15-411: Compiler Design — Fall 2020

Recitation 6: L3 and Dynamic Semantics Solutions — 16 October

Register Allocation in L3

As we mentioned last week, your register allocator for L3 will need to distinguish between caller- and callee-saved registers. To recap what was said in lecture, here are a few tips:

- When pre-coloring temps, you should ensure that the live ranges of the pre-colored temps are as short as possible. This is specifically relevant to the arguments and results of function calls—the best strategy is to have the arguments remain in temps until immediately before the function call, when they should be moved into argument registers.

- In order to account for caller-saved registers' values changing across function calls, you can just add the following rule to your liveness analysis:

$$\frac{l : \texttt{call}\ f \quad \text{caller-save}(r)}{\text{def}(\text{l}, \text{r})}\ J_8'$$

- When assigning registers to temps, choose caller-saved registers first. If you need to use callee-saved registers, be sure to save and restore them at the beginning and end of the function.

Review: A Dynamic Semantics for L3

A configuration of an L2 program could be modeled as one of two forms of three-tuple:

- $\eta \vdash s \blacktriangleright K$, or
- $\eta \vdash e \triangleright K$.

Here, η represents a map from variables to values, s represents the currently-executing statement, e represents the currently-evaluating expression, and K represents the continuation (what to do next with the result of evaluating the current expression or statement).

We're interested in the judgment $c \to c'$, indicating that a configuration c of the form above steps to a configuration c'. To recap, here are some of the rules defining this judgment for L2:

$$\begin{array}{rcr} \eta \vdash \text{assign}(x, e) \blacktriangleright K & \longrightarrow & \eta \vdash e \triangleright (\text{assign}(x, \_), K) \\ \eta \vdash v \triangleright (\text{assign}(x, \_), K) & \longrightarrow & \eta[x \mapsto v] \vdash \text{nop} \blacktriangleright K \\ \eta \vdash \text{nop} \blacktriangleright (s, K) & \longrightarrow & \eta \vdash s \blacktriangleright K \end{array}$$

We omit many rules—for a more complete set, refer to the Dynamic Semantics lecture or notes. In particular, not shown are the rules that indicate how to evaluate e to a value in the case of $\eta \vdash e \triangleright K$.

Let c_1 be the initial configuration, and suppose $c_i \to c_{i+1}$. If c_n is a final configuration of the form $\eta \vdash v \triangleright (\text{return}(\_), K)$, then we say that $c_1, c_2, \ldots c_n$ is the *execution trace* of c_1.

Checkpoint 0

Draw the execution trace of configurations starting from:

$$\cdot \vdash \text{seq}(\text{assign}(x, 3), \text{return}(x + 1)) \blacktriangleright \cdot$$

Solution:

$$
\begin{aligned}
\cdot \vdash \mathsf{seq}(\mathsf{assign}(x,3), \mathsf{return}(x)) \blacktriangleright \cdot \quad &\longrightarrow \quad \cdot \vdash \mathsf{assign}(x,3) \blacktriangleright (\mathsf{return}(x), \cdot) \\
&\longrightarrow \quad \cdot \vdash 3 \triangleright (\mathsf{assign}(x, \_), (\mathsf{return}(x), \cdot)) \\
&\longrightarrow \quad [x \mapsto 3] \vdash \mathsf{nop} \blacktriangleright (\mathsf{return}(x), \cdot) \\
&\longrightarrow \quad [x \mapsto 3] \vdash \mathsf{return}(x) \blacktriangleright \cdot \\
&\longrightarrow \quad [x \mapsto 3] \vdash x \triangleright (\mathsf{return}(\_), \cdot) \\
&\longrightarrow \quad [x \mapsto 3] \vdash 3 \triangleright (\mathsf{return}(\_), \cdot)
\end{aligned}
$$

L3's dynamic semantics is slightly more interesting in that returning from a function call should restore state and control to the configuration prior to the call. We amend our configuration to hold a fourth element, the call stack S, which consists of tuples of the form $\langle \eta, K \rangle$. We reproduce the rules for single-argument functions below:

$$
\begin{aligned}
S; \eta \vdash f(e) \triangleright K \quad &\longrightarrow \quad S; \eta \vdash e \triangleright (f(\_), K) \\
S; \eta \vdash v \triangleright (f(\_), K) \quad &\longrightarrow \quad (S, \langle \eta, K \rangle); [x \mapsto v] \vdash s_f \triangleright \cdot \\
& \quad \textit{supposing that } f \textit{ is defined as } f(x)\{s_f;\} \\
(S, \langle \eta, K \rangle); \eta' \vdash v \triangleright (\mathsf{return}(\_), K') \quad &\longrightarrow \quad S; \eta \vdash v \triangleright K
\end{aligned}
$$

