15-411: Compiler Design — Fall 2018

Recitation 6: Miscellaneous — 12 October

About Code Review Week

First: if you haven't already, you should sign up for a code review timeslot using the link on Piazza.

After Lab 3 is due, you'll have one extra week before the Lab 4 tests are due. You should use this time to your advantage to improve your codebase and make up for technical debt. If you used any terrible hacks to get register allocation or calling conventions to work, fix them. If you think any part of your code is a jumbled mess, refactor it. You'll want to have a solid base upon which to build Lab 4.

After you do this, push your code to the `code_review` branch. We will read your code and ask you questions about it during your team's code review. While we can give you pointers on your style and structure, what we really want to assess how well you **understand** the code you and your partner have written. We will be using your git commit log to guide our understanding of who implemented what.

If there's any significant section of your compiler that your partner implemented and you did not read, you should read it. And if you don't understand how part of your compiler works, you should ask your partner to explain it to you. That said, we don't expect you to remember every detail of your implementation—we just want to make sure that both team members are participating roughly equally and have an understanding of the compiler's structure.

Announcements

To recap, here are some important dates and deadlines in the coming weeks:

- Lab 3 is due on Tuesday (10/16).
- There is no recitation next Friday (10/19) due to mid-semester break.
- Push your code to the `code_review` branch by 9am on the following Monday (10/22).
- There is no recitation the following Friday (10/27) due to CMU's presidential inauguration.

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:

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

We omit many rules—for a more complete set, refer to Lecture 14. 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 $\eta \vdash v \triangleright (\mathsf{return}(\_), K)$, 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:

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

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:

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

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; }
```

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?