame,

  $$\langle s'_1, \tau.e_1, \epsilon, \varphi_1 \rangle \vdash C' \rightsquigarrow \langle s_2, e', \varphi_2 \rangle \tag{10}$$

and by (10), (9), and I.H.,

$$\langle s_1'.s_3, \tau.e_1, (e_2, C_2).k_2, \varphi_1 \cup \varphi' \rangle \vdash C' \leadsto \langle s, e, \varphi \rangle. \tag{11}$$

Thus by (11) and frame,

$$\langle \tau.s_1'.s_3, e_1, (e_2, C_2).k_2, \varphi_1 \cup \varphi' \rangle \vdash \texttt{frame}.C' \leadsto \langle s, e, \varphi \rangle.$$

- **case** $C_1 = \texttt{deframe}.C'$
  By assumption, $e_1 = \tau.e_1'$ such that

$$\langle s_1, \tau.e_1', \epsilon, \varphi_1 \rangle \vdash \texttt{deframe}.C' \leadsto \langle s_2, e', \varphi_2 \rangle \tag{12}$$

$$\langle s_2.s_3, e_2, k_2, \varphi_2 \cup \varphi' \rangle \vdash C_2 \leadsto \langle s, e, \varphi \rangle. \tag{13}$$

Then by (12) and deframe,

$$\langle s_1, e_1', \epsilon, \varphi_1 \rangle \vdash C' \leadsto \langle s_2, e', \varphi_2 \rangle \tag{14}$$

and by (13), (14), and I.H.,

$$\langle s_1.s_3, e_1', (e_2, C_2).k_2, \varphi' \cup \varphi_1 \rangle \vdash C' \leadsto \langle s, e, \varphi \rangle. \tag{15}$$

Thus by (15) and deframe,

$$\langle s_1.s_3, \tau.e_1', (e_2, C_2).k_2, \varphi' \cup \varphi_1 \rangle \vdash \texttt{deframe}.C' \leadsto \langle s, e, \varphi \rangle.$$

- **case** $C_1 = \texttt{fetch}(n).C'$
  By assumption, $e_1 = \tau_0.\cdots.\tau_n.e_1'$ such that

$$\langle s_1, \tau_0.\cdots.\tau_n.e_1', \epsilon, \varphi_1 \rangle \vdash \texttt{fetch}(n).C' \leadsto \langle s_2, e', \varphi_2 \rangle \tag{16}$$

$$\langle s_2.s_3, e_2, k_2, \varphi_2 \cup \varphi' \rangle \vdash C_2 \leadsto \langle s, e, \varphi \rangle. \tag{17}$$

Then by (16) and fetch,

$$\langle \tau_n.s_1, \tau_0.\cdots.\tau_n.e_1', \epsilon, \varphi_1 \rangle \vdash C' \leadsto \langle s_2, e', \varphi_2 \rangle \tag{18}$$

and by (18), (17), and I.H.,

$$\langle \tau_n.s_1.s_3, \tau_0.\cdots.\tau_n.e_1', (e_2, C_2).k_2, \varphi' \cup \varphi_1 \rangle \vdash C' \leadsto \langle s, e, \varphi \rangle. \tag{19}$$

Thus by (19) and fetch,

$$\langle \tau_n.s_1.s_3, \tau_0.\cdots.\tau_n.e_1', (e_2, C_2).k_2, \varphi' \cup \varphi_1 \rangle \vdash \texttt{fetch}(n).C' \leadsto \langle s, e, \varphi \rangle.$$

- **case** $C_1 = \texttt{quote}(p).C'$
  Let $\tau = \texttt{type\_of}(p)$ then by assumption,

$$\langle s_1, e_1, \epsilon, \varphi_1 \rangle \vdash \texttt{quote}(p).C' \leadsto \langle s_2, e', \varphi_2 \rangle \tag{20}$$

$$\langle s_2.s_3, e_2, k_2, \varphi_2 \cup \varphi' \rangle \vdash C_2 \leadsto \langle s, e, \varphi \rangle. \tag{21}$$

Then by (20) and quote,

$$\langle \tau.s_1, e_1, \epsilon, \varphi_1 \rangle \vdash C' \leadsto \langle s_2, e', \varphi_2 \rangle \tag{22}$$

and by (22), (21), and I.H,

$$\langle \tau.s_1.s_3, e_1, (e_2, C_2).k_2, \varphi' \cup \varphi_1 \rangle \vdash C' \leadsto \langle s, e, \varphi \rangle. \tag{23}$$

Thus by (23) and quote,

$$\langle s_1.s_3, e_1, (e_2, C_2).k_2, \varphi' \cup \varphi_1 \rangle \vdash \texttt{quote}(p).C' \leadsto \langle s, e, \varphi \rangle.$$

- **case** $C_1 = \texttt{add}.C'$
  By assumption, $s_1 = i.i.s_1'$ such that

$$\langle i.i.s_1', e_1, \epsilon, \varphi_1 \rangle \vdash \texttt{add}.C' \leadsto \langle s_2, e', \varphi_2 \rangle \tag{24}$$

$$\langle s_2.s_3, e_2, k_2, \varphi_2 \cup \varphi' \rangle \vdash C_2 \leadsto \langle s, e, \varphi \rangle. \tag{25}$$

Then by (24) and add,

$$\langle i.s_1', e_1, \epsilon, \varphi_1 \rangle \vdash C' \leadsto \langle s_2, e', \varphi_2 \rangle \tag{26}$$

and by (26), (25), and I.H.,

$$\langle i.s_1'.s_3, e_1, (e_2, C_2).k_2, \varphi' \cup \varphi_1 \rangle \vdash C' \leadsto \langle s, e, \varphi \rangle. \tag{27}$$

