Assignment 5

Anyang He (heanyang1)

The T-Call and T-Return semantics that have problem displaying in the lecture note probably look like:

$$\frac{f = \Gamma.\mathsf{funcs}[i]}{\Gamma \vdash \mathsf{call}\ i : f.\mathsf{params} \to f.\mathsf{return}}\ (\text{T-Call}) \qquad \frac{\Gamma.\mathsf{curfunc.return} = \tau_{\mathsf{ret}}}{\Gamma \vdash \mathsf{return} : \tau_1^*, \tau_{\mathsf{ret}} \to \tau_2^*}\ (\text{T-Return})$$

Problem 2

Part 1:

$$\frac{\Gamma \vdash e^*_{\mathsf{then}} : \tau_1^* \to \tau_2^* \quad \Gamma \vdash e^*_{\mathsf{else}} : \tau_1^* \to \tau_2^*}{\Gamma \vdash \mathsf{if}\ e^*_{\mathsf{then}}\ \mathsf{else}\ e^*_{\mathsf{else}} : \tau_1^*, \mathsf{i32} \to \tau_2^*}\ (\text{T-If})$$

$$\frac{\{C\ \underline{\mathsf{with}}\ \mathsf{instrs}:\ e^*_{\mathsf{then}};\ \mathsf{stack}:\ n^*\} \mapsto \{C'\ \underline{\mathsf{with}}\ \mathsf{instrs}:\ \varepsilon;\ \mathsf{stack}:\ n'^*\} \quad x \neq 0}{\{C\ \underline{\mathsf{with}}\ \mathsf{instrs}:\ \mathsf{if}\ e^*_{\mathsf{then}}\ \mathsf{else}\ e^*_{\mathsf{else}};\ \mathsf{stack}:\ n^*, x\} \mapsto \{C'\ \underline{\mathsf{with}}\ \mathsf{instrs}:\ \varepsilon;\ \mathsf{stack}:\ n'^*\}}\ (\text{D-If-True})$$

$$\frac{\{C\ \underline{\mathsf{with}}\ \mathsf{instrs}:\ e^*_{\mathsf{else}};\ \mathsf{stack}:\ n^*\} \mapsto \{C'\ \underline{\mathsf{with}}\ \mathsf{instrs}:\ \varepsilon;\ \mathsf{stack}:\ n'^*\}}{\{C\ \underline{\mathsf{with}}\ \mathsf{instrs}:\ \mathsf{if}\ e^*_{\mathsf{then}}\ \mathsf{else}\ e^*_{\mathsf{else}};\ \mathsf{stack}:\ n^*, 0\} \mapsto \{C'\ \underline{\mathsf{with}}\ \mathsf{instrs}:\ \varepsilon;\ \mathsf{stack}:\ n'^*\}}\ (\text{D-If-False})$$

Part 2:

We need to add a binary operation to define for loop:

$$\begin{aligned} \mathsf{Binop}\ \oplus ::= &\ \oplus \\ \mid &\ \mathsf{eq} \end{aligned}$$

A for loop can be decomposed to a block and a loop. Inside the for loop, br 0 means "continue", br 1 means "break", and br $x+2$ where $x \geq 0$ goes to the label x outside the for loop.

$$\frac{\begin{array}{c}\Gamma \vdash e^*_{\mathsf{init}} : \varepsilon \to \varepsilon \quad \{\Gamma\ \underline{\mathsf{with}}\ \mathsf{labels} : \Gamma.\mathsf{labels} + 2\} \vdash e^*_{\mathsf{cond}} : \varepsilon \to \mathsf{i32} \\ \{\Gamma\ \underline{\mathsf{with}}\ \mathsf{labels} : \Gamma.\mathsf{labels} + 2\} \vdash e^*_{\mathsf{post}} : \varepsilon \to \varepsilon \quad \{\Gamma\ \underline{\mathsf{with}}\ \mathsf{labels} : \Gamma.\mathsf{labels} + 2\} \vdash e^*_{\mathsf{body}} : \varepsilon \to \varepsilon\end{array}}{\Gamma \vdash \mathsf{for}\ (\mathsf{init}\ e^*_{\mathsf{init}})\ (\mathsf{cond}\ e^*_{\mathsf{cond}})\ (\mathsf{post}\ e^*_{\mathsf{post}})\ e^*_{\mathsf{body}} : \varepsilon \to \varepsilon}\ (\text{T-For})$$

$$\frac{}{\begin{array}{c}\{\mathsf{instrs}:\ \mathsf{for}\ (\mathsf{init}\ e^*_{\mathsf{init}})\ (\mathsf{cond}\ e^*_{\mathsf{cond}})\ (\mathsf{post}\ e^*_{\mathsf{post}})\ e^*_{\mathsf{body}}\} \\ \mapsto \{\mathsf{instrs}:\ e^*_{\mathsf{init}}, (\mathsf{block}\ (\mathsf{loop}\ e^*_{\mathsf{cond}}, \mathsf{i32.const0}, \mathsf{i32.eq}, \mathsf{br\_if}\ 1, e^*_{\mathsf{body}}, e^*_{\mathsf{post}}, \mathsf{br\_if}\ 0))\}\end{array}}\ (\text{D-For})$$

Part 3:

A raise should have the same type as labels (i.e. $\varepsilon \to \varepsilon$) so that it can exit labels without violating preservation, but it should consume an i32 as error number. My design choice is to add a administrative instruction that can store the error number while having type $\varepsilon \to \varepsilon$:

$$\begin{aligned} \text{Expression } e ::= \quad & e \\ \mid \quad & \mathsf{unwind}\{n\} \end{aligned}$$

where $n : \mathsf{i32}$.

$$\frac{\Gamma \vdash e^*_{\mathsf{try}} : \varepsilon \to \varepsilon \quad \Gamma \vdash e^*_{\mathsf{raise}} : \mathsf{i32} \to \varepsilon}{\Gamma \vdash \mathsf{try}\ e^*_{\mathsf{try}}\ \mathsf{catch}\ e^*_{\mathsf{raise}} : \varepsilon \to \varepsilon}\ (\text{T-Try}) \qquad \frac{}{\Gamma \vdash \mathsf{raise} : \mathsf{i32} \to \varepsilon}\ (\text{T-Raise})$$

$$\frac{}{\Gamma \vdash \mathsf{unwind}\{n\} : \varepsilon \to \varepsilon}\ (\text{T-Unwind}) \qquad \frac{}{\{\mathsf{instrs}: \mathsf{try}\ \varepsilon\ \mathsf{catch}\ e^*_{\mathsf{raise}}\} \mapsto \{\}}\ (\text{D-Try-Finish})$$

$$\frac{}{\{\mathsf{instrs}: \mathsf{try}\ \mathsf{unwind}\{n\}\ \mathsf{catch}\ e^*_{\mathsf{raise}}\} \mapsto \{\mathsf{instrs}: \mathsf{i32.const}\ n, e^*_{\mathsf{raise}}\}}\ (\text{D-Try-Catch})$$

$$\frac{}{\{\mathsf{instrs}: \mathsf{raise}; \mathsf{stack}: n\} \mapsto \{\mathsf{instrs}: \mathsf{unwind}\{n\}\}}\ (\text{D-Raise})$$

$$\frac{}{\{\mathsf{instrs}: \mathsf{label}\ \{\_\}\ (\_; \mathsf{unwind}\{n\}, \_)\} \mapsto \{\mathsf{instrs}: \mathsf{unwind}\{n\}\}}\ (\text{D-Unwind})$$

$$\frac{}{\{\mathsf{instrs}: \mathsf{unwind}\{n\}\}\ \mathsf{val}}\ (\text{D-Unwind-Val})$$

Now our $\mathsf{unwind}\{n\}$ needs to exit function frame. The problem is functions can have arbitrary return type, and the return value is not ready when a raise happens. My (inelegant) solution is to add a "dummy" value for every type and use it as the return value in such cases:

$$\begin{aligned} \text{Expression } e ::= \quad & e \\ \mid \quad & \mathsf{dummy}\{\tau\} \end{aligned}$$

Since we only have i32 type and the function can only return one value, so these should be sufficient:

$$\frac{}{\Gamma \vdash \mathsf{dummy}\{\tau\} : \tau}\ (\text{T-Dummy}) \qquad \frac{}{\{\mathsf{instrs}: \mathsf{dummy}\{\mathsf{i32}\}\} \mapsto \{\mathsf{stack}: 42\}}\ (\text{D-Dummy-I32})$$

Then we can escape functions with any types by placing a $\mathsf{dummy}\{\tau\}$ to pass preservation check (both sides have type $\varepsilon \to \tau$):

$$\frac{C.\mathsf{module.func}[i].\mathsf{return} = \tau}{\{C\ \underline{\mathsf{with}}\ \mathsf{instrs}: \mathsf{frame}\ (i, \{\mathsf{instrs}: \mathsf{unwind}\{n\}, \_\})\} \mapsto \{C\ \underline{\mathsf{with}}\ \mathsf{instrs}: \mathsf{unwind}\{n\}, \mathsf{dummy}\{\tau\}\}}\ (\text{D-Unwind-Return})$$

Remark: it doesn't matter which value we use as $\mathsf{dummy}\{\tau\}$ because it will never be used.

Problem 3

Part 1:

1. **Undefined behavior:** No. Our type system is independent on memory configuration ($C.\mathsf{mem}$ is not involved in any of the static semantics) and the only difference between C and C' is $C.\mathsf{mem} \neq C'.\mathsf{mem}$.
2. **Private function call:** No. The only way to call function 0 is to issue ($\mathsf{call}\ 0$). Since $C.\mathsf{inst}$ does not contain ($\mathsf{call}\ 0$) and $C'.\mathsf{inst} = C.\mathsf{inst}$, $C'.\mathsf{inst}$ does not contain ($\mathsf{call}\ 0$) neither.

Part 2:

1. **Undefined behavior:** The answer is the same as part 1.
2. **Private function call:** Here's an example where the attacker can use buffer overflow to call the private function 0:

```
(module
  (func $a (param $x i32) (result i32)
    (get_local $x)
    (i32.const 1)
    (i32.add))

  (func $b (param $x i32) (result i32)
    (get_local $x)
    (i32.const 2)
    (i32.add))

  (func $main (result i32)
    (i32.const 2)
    (i32.const 1)
    (i32.const 10)
    (i32.store)
    ;; use the first 40 bytes of the memory as buffer to do something
    (i32.const 10)
    (i32.load)
    (call_indirect (param i32) (result i32)))
)
```