Meeting 9: Imperative Computation

Announcements

- Homework 2 due next week: Friday at 6:00pm
- Some Homework 0 feedback in GitHub
- Upcoming with Sean:
  - Thu 11:45 to 12 Feedback sessions ("interview light"). Schedule 5 minutes to discuss your homework feedback via moodle. Bring your homework (either ready on your laptop or print out) and a question.
  - Fri 10 to 11: Group tutoring session ("recitation light"). Come ask questions about the prior homework, ask to see steps worked out in detail.
  - Tue 11:45 to 12 Individual tutoring hours (office hours).

Questions

- Some remaining questions from Homework 1
  - Walk through Chapter 3
  - Contextual dynamics (with proof of 5.4)
  - Equational dynamics
Assignment #2:  
Language Design and Implementation

CSCI 5535 / ECEN 5533: Fundamentals of Programming Languages  
Spring 2018: Due Friday, February 23, 2018

The tasks in this homework ask you to formalize and prove meta-theoretical properties of an imperative core language IMP based on your experience with E. This homework also asks you to implement an extension of E in OCaml to gain experience translating formalization to implementation.

1 Language Design: IMP

In this section, we will formalize a variant of IMP from Chapter 2 of FSPL based on our experience from Homework Assignment 1. Consider the following syntax chart for IMP:

Typ \( \tau \) ::= num num numbers  
bool bool booleans  
Exp \( e \) ::= addr[a] a addresses (or “assignables”)  
num[n] n numeral  
bool[b] b boolean  
plus(e_1; e_2) \( e_1 + e_2 \) addition  
times(e_1; e_2) \( e_1 \ast e_2 \) multiplication  
eq(e_1; e_2) \( e_1 == e_2 \) equal  
le(e_1; e_2) \( e_1 \leq e_2 \) less-than-or-equal  
not(e_1) \( !e_1 \) negation  
and(e_1; e_2) \( e_1 \&\& e_2 \) conjunction  
or(e_1; e_2) \( e_1 || e_2 \) disjunction  

Cmd \( c \) ::= set[a](e) \( a := e \) assignment  
skip skip skip  
seq(c_1; c_2) \( c_1; c_2 \) sequencing  
if(e; c_1; c_2) if \( e \) then \( c_1 \) else \( c_2 \) conditional  
while(e; c_1) while \( e \) do \( c_1 \) looping  

Addr \( a \) 

Addresses \( a \) represent static memory store locations and are drawn from some unbounded set Addr. For simplicity, we fix all memory locations to only store numbers (as in FSPL). A store \( \sigma \) is thus a mapping from addresses to numbers, written as follows:

\[ \sigma ::= \cdot | \sigma, a \mapsto n \]
**Extra Credit.** Complete this section where instead memory locations can store any values (i.e., numbers or booleans).

1.1. Formalize the statics for IMP with two judgment forms $e : \tau$ and $c \text{ ok}$.

1.2. Formalize the dynamics for IMP by the following:
   (a) Define values and final states $e \text{ val}$ and $\langle c, \sigma \rangle \text{ final}$.
   (b) Define a big-step operational semantics with the judgment forms $\langle e, \sigma \rangle \Downarrow e'$ and $\langle c, \sigma \rangle \Downarrow \sigma'$.
   (c) Define a small-step operational semantics with the judgment forms $\langle e, \sigma \rangle \longrightarrow \langle e', \sigma' \rangle$ and $\langle c, \sigma \rangle \longrightarrow \langle c', \sigma' \rangle$.
   (d) State canonical forms. Then, state and prove progress and preservation.

2 **Language Implementation: $T$ with Products and Sums**

3 **Final Project Preparation**
Imperative Computation

What characterizes imperative computation?

Functional - is computing by "rewriting" or "reduction" or "simplifying"

\[(1+3)+3 \rightarrow 4+3\]

Code and data are "together"

Imperative separates code and data

The code modifies a memory store
\( \langle e, 0 \rangle \uparrow e' \) vs \( \langle c, 0 \rangle \uparrow c' \)

\( \langle e, 0 \rangle \uparrow \langle e', 0' \rangle \) but no store

\( \langle e, 0 \rangle \uparrow \langle c', 0' \rangle \) because commands can't return "nothing" values

<table>
<thead>
<tr>
<th>eval</th>
</tr>
</thead>
<tbody>
<tr>
<td>n val</td>
</tr>
<tr>
<td>b val</td>
</tr>
</tbody>
</table>

\( \text{c: final} \)

\( \text{skip final} \)
\[ \langle e, 0 \rangle \perp e' \]

\[ \langle c, 0 \rangle \perp \langle skp, 0 \rangle \]

\[ \langle c_1, 0 \rangle \perp \langle skp, 0 \rangle \]

\[ \langle c_2, 0 \rangle \perp \langle skp, 0 \rangle \]

\[ \langle c_1, c_2, 0 \rangle \perp 0 \]

\[ \langle e_1, 0 \rangle \perp e_1' \]

\[ \langle e_2, 0 \rangle \perp e_2' \]

\[ e_1 = e_2 \]

\[ b = (e'_1 = e'_2) \]

\[ \langle e_1 \rangle \perp e_1' \]

\[ \langle e_2 \rangle \perp e_2' \]

\[ \langle e_1, 0 \rangle \perp b \]
(1+2) = 3