Thus by (27) and add,

$$\langle i.i.s_1'.s_3, e_1, (e_2, C_2).k_2, \varphi' \cup \varphi_1 \rangle \vdash \texttt{add}.C' \leadsto \langle s, e, \varphi \rangle.$$

- **case** $C_1 = \texttt{cond}(C_t, C_f).C'$
  By assumption, $s_1 = b.s_1'$ such that

$$\langle b.s_1', e_1, \epsilon, \varphi_1 \rangle \vdash \texttt{cond}(C_t, C_f).C' \leadsto \langle s_2, e', \varphi_2 \rangle \tag{28}$$

$$\langle s_2.s_3, e_2, k_2, \varphi_2 \cup \varphi' \rangle \vdash C_2 \leadsto \langle s, e, \varphi \rangle. \tag{29}$$

Then by (28) and cond,

$$\langle s_1', e_1, \epsilon, \epsilon \rangle \vdash C_t \leadsto \langle s_b, e_b, \varphi_{b_1} \rangle \qquad \langle s_1', e_1, \epsilon, \epsilon \rangle \vdash C_f \leadsto \langle s_b, e_b, \varphi_{b_2} \rangle \tag{30}$$

$$\langle s_b, e_b, \epsilon, \varphi_{b_1} \cup \varphi_{b_2} \cup \varphi_1 \rangle \vdash C' \leadsto \langle s_2, e', \varphi_2 \rangle. \tag{31}$$

and by applying Lemma 3 to (30),

$$\langle s_1'.s_3, e_1, \epsilon, \epsilon \rangle \vdash C_t \leadsto \langle s_b.s_3, e_b, \varphi_{b_1} \rangle \qquad \langle s_1'.s_3, e_1, \epsilon, \epsilon \rangle \vdash C_f \leadsto \langle s_b.s_3, e_b, \varphi_{b_2} \rangle. \tag{32}$$

By applying I.H. to (31) and (29),

$$\langle s_b.s_3, e_b, (e_2, C_2).k_2, \varphi_{b_1} \cup \varphi_{b_2} \cup \varphi_1 \cup \varphi' \rangle \vdash C' \leadsto \langle s, e, \varphi \rangle \tag{33}$$

and by (32), (33), and cond,

$$\langle b.s_1'.s_3, e_1, (e_2, C_2).k_2, \varphi_1 \cup \varphi' \rangle \vdash \texttt{cond}(C_t, C_f).C' \leadsto \langle s, e, \varphi \rangle.$$

- **case** $C_1 = \texttt{fn}(\forall\vec{\alpha},\vec{\rho}.C_f : \tau_1\xrightarrow{\varphi_f}\tau_2).C'$

  By assumption,

$$\langle s_1, e_1, \epsilon, \varphi_1\rangle \vdash \texttt{fn}(\forall\vec{\alpha},\vec{\rho}.C_f : \tau_1\xrightarrow{\varphi_f}\tau_2).C' \rightsquigarrow \langle s_2, e', \varphi_2\rangle \tag{34}$$

$$\langle s_2.s_3, e_2, k_2, \varphi_2 \cup \varphi'\rangle \vdash C_2 \rightsquigarrow \langle s, e, \varphi\rangle. \tag{35}$$

  Then by (34) and fn,

$$\langle \epsilon, \tau_1.e_1, \epsilon, \epsilon\rangle \vdash C_f \rightsquigarrow \langle \tau_2, \tau_1.e_1, \varphi_f'\rangle \quad \varphi_f' \sqsubseteq \varphi_f \quad \alpha_i, \rho_i \notin \mathrm{FTV}(e_1) \tag{36}$$

$$\langle (\forall(\vec{\alpha},\vec{\rho}).\tau_1\xrightarrow{\varphi_f}\tau_2).s_1, e_1, \epsilon, \varphi_1\rangle \vdash C' \rightsquigarrow \langle s_2, e', \varphi_2\rangle \tag{37}$$

  and by (37), (35), and I.H.,

$$\langle (\forall(\vec{\alpha},\vec{\rho}).\tau_1\xrightarrow{\varphi_f}\tau_2).s_1.s_3, e_1, (e_2, C_2).k_2, \varphi' \cup \varphi_1\rangle \vdash C' \rightsquigarrow \langle s, e, \varphi\rangle. \tag{38}$$

  Thus by (38), (36), and fn,

$$\langle s_1.s_3, e_1, (e_2, C_2).k_2, \varphi' \cup \varphi_1\rangle \vdash \texttt{fn}(\forall\vec{\alpha},\vec{\rho}.C_f : \tau_1\xrightarrow{\varphi_f}\tau_2).C' \rightsquigarrow \langle s, e, \varphi\rangle.$$

- **case** $C_1 = \texttt{rfn}(\forall\vec{\alpha},\vec{\rho}.C_f : \tau_1\xrightarrow{\varphi_f}\tau_2).C'$

  The proof of this case is same for the case $C_1 = \texttt{fn}(\forall\vec{\alpha},\vec{\rho}.C_f : \tau_1\xrightarrow{\varphi_f}\tau_2).C'$.