Checkpoint 1

Draw the execution trace of the following program, starting execution at the beginning of `main`:

```
int f(int x) { return x; }
int main() { int x = 4; int y = f(3); return y; }
```

Solution: $\cdot \vdash \mathsf{seq}(\mathsf{assign}(x,4), \mathsf{seq}(\mathsf{assign}(y, \mathsf{call}(f,3)), \mathsf{return}(y))) \blacktriangleright \cdot$

$$
\begin{aligned}
&\longrightarrow \quad \cdot \vdash \mathsf{assign}(x,4) \blacktriangleright (\mathsf{seq}(\mathsf{assign}(y, \mathsf{call}(f,3)), \mathsf{return}(y)), \cdot) \\
&\longrightarrow \quad \cdot \vdash 4 \triangleright (\mathsf{assign}(x, \_), (\mathsf{seq}(\mathsf{assign}(y, \mathsf{call}(f,3)), \mathsf{return}(y)), \cdot) \\
&\longrightarrow \quad [x \mapsto 4] \vdash \mathsf{nop} \blacktriangleright (\mathsf{seq}(\mathsf{assign}(y, \mathsf{call}(f,3)), \mathsf{return}(y)), \cdot) \\
&\longrightarrow \quad [x \mapsto 4] \vdash \mathsf{seq}(\mathsf{assign}(y, \mathsf{call}(f,3)), \mathsf{return}(y)) \blacktriangleright \cdot \\
&\longrightarrow \quad [x \mapsto 4] \vdash \mathsf{assign}(y, \mathsf{call}(f,3)) \blacktriangleright (\mathsf{return}(y), \cdot) \\
&\longrightarrow \quad [x \mapsto 4] \vdash \mathsf{call}(f,3) \blacktriangleright (\mathsf{assign}(y, \_), (\mathsf{return}(y), \cdot)) \\
&\longrightarrow \quad [x \mapsto 4] \vdash 3 \triangleright (\mathsf{call}(f, \_), (\mathsf{assign}(y, \_), (\mathsf{return}(y), \cdot))) \\
&\longrightarrow \quad \langle [x \mapsto 4], (\mathsf{assign}(y, \_), (\mathsf{return}(y), \cdot)) \rangle; [x \mapsto 3] \vdash \mathsf{return}(x) \blacktriangleright \cdot \\
&\longrightarrow \quad \langle [x \mapsto 4], (\mathsf{assign}(y, \_), (\mathsf{return}(y), \cdot)) \rangle; [x \mapsto 3] \vdash x \triangleright (\mathsf{return}(\_), \cdot) \\
&\longrightarrow \quad \langle [x \mapsto 4], (\mathsf{assign}(y, \_), (\mathsf{return}(y), \cdot)) \rangle; [x \mapsto 3] \vdash 3 \triangleright (\mathsf{return}(\_), \cdot) \\
&\longrightarrow \quad [x \mapsto 4] \vdash 3 \triangleright (\mathsf{assign}(y, \_), (\mathsf{return}(y), \cdot)) \\
&\longrightarrow \quad [x \mapsto 4, y \mapsto 3] \vdash \mathsf{nop} \blacktriangleright (\mathsf{return}(y), \cdot) \\
&\longrightarrow \quad [x \mapsto 4, y \mapsto 3] \vdash \mathsf{return}(y) \blacktriangleright \cdot \\
&\longrightarrow \quad [x \mapsto 4, y \mapsto 3] \vdash y \triangleright (\mathsf{return}(\_), \cdot) \\
&\longrightarrow \quad [x \mapsto 4, y \mapsto 3] \vdash 3 \triangleright (\mathsf{return}(\_), \cdot)
\end{aligned}
$$

Checkpoint 2

Give an algorithm for determining whether an L2 program terminates. Do the same for L3. Assume unlimited stack space but 32-bit ints. Hint: what configurations are possible? How can you detect if there is a loop in an execution trace?

Solution: Finitely many configurations of an L2 program. You can write a decision procedure for the termination of an L3 program, too. (Finitely many ways to call a function.)