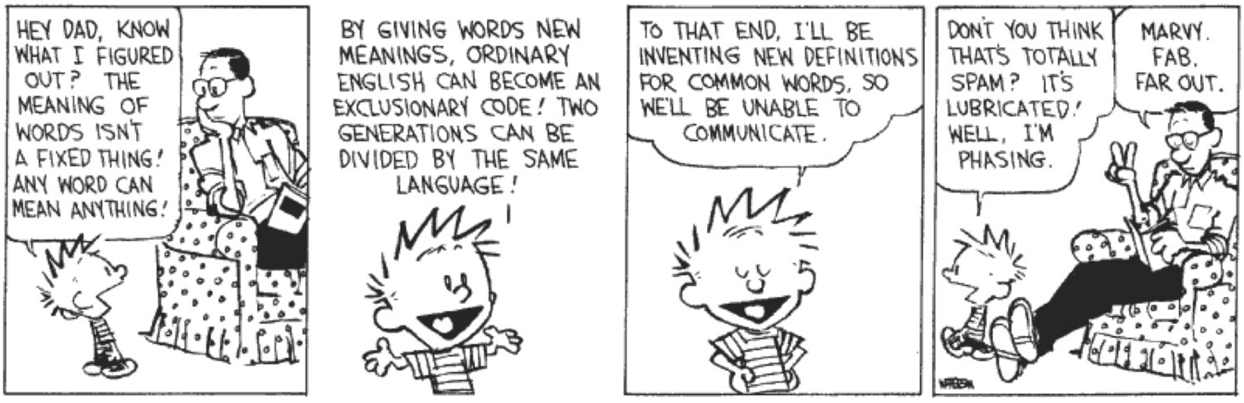
Meeting 9: Imperative Computation


HEY DAD, KNOW WHAT I FIGURED OUT? THE MEANING OF WORDS ISN'T A FIXED THING! ANY WORD CAN MEAN ANYTHING!
BY GIVING WORDS NEW MEANINGS, ORDINARY ENGLISH CAN BECOME AN EXCLUSIONARY CODE! TWO GENERATIONS CAN BE DIVIDED BY THE SAME LANGUAGE!
TO THAT END, I'LL BE INVENTING NEW DEFINITIONS FOR COMMON WORDS, SO WE'LL BE UNABLE TO COMMUNICATE.
DON'T YOU THINK THAT'S TOTALLY SPAM? IT'S LUBRICATED! WELL, I'M PHASING.
MARVY. FAB. FAR OUT.

Announcements

- Homework 2 due next week: Friday at 6:00pm
- Some Homework 0 feedback in GitHub
- Upcoming with Sean:
  - 2 sessions 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

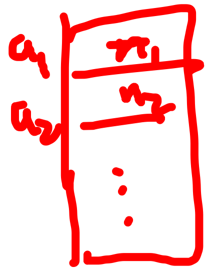
The tasks in this homework ask you to formalize and prove meta-theoretical properties of an imperative core language **IMP** based on your experience with **E**. This homework also asks you to implement an extension of **E** in OCaml to gain experience translating formalization to implementation.

1 Language Design: IMP

$\boxed{\langle e, \sigma \rangle \Downarrow e'}$ $\boxed{\langle c, \sigma \rangle \Downarrow \sigma'}$

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	τ	::=	`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)	$e_1 * e_2$	multiplication
			`eq`(e_1; e_2)	$e_1 == e_2$	equal
			`le`(e_1; e_2)	$e_1 <= e_2$	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				



left-to-right

Yes,
short-circuit
&& and ||

$y = (x == 3)$

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 σ is thus a mapping from addresses to numbers, written as follows:

$$\sigma \quad ::= \quad \cdot \mid \sigma, a \hookrightarrow n$$

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 $c \, \mathsf{ok}$.

1.2. Formalize the dynamics for IMP by the following:

(a) Define values and final states $e \, \mathsf{val}$ and ~~$\langle c, \sigma \rangle$ final~~ c fnl

(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 $\langle e, \sigma \rangle \longmapsto \langle e', \sigma' \rangle$ and $\langle c, \sigma \rangle \longmapsto \langle c', \sigma' \rangle$.

(d) State canonical forms. Then, state and prove progress and preservation.

2 Language Implementation: $\top$ with Products and Sums

3 Final Project Preparation

Imperative Computation

What characterizes imperative computation?

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

$(1+3)+3 \rightarrow 4+3$

code and data are "together"

Imperative separates code and data

The code modifies a memory store

$\boxed{\langle e, \sigma \rangle \Downarrow e'}$

value

$\langle e, \sigma \rangle \Downarrow \langle e', \sigma' \rangle$

but no store

$\boxed{\langle c, \sigma \rangle \Downarrow \sigma'}$

store but no value?

because commands don't return "interesting" values

$\langle c, \sigma \rangle \Downarrow \langle c', \sigma' \rangle$

$\boxed{\langle e, \sigma \rangle \mapsto \langle e', \sigma' \rangle}$

vs

$c \underset{\sigma}{\mapsto} e'$

$\boxed{\langle c, \sigma \rangle \mapsto \langle c', \sigma' \rangle}$

$\boxed{e \text{ val}}$

$\boxed{c \text{ final}}$

$$\frac{}{n \text{ val}}$$

$$\frac{}{b \text{ val}}$$

$$\frac{}{\text{skip final}}$$

$\boxed{\langle e, \sigma \rangle \Downarrow e'}$ $\boxed{\langle c, \sigma \rangle \Downarrow \sigma'}$

$\langle c, \sigma \rangle \Downarrow \langle c', \sigma' \rangle$

$$\frac{}{\langle \text{skip}, \sigma \rangle \Downarrow \langle \text{skip}, \sigma \rangle} \qquad \frac{}{\langle \text{skip}, \sigma \rangle \Downarrow \sigma}$$

$$\frac{\langle c_1, \sigma \rangle \Downarrow \sigma' \qquad \langle c_2, \sigma' \rangle \Downarrow \sigma''}{\langle c_1 ; c_2, \sigma \rangle \Downarrow \sigma''}$$

$$\frac{\langle e_1, \sigma \rangle \Downarrow \text{false}}{\langle e_1 \&\& e_2, \sigma \rangle \Downarrow \text{false}} \qquad \frac{\langle e_1, \sigma \rangle \Downarrow \text{true} \qquad \langle e_2, \sigma \rangle \Downarrow b_2}{\langle e_1 \&\& e_2, \sigma \rangle \Downarrow b_2}$$

$$\frac{\langle e_1, \sigma \rangle \Downarrow e_1' \qquad \langle e_2, \sigma \rangle \Downarrow e_2' \qquad b = (e_1' = e_2')}{\langle e_1 == e_2, \sigma \rangle \Downarrow b}$$

$(1+2) == 3$