- **case** $C_1 = \texttt{app}.C'$

  By assumption, $s_1 = \tau_1.\tau_1\xrightarrow{\varphi_f}\tau_2.s_1'$ such that

$$\langle \tau_1.\tau_1\xrightarrow{\varphi_f}\tau_2.s_1', e_1, \epsilon, \varphi_1\rangle \vdash \texttt{app}.C' \rightsquigarrow \langle s_2, e', \varphi_2\rangle \tag{39}$$

$$\langle s_2.s_3, e_2, k_2, \varphi_2 \cup \varphi'\rangle \vdash C_2 \rightsquigarrow \langle s, e, \varphi\rangle. \tag{40}$$

  Then by (39) and app,

$$\langle \tau_2.s_1', e_1, \epsilon, \varphi_1 \cup \varphi_f\rangle \vdash C' \rightsquigarrow \langle s_2, e', \varphi_2\rangle \tag{41}$$

  and by (40), (41), and I.H.,

$$\langle \tau_2.s_1'.s_3, e_1, (e_2, C_2).k_2, \varphi' \cup \varphi_1 \cup \varphi_f\rangle \vdash C' \rightsquigarrow \langle s, e, \varphi\rangle. \tag{42}$$

  Thus by (42) and app,

$$\langle \tau_1.\tau_1\xrightarrow{\varphi_f}\tau_2.s_1'.s_3, e_1, (e_2, C_2).k_2, \varphi' \cup \varphi_1\rangle \vdash \texttt{app}.C' \rightsquigarrow \langle s, e, \varphi\rangle.$$

- **case** $C_1 = \texttt{tapp}(\vec{\tau},\vec{r}).C'$

  By assumption, $s_1 = (\forall(\vec{\alpha},\vec{\rho}).\tau).s_1'$ such that

$$\langle (\forall(\vec{\alpha},\vec{\rho}).\tau).s_1', e_1, \epsilon, \varphi_1\rangle \vdash \texttt{tapp}(\vec{\tau},\vec{r}).C' \rightsquigarrow \langle s_2, e', \varphi_2\rangle \tag{43}$$

$$\langle s_2.s_3, e_2, k_2, \varphi_2 \cup \varphi' \rangle \vdash C_2 \leadsto \langle s, e, \varphi \rangle. \tag{44}$$

Let $\mathcal{S} = [\tau_i/\alpha_i, r_i/\rho_i]$ then by tapp,

$$\langle \mathcal{S}\tau.s_1', e_1, \epsilon, \varphi_1 \rangle \vdash C' \leadsto \langle s_2, e', \varphi_2 \rangle \tag{45}$$

and by (44), (45), and I.H.,

$$\langle \mathcal{S}\tau.s_1'.s_3, e_1, (e_2, C_2).k_2, \varphi' \cup \varphi_1 \rangle \vdash C' \leadsto \langle s, e, \varphi \rangle. \tag{46}$$

Thus by (46) and tapp,

$$\langle (\forall(\vec{\alpha}, \vec{\rho}).\tau).s_1'.s_3, e_1, (e_2, C_2).k_2, \varphi' \cup \varphi_1 \rangle \vdash \mathtt{tapp}(\vec{\tau}, \vec{r}).C' \leadsto \langle s, e, \varphi \rangle.$$

- **case** $C_1 = \mathtt{ref}(\rho).C'$
  By assumption, $s_1 = \tau.s_1'$ such that

$$\langle \tau.s_1', e_1, \epsilon, \varphi_1 \rangle \vdash \mathtt{ref}(\rho).C' \leadsto \langle s_2, e', \varphi_2 \rangle \tag{47}$$

$$\langle s_2.s_3, e_2, k_2, \varphi_2 \cup \varphi' \rangle \vdash C_2 \leadsto \langle s, e, \varphi \rangle. \tag{48}$$

Then by (47) and ref,

$$\langle \mathit{ref}_\rho \tau.s_1', e_1, \epsilon, \varphi_1 \cup \{\mathit{init}(\rho)\} \rangle \vdash C' \leadsto \langle s_2, e', \varphi_2 \rangle \tag{49}$$

and by (48), (49), and I.H.,

$$\langle \mathit{ref}_\rho \tau.s_1'.s_3, e_1, (e_2, C_2).k_2, \varphi' \cup \varphi_1 \cup \{\mathit{init}(\rho)\} \rangle \vdash C' \leadsto \langle s, e, \varphi \rangle. \tag{50}$$

Thus by (50) and ref,

$$\langle \tau.s_1'.s_3, e_1, (e_2, C_2).k_2, \varphi' \cup \varphi_1 \rangle \vdash \mathtt{ref}(\rho).C' \leadsto \langle s, e, \varphi \rangle.$$

- **case** $C_1 = \mathtt{get}.C'$
  By assumption, $s_1 = \mathit{ref}_\rho \tau.s_1'$ such that

$$\langle \mathit{ref}_\rho \tau.s_1', e_1, \epsilon, \varphi_1 \rangle \vdash \mathtt{get}.C' \leadsto \langle s_2, e', \varphi_2 \rangle \tag{51}$$

$$\langle s_2.s_3, e_2, k_2, \varphi_2 \cup \varphi' \rangle \vdash C_2 \leadsto \langle s, e, \varphi \rangle. \tag{52}$$

Then by (51) and get,

$$\langle \tau.s_1', e_1, \epsilon, \varphi_1 \cup \{\mathit{read}(\rho)\} \rangle \vdash C' \leadsto \langle s_2, e', \varphi_2 \rangle \tag{53}$$

and by (52), (53), and I.H.,

$$\langle \tau.s_1'.s_3, e_1, (e_2, C_2).k_2, \varphi' \cup \varphi_1 \cup \{\mathit{read}(\rho)\} \rangle \vdash C' \leadsto \langle s, e, \varphi \rangle. \tag{54}$$

Thus by (54) and get,

$$\langle \mathit{ref}_\rho \tau.s_1'.s_3, e_1, (e_2, C_2).k_2, \varphi' \cup \varphi_1 \rangle \vdash \mathtt{get}.C' \leadsto \langle s, e, \varphi \rangle.$$

- **case** $C_1 = \mathtt{set}.C'$

  By assumption, $s_1 = \tau.\mathit{ref}_\rho\tau.s_1'$ such that

$$\langle \tau.\mathit{ref}_\rho\tau.s_1', e_1, \epsilon, \varphi_1\rangle \vdash \mathtt{set}.C' \leadsto \langle s_2, e', \varphi_2\rangle \tag{55}$$

$$\langle s_2.s_3, e_2, k_2, \varphi_2 \cup \varphi'\rangle \vdash C_2 \leadsto \langle s, e, \varphi\rangle. \tag{56}$$

  Then by (55) and set,

$$\langle \mathit{unit}.s_1', e_1, \epsilon, \varphi_1 \cup \{\mathit{write}(\rho)\}\rangle \vdash C' \leadsto \langle s_2, e', \varphi_2\rangle \tag{57}$$

  and by (56), (57), and I.H.,

$$\langle \mathit{unit}.s_1'.s_3, e_1, (e_2, C_2).k_2, \varphi' \cup \varphi_1 \cup \{\mathit{write}(\rho)\}\rangle \vdash C' \leadsto \langle s, e, \varphi\rangle. \tag{58}$$

  Thus by (58) and set,

$$\langle \tau.\mathit{ref}_\rho\tau.s_1'.s_3, e_1, (e_2, C_2).k_2, \varphi' \cup \varphi_1\rangle \vdash \mathtt{set}.C' \leadsto \langle s, e, \varphi\rangle.$$

□

Theorem 6 states that if a machine configuration has a type, its next configuration, if any, will have the same type. Since effects annotated on types may not occur, effects can be reduced after one step transition.

Lemma 6 (Subject Reduction) *If* $(S, H, E, C, K, \varphi) \leadsto (s_o, e_o, \varphi_o)$ *and* $(S, H, E, C, K, \varphi) \Rightarrow \mathcal{M}$*, then* $\mathcal{M} \leadsto (s_o, e_o, \varphi_o')$ *and* $\varphi_o' \subseteq \varphi_o$.

Proof We prove by the case analysis of the transition relation($\Rightarrow$).

- **case** $(S, H, E, \mathit{nil}, (E', C).K, \varphi) \Rightarrow (S, H, E', C, K, \varphi)$

  By assumption, there exist s, e, k, and e' such that $H \models S : s$, $H \models E : e$, $H \models K : k$, $H \models E' : e'$, $H \models (E', C).K : (e', C).k$, and

$$\langle s, e, (e', C).k, \varphi\rangle \vdash \mathit{nil} \leadsto \langle s_o, e_o, \varphi_o\rangle.$$

  Then by nilk, $\langle s, e', k, \varphi\rangle \vdash C \leadsto \langle s_o, e_o, \varphi_o\rangle$ and $(S, H, E', C, K, \varphi) \leadsto (s_o, e_o, \varphi_o)$.

- **case** $(v.S, H, E, \mathtt{frame}.C, K, \varphi) \Rightarrow (S, H, v.E, C, K, \varphi)$

  Since by assumption $(v.S, H, E, \mathtt{frame}.C, K, \varphi) \leadsto (s_o, e_o, \varphi_o)$, there exist τ, s, e and k such that $H \models v : \tau$, $H \models S : s$, $H \models E : e$, $H \models K : k$ and

$$\langle \tau.s, e, k, \varphi\rangle \vdash \mathtt{frame}.C \leadsto \langle s_o, e_o, \varphi_o\rangle.$$

  By frame, $\langle s, \tau.e, k, \varphi\rangle \vdash C \leadsto \langle s_o, e_o, \varphi_o\rangle$. Since $H \models v.E : \tau.e$,

$$(S, H, v.E, C, K, \varphi) \leadsto (s_o, e_o, \varphi_o) \text{ and } \varphi_o \subseteq \varphi_o.$$

- **case** $(S, H, v.E, \mathtt{deframe}.C, K, \varphi) \Rightarrow (S, H, E, C, K, \varphi)$

  There exist s, e, τ, and k such that $H \models S : s$, $H \models E : e$, $H \models v : \tau$, $H \models K : k$, and $\langle s, \tau.e, k, \varphi\rangle \vdash \mathtt{deframe}.C \leadsto \langle s_o, e_o, \varphi_o\rangle$. Then by $\mathsf{deframe}$, $\langle s, e, k, \varphi\rangle \vdash C \leadsto \langle s_o, e_o, \varphi_o\rangle$, and $(S, H, E, C, K, \varphi) \leadsto (s_o, e_o, \varphi_o)$.

- **case** $(S, H, v_0.\cdots.v_n.E, \texttt{fetch}(n).C, K, \varphi) \Rightarrow (v_n.S, H, v_0.\cdots.v_n.E, C, K, \varphi)$

  There exist s, e, τ_i $(i = 0, \cdots, n)$, and k such that $H \models S : s$, $H \models E : e$, $H \models v_i : \tau_i$, $H \models K : k$ and $\langle s, \tau_0.\cdots.\tau_n.e, k, \varphi\rangle \vdash \texttt{fetch}(n).C \rightsquigarrow \langle s_o, e_o, \varphi_o\rangle$. Then by fetch, $\langle \tau_n.s, \tau_0.\cdots.\tau_n.e, k, \varphi\rangle \vdash C \rightsquigarrow \langle s_o, e_o, \varphi_o\rangle$ and

$$(v_n.S, H, E, C, K, \varphi) \rightsquigarrow (s_o, v_o, \varphi_o).$$

- **case** $(S, H, E, \texttt{quote}(p).C, K, \varphi) \Rightarrow (p.S, H, E, C, K, \varphi)$

  There exist s, e, and k such that $H \models S : s, H \models E : e, H \models K : k$ and

$$\langle s, e, k, \varphi\rangle \vdash \texttt{quote}(p).C \rightsquigarrow \langle s_o, e_o, \varphi_o\rangle.$$

  Let $\tau = \texttt{type\_of}(p)$ then by quote, $\langle \tau.s, e, k, \varphi\rangle \vdash C \rightsquigarrow \langle s_o, e_o, \varphi_o\rangle$ and

$$(p.S, H, E, C, K, \varphi) \rightsquigarrow (s_o, e_o, \varphi_o).$$

- **case** $(i_2.i_1.S, H, E, \mathsf{add}.C, K, \varphi) \Rightarrow (i_1 + i_2.S, H, E, C, K, \varphi)$

  There exist s, e, and k such that $H \models S : s$, $H \models i_j : i$ $(j = 1, 2)$, $H \models E : e$, $H \models K : k$ and $\langle i.i.s, e, k, \varphi\rangle \vdash \mathsf{add}.C \rightsquigarrow \langle s_o, e_o, \varphi_o\rangle$. Then by add, $\langle i.s, e, k, \varphi\rangle \vdash C \rightsquigarrow \langle s_o, e_o, \varphi_o\rangle$ and $H \models i_i + i_2 : i$. Thus,

$$(i_i + i_2.S, H, E, C, K, \varphi) \rightsquigarrow (s_o, e_o, \varphi_o).$$

- **case** $(\texttt{true}.S, H, E, \texttt{cond}(C_1, C_2).C, K, \varphi) \Rightarrow (S, H, E, C_1.C, K, \varphi)$

  By the same reasoning, there exist s, e, and k such that $H \models S : s$, $H \models E : e$, $H \models K : k$ and $\langle b.s, e, k, \varphi\rangle \vdash \texttt{cond}(C_1, C_2).C \rightsquigarrow \langle s_o, e_o, \varphi_o\rangle$. Then by cond,

$$\langle s, e, \epsilon, \epsilon\rangle \vdash C_1 \rightsquigarrow \langle s_b, e_b, \varphi_{b_1}\rangle \qquad \langle s, e, \epsilon, \epsilon\rangle \vdash C_2 \rightsquigarrow \langle s_b, e_b, \varphi_{b_2}\rangle \tag{59}$$

$$\langle s_b, e_b, k, \varphi_{b_1} \cup \varphi_{b_2} \cup \varphi\rangle \vdash C \rightsquigarrow \langle s_o, e_o, \varphi_o\rangle. \tag{60}$$

  By applying Lemma 4 to (60),

$$\langle s_b, e_b, k, \varphi_{b_1} \cup \varphi\rangle \vdash C \rightsquigarrow \langle s_o, e_o, \varphi'_o\rangle \quad \varphi'_o \subseteq \varphi_o \tag{61}$$

  for some φ'_o and by applying Lemma 4 to (59),

$$\langle s, e, \epsilon, \varphi\rangle \vdash C_1 \rightsquigarrow \langle s_b, e_b, \varphi_{b_1} \cup \varphi\rangle. \tag{62}$$

  Thus by (62) and (61),

$$\langle s, e, k, \varphi\rangle \vdash C_1.C \rightsquigarrow \langle s_o, e_o, \varphi'_o\rangle \text{ and } \varphi'_o \subseteq \varphi_o.$$

- **case** $(\texttt{false}.S, H, E, \texttt{cond}(C_1, C_2).C, K, \varphi) \Rightarrow (S, H, E, C_1.C, K, \varphi)$

  The proof of this case is same for the case

$$(\texttt{true}.S, H, E, \texttt{cond}(C_1, C_2).C, K, \varphi) \Rightarrow (S, H, E, C_1.C, K, \varphi).$$

- **case** $(S, H, E, \mathtt{fn}(\forall\vec{\alpha}, \vec{\rho}.C_f : \tau_1 \xrightarrow{\varphi_f} \tau_2).C, K, \varphi) \Rightarrow (\Lambda(\vec{\alpha}, \vec{\rho}).\mathsf{fn}(\tau_1 \xrightarrow{\varphi_f} \tau_2, C_f, E).S, H, E, C, K, \varphi)$

  There exist s, e, and k such that $H \models S : s$, $H \models E : e$, $H \models K : k$ and

  $$\langle s, e, k, \varphi\rangle \vdash \mathtt{fn}(\forall\vec{\alpha}, \vec{\rho}.C_f : \tau_1 \xrightarrow{\varphi_f} \tau_2).C \rightsquigarrow \langle s_o, e_o, \varphi_o\rangle.$$

  Then by fn,

  $$\langle \epsilon, \tau_1.e, \epsilon, \epsilon\rangle \vdash C_f \rightsquigarrow \langle \tau_2, \tau_1.e, \varphi'_f\rangle \quad \varphi'_f \sqsubseteq \varphi_f \quad \alpha_i, \rho_i \notin \mathrm{FTV}(e) \tag{63}$$

  $$\langle (\forall(\vec{\alpha}, \vec{\rho}).\tau_1 \xrightarrow{\varphi_f} \tau_2).s, e, k, \varphi\rangle \vdash C \rightsquigarrow \langle s_o, e_o, \varphi_o\rangle. \tag{64}$$

  By (63) and $H \models E : e$,

  $$H \models \mathsf{fn}(\tau_1 \xrightarrow{\varphi_f} \tau_2, C_f, E) : \tau_1 \xrightarrow{\varphi_f} \tau_2 \tag{65}$$

  and by (65), (63), and $H \models E : e$,

  $$H \models \Lambda(\vec{\alpha}, \vec{\rho}).\mathsf{fn}(\tau_1 \xrightarrow{\varphi_f} \tau_2, C_f, E) : \forall(\vec{\alpha}, \vec{\rho}).\tau_1 \xrightarrow{\varphi_f} \tau_2. \tag{66}$$

  Thus by (64) and (66),

  $$(\Lambda(\vec{\alpha}, \vec{\rho}).\mathsf{fn}(\tau_1 \xrightarrow{\varphi_f} \tau_2, C_f, E).S, H, E, C, K, \varphi) \rightsquigarrow (s_o, e_o, \varphi_o).$$

- **case** $(S, H, E, \mathtt{rfn}(\forall\vec{\alpha}, \vec{\rho}.C_f : \tau_1 \xrightarrow{\varphi_f} \tau_2).C, K, \varphi) \Rightarrow (\Lambda(\vec{\alpha}, \vec{\rho}).\mathsf{rfn}(\tau_1 \xrightarrow{\varphi_f} \tau_2, C_f, E).S, H, E, C, K, \varphi)$

  The proof of this case is same for the case whose code is $\mathtt{fn}(\forall\vec{\alpha}, \vec{\rho}.C_f : \tau_1 \xrightarrow{\varphi_f} \tau_2).C$.

- **case** $(v.\mathsf{fn}(\tau_1 \xrightarrow{\varphi_f} \tau_2, C_f, E_f).S, H, E, \mathsf{app}.C, K, \varphi) \Rightarrow (S, H, v.E_f, C_f, (E, C).K, \varphi)$

  There exist s, e, and k such that $H \models v : \tau_1$, $H \models \mathsf{fn}(\tau_1 \xrightarrow{\varphi_f} \tau_2, C_f, E_f) : \tau_1 \xrightarrow{\varphi_f} \tau_2$, $H \models S : s$, $H \models E : e$, $H \models K : k$ and

  $$\langle \tau_1.\tau_1 \xrightarrow{\varphi_f} \tau_2.s, e, k, \varphi\rangle \vdash \mathsf{app}.C \rightsquigarrow \langle s_o, e_o, \varphi_o\rangle. \tag{67}$$

  Since $H \models \mathsf{fn}(\tau_1 \xrightarrow{\varphi_f} \tau_2, C_f, E_f) : \tau_1 \xrightarrow{\varphi_f} \tau_2$, $H \models E_f : e_f$ and

  $$\langle \epsilon, \tau_1.e_f, \epsilon, \epsilon\rangle \vdash C_f \rightsquigarrow \langle \tau_2, \tau_1.e_f, \varphi'_f\rangle \quad \varphi'_f \subseteq \varphi_f. \tag{68}$$

  By (67) and app,

  $$\langle \tau_2.s, e, k, \varphi \cup \varphi_f\rangle \vdash C \rightsquigarrow \langle s_o, e_o, \varphi_o\rangle \tag{69}$$

  and by (69) and Lemma 4,

  $$\langle \tau_2.s, e, k, \varphi \cup \varphi'_f\rangle \vdash C \rightsquigarrow \langle s_o, e_o, \varphi'_o\rangle \quad \varphi'_o \subseteq \varphi_o \tag{70}$$

  for some φ'_o. By (68), (70), and Lemma 5,

  $$\langle s, \tau_1.e_f, (e, C).k, \varphi\rangle \vdash C_f \rightsquigarrow \langle s_o, e_o, \varphi'_o\rangle$$

  which means

  $$(S, H, v.E_f, C_f, (E, C).K, \varphi) \rightsquigarrow (s_o, e_o, \varphi'_o) \quad \varphi'_o \subseteq \varphi_o.$$

- **case** $(v.\mathsf{rfn}(\tau_1 \xrightarrow{\varphi_f} \tau_2, C_f, E_f).S, H, E, \mathsf{app}.C, K, \varphi) \Rightarrow$
  $(S, H, v.\mathsf{rfn}(\tau_1 \xrightarrow{\varphi_f} \tau_2, C_f, E_f).E_f, C_f, (E, C).K, \varphi)$
  The proof of this case is same for the case
  $$(v.\mathsf{rfn}(\tau_1 \xrightarrow{\varphi_f} \tau_2, C_f, E_f).S, H, E, \mathsf{app}.C, K, \varphi) \Rightarrow (S, H, v.E_f, C_f, (E, C).K, \varphi).$$

- **case** $(\Lambda(\vec{\alpha}, \vec{\rho}).\mathsf{fn}(\tau_1 \xrightarrow{\varphi_f} \tau_2, C_f, E_f).S, H, E, \mathtt{tapp}(\vec{\tau}, \vec{r}).C, K, \varphi) \Rightarrow$
  $([\tau_i/\alpha_i, r_i/\rho_i]\mathsf{fn}(\tau_1 \xrightarrow{\varphi_f} \tau_2, C_f, E_f).S, H, E, C, K, \varphi)$
  There exist s, e, and k such that
  $$H \models \Lambda(\vec{\alpha}, \vec{\rho}).\mathsf{fn}(\tau_1 \xrightarrow{\varphi_f} \tau_2, C_f, E_f) : \forall(\vec{\alpha}, \vec{\rho}).\tau_1 \xrightarrow{\varphi_f} \tau_2, \tag{71}$$
  $H \models S : s$, $H \models E : e$, $H \models K : k$ and
  $$\langle \forall(\vec{\alpha}, \vec{\rho}).\tau_1 \xrightarrow{\varphi_f} \tau_2.s, e, k, \varphi \rangle \vdash \mathtt{tapp}(\vec{\tau}, \vec{r}).C \rightsquigarrow \langle s_o, e_o, \varphi_o \rangle.$$
  Then by tapp,
  $$\langle [\tau_i/\alpha_i, r_i/\rho_i](\tau_1 \xrightarrow{\varphi_f} \tau_2).s, e, k, \varphi \rangle \vdash C \rightsquigarrow \langle s_o, e_o, \varphi_o \rangle. \tag{72}$$
  Let $\mathcal{S} \equiv [\tau_i/\alpha_i, r_i/\rho_i]$ then by (71) and Lemma 2,
  $$H \models \mathcal{S}\ \mathsf{fn}((\tau_1 \xrightarrow{\varphi_f} \tau_2), C_f, E_f) : \mathcal{S}(\tau_1 \xrightarrow{\varphi_f} \tau_2). \tag{73}$$
  Thus by (72) and (73),
  $$(\mathcal{S}\ \mathsf{fn}((\tau_1 \xrightarrow{\varphi_f} \tau_2), C_f, E_f).S, H, E, C, K, \varphi) \rightsquigarrow (s_o, e_o, \varphi_o).$$

- **case** $(\Lambda(\vec{\alpha}, \vec{\rho}).\mathsf{rfn}(\tau_1 \xrightarrow{\varphi_f} \tau_2, C_f, E_f).S, H, E, \mathtt{tapp}(\vec{\tau}, \vec{r}).C, K, \varphi) \Rightarrow$
  $([\tau_i/\alpha_i, r_i/\rho_i]\mathsf{rfn}(\tau_1 \xrightarrow{\varphi_f} \tau_2, C_f, E_f).S, H, E, C, K, \varphi)$
  The proof of this case is same for the case
  $(\Lambda(\vec{\alpha}, \vec{\rho}).\mathsf{fn}(\tau_1 \xrightarrow{\varphi_f} \tau_2, C_f, E_f).S, H, E, \mathtt{tapp}(\vec{\tau}, \vec{r}).C, K, \varphi) \Rightarrow$
  $([\tau_i/\alpha_i, r_i/\rho_i]\mathsf{fn}(\tau_1 \xrightarrow{\varphi_f} \tau_2, C_f, E_f).S, H, E, C, K, \varphi)$.

- **case** $(v.S, H, E, \mathtt{ref}(\rho).C, K, \varphi) \Rightarrow (l_\rho.S, H[l_\rho \to v], E, C, K, \varphi \cup \{init(\rho)\}) \quad l_\rho \notin \mathrm{Dom(H)}$
  There exist s, e, and k such that $H \models v : \tau$, $H \models S : s$, $H \models E : e$, $H \models K : k$ and $\langle \tau.s, e, k, \varphi \rangle \vdash \mathtt{ref}(\rho).C \rightsquigarrow \langle s_o, e_o, \varphi_o \rangle$. Then by ref,
  $$\langle ref_\rho \tau.s, e, k, \varphi \cup \{init(\rho)\} \rangle \vdash C \rightsquigarrow \langle s_o, e_o, \varphi_o \rangle. \tag{74}$$
  Let $H' = H[l_\rho \mapsto v]$. Since $H'(l_\rho) = v$ and $H \models v : \tau$, $H' \models v : \tau$. Therefore
  $$H' \models l_\rho : ref_\rho \tau, H' \models S : s, H' \models E : e, H' \models K : k. \tag{75}$$
  By (74) and (75),
  $$(l_\rho.S, H[l_\rho \to v], E, C, K, \varphi \cup \{init(\rho)\}) \rightsquigarrow (s_o, e_o, \varphi_o).$$

- **case** $(l_\rho.S,H,E,\texttt{get}.C,K,\varphi) \Rightarrow (H(l_\rho).S,H,E,C,K,\varphi \cup \{read(\rho)\}) \quad l_\rho \in \mathrm{Dom(H)}$

  There exist s,e, and k such that $H \models l_\rho : ref_\rho\tau$, $H \models S : s$, $H \models E : e$, $H \models K : k$ and $\langle ref_{l_\rho}\tau.s,e,k,\varphi\rangle \vdash \texttt{get}.C \rightsquigarrow \langle s_o,e_o,\varphi_o\rangle$. Then by get,

  $$\langle \tau.s,e,k,\varphi \cup \{read(\rho)\}\rangle \vdash C \rightsquigarrow \langle s_o,e_o,\varphi_o\rangle. \tag{76}$$

  Since $H \models l_\rho : ref_\rho\tau$ and $l_\rho \in \mathrm{Dom}(H)$,

  $$H \models H(l_\rho) : \tau \tag{77}$$

  and by (76) and (77),

  $$(H(l_\rho).S,H,E,C,K,\rho \cup \{read(\rho)\}) \rightsquigarrow (s_o,e_o,\varphi_o).$$

- **case** $(v.l_\rho.S,H,E,\texttt{set}.C,K,\varphi) \Rightarrow (().S,H[l_\rho \rightarrow v],E,C,K,\varphi \cup \{write(\rho)\}) \quad l_\rho \in \mathrm{Dom(H)}$

  There exist τ,s,e, and k such that $H \models v : \tau, H \models l_\rho : ref_\rho\tau$, $H \models S : s$, $H \models E : e$, $H \models K : k$ and $\langle \tau.ref_\rho\tau.s,e,k,\varphi\rangle \vdash \texttt{set}.C \rightsquigarrow \langle s_o,e_o,\varphi_o\rangle$. Then by set,

  $$\langle unit.s,e,k,\varphi \cup \{write(\rho)\}\rangle \vdash C \rightsquigarrow \langle s_o,e_o,\varphi_o\rangle. \tag{78}$$

  Let $H' = H[l_\rho \mapsto v]$. Since $H' \models l_\rho : ref_\rho\tau$,

  $$H' \models () : unit,\ H' \models S : s,\ H' \models E : e,\ H' \models K : k \tag{79}$$

  and by (78) and (79),

  $$((). S,H[l_\rho \rightarrow v],E,C,K,\varphi \cup \{write(\rho)\}) \quad \rightsquigarrow (s_o,e_o,\varphi_o).$$

□

Lemma 7 (Progress) *If* $(S,H,E,C,K,\varphi) \rightsquigarrow (s_o,e_o,\varphi_o)$ *then either* C *and* K *are nil or there exsists* $\mathcal{M}$ *such that* $(S,H,E,C,K,\varphi) \Rightarrow \mathcal{M}$.

Proof If the configuration is typable, there exist s,e and k such that $H \models S : s$, $H \models E : e$, $H \models K : k$, and $\langle s,e,k,\varphi\rangle \vdash C \rightsquigarrow \langle s_o,e_o,\varphi_o\rangle$. Since the code C is typable, there exists a rule last used during the typing of C. If the used rule is nil then C and K are *nil*. Otherwise we prove by the case analysis of the typing rules in Figure 4 that there exists a machine transition rule for each last used typing rule. Since the proof is obvious we omit the details. □

Now we prove the soundness theorem of our typing rules. Theorem 1 states that *a machine configuration which has a type would not go wrong.*

Theorem 1 (Soundness) *Let* C *be a program. If* $(nil,\phi,nil,C,nil,\phi) \rightsquigarrow (s_o,e_o,\varphi_o)$ *then either the machine transition sequence from* $(nil,\phi,nil,C,nil,\phi)$ *is infinite or it stops with a final configuration* $\mathcal{M}$ *such that* $\mathcal{M} \rightsquigarrow (s_o,e_o,\varphi'_o)$, *and* $\varphi'_o \subseteq \varphi_o$.

Proof By Lemma 6 and 7. □

When we extend our system to add new effects, the added effects should satisfy some properties to guarantee the soundness of the extended system. Since the restrictions on effects we made while proving Theorem 1 are only Lemma 4, we only need to check whether the new effects satisfy the lemma.

```
 λinitial.
    let counter = new initial in
        λinc. (counter := !counter + inc;
               !counter)

==>  compiled to etySECK

fn (∀ρ.
      fetch (0); ref ρ; frame;
      fn (fetch (1); fetch (1); get; fetch (0);
          add; set; frame; deframe; fetch (1);
          get
          : int -read(ρ),write(ρ)-> int); deframe
      :int -init(ρ)-> (int -init(ρ),write(ρ)-> int))
;;

==>  the result of type checking

: (∀ρ.int -init(ρ)-> (int -read(ρ),write(ρ)-> int), nil, nil)

==>  the result of actual run

(Λρ.fn (fetch (0); ref ρ; frame;
        fn (fetch (1); fetch (1); get; fetch (0);
            add; set; frame; deframe; fetch (1);
            get
           : int -read(ρ),write(ρ)-> int); deframe
      :int -init(ρ)-> (int -init(ρ),write(ρ)-> int)),
nil, nil)
```

Figure 5: Example program

3 Examples

Since our typing rules in Figure 3 and 4 are deterministic, implementation of a type checker is straightforward. Thus we omit a checking algorithm in this paper.

We present an example program from Talpin's paper[5] and the type checking result of the program in Figure 5. The figure shows that our type checker verifies that annotated types and effects are valid and the result of the execution actually has the same type.

In order to show what happens to the code annotated with wrong types and effects, we present a modified code in Figure 6. The figure shows that since the return type of the outer function mismatches with the type the type checker expects, the type checker reports a type error.

Compilation from mini-ML to etySECK is trivial and we omit the detailed description of it. The key idea is to instantiate each polymorphic value when it is used by inserting a `tapp` instruction. In this case, we obtain the argument of the `tapp` instruction by unifying the

```
fn (∀ρ.
      fetch (0); ref ρ; frame;
      fn (fetch (1); fetch (1); get; fetch (0);
          add; set; frame; deframe; fetch (1);
          get
          : int --read(ρ),write(ρ)--> int); deframe
      :int --init(ρ)--> (int --init(ρ)--> int))
;;

==>  the result of type checking

Type Error: Function return type mismatch
(nil,
 nil,
 nil,
 fn (∀ρ.
     fetch (0); ref ρ; frame;
     fn (fetch (1); fetch (1); get; fetch (0);
         add; set; frame; deframe; fetch (1);
         get
       : int --read(ρ),write(ρ)--> int); deframe
   :int --init(ρ)--> (int --init(ρ)--> int)),
 nil)
DECLARED : int --write(ρ)--> int,
ACTUALLY : int --read(ρ),write(ρ)--> int,
```

Figure 6: Example program *Modified*

polymorphic type and the type of the value at that point.

References

[1] P. J. Landin. The mechanical evaluation of expressions. *The Computer Journal*, 6(4):308–320, January 1964.

[2] P. J. Landin. A correspondence between ALGOL 60 and Church's lambda-notation: Part I. *Communications of the ACM*, 8(2):89–101, February 1965.

[3] P. J. Landin. A correspondence between ALGOL 60 and Church's lambda-notation: Part II. *Communications of the ACM*, 8(3):158–165, March 1965.

[4] Gordon D. Plotkin. Call-by-name, call-by-value and the λ-calculus. *Theoretical Computer Sicence*, 1:125–159, 1975.

[5] Jean-Piere Talpin and Pierre Jouvelot. Polymorphic type, region and effect inference. *Journal of Functional Programming*, 2(3):245–271, July 1992.

[6] Andrew K. Wright and Matthias Felleisen. A syntactic approach to type soundness. Technical Report TR91-160, Dept. of Computer Science, Rice University, (to appear in Information and Computation), June 1992.