CSCI 5535: Homework Assignment 2: Language Design and Implementation

David Baines [*][†][‡]

October 14, 2023

1 Language Design: IMP

1.1. The two judgment forms (one for expressions, the other for functions / commands, respectively e and c in **IMP**, are:

For e:

$$\frac{}{\Gamma \vdash \texttt{addr}[a] : \texttt{num}} \qquad \frac{}{\Gamma \vdash \texttt{num}[n] : \texttt{num}} \qquad \frac{}{\Gamma \vdash \texttt{bool}[b] : \texttt{bool}} \qquad \frac{\Gamma \vdash e_1 : \texttt{num} \qquad \Gamma \vdash e_2 : \texttt{num}}{\Gamma \vdash \texttt{plus}(e_1; e_2) : \texttt{num}}$$

$$\frac{\Gamma \vdash e_1 : \texttt{num} \qquad \Gamma \vdash e_2 : \texttt{num}}{\Gamma \vdash \texttt{times}(e_1; e_2) : \texttt{num}} \qquad \frac{\Gamma \vdash e_1 : \texttt{num} \qquad \Gamma \vdash e_2 : \texttt{num}}{\Gamma \vdash \texttt{eq}(e_1; e_2) : \texttt{bool}} \qquad \frac{\Gamma \vdash e_1 : \texttt{bool} \qquad \Gamma \vdash e_2 : \texttt{bool}}{\Gamma \vdash \texttt{eq}(e_1; e_2) : \texttt{bool}}$$

$$\frac{\Gamma \vdash e_1 : \texttt{num} \qquad \Gamma \vdash e_2 : \texttt{num}}{\Gamma \vdash \texttt{le}(e_1; e_2) : \texttt{bool}} \qquad \frac{\Gamma \vdash e : \texttt{num}}{\Gamma \vdash \texttt{not}(e) : \texttt{bool}} \qquad \frac{\Gamma \vdash e : \texttt{bool}}{\Gamma \vdash \texttt{not}(e) : \texttt{bool}}$$

$$\frac{\Gamma \vdash e_1 : \texttt{bool} \qquad \Gamma \vdash e_2 : \texttt{bool}}{\Gamma \vdash \texttt{and}(e_1; e_2) : \texttt{bool}} \qquad \frac{\Gamma \vdash e_1 : \texttt{bool} \qquad \Gamma \vdash e_2 : \texttt{bool}}{\Gamma \vdash \texttt{or}(e_1; e_2) : \texttt{bool}}$$

For c:

$$\frac{\Gamma \vdash a : \texttt{num} \qquad \Gamma \vdash e : \tau}{\Gamma \vdash \texttt{set}[a](e) : \texttt{num}} \qquad \frac{\Gamma \vdash c_1 : \tau \qquad \Gamma \vdash c_2 : \tau}{\Gamma \vdash \texttt{seq}(c_1; c_2) : \tau} \qquad \frac{\Gamma \vdash e : \texttt{bool} \qquad \Gamma \vdash c_1 : \tau \qquad \Gamma \vdash c_2 : \tau}{\Gamma \vdash \texttt{if}(e; c_1; c_2) : \tau}$$

$$\frac{\Gamma \vdash e : \texttt{bool} \qquad \Gamma \vdash c : \tau}{\Gamma \vdash \texttt{while}(e; c) : \tau}$$

1.2. (a) Define values and commands.

[*]https://courses.cs.cornell.edu/cs412/2004sp/lectures/lec13.pdf
[†]https://csci3155.cs.colorado.edu/csci3155-notes.pdf
[‡]https://www.hedonisticlearning.com/posts/understanding-typing-judgments.html

For “e val”:

$$\frac{}{\texttt{addr}[a]\ \mathsf{val}} \qquad \frac{}{\texttt{num}[n]\ \mathsf{val}} \qquad \frac{}{\texttt{bool}[b]\ \mathsf{val}}$$

“plus()”:

$$\frac{n_1 + n_2 = n}{\texttt{plus}(\texttt{num}[n_1];\texttt{num}[n_2]) \longmapsto \texttt{num}[n]} \qquad \frac{e_1 \longmapsto e_1'}{\texttt{plus}(e_1;e_2) \longmapsto \texttt{plus}(e_1';e_2)}$$

$$\frac{e\ \mathsf{val} \qquad e_2 \longmapsto e_2'}{\texttt{plus}(e_1;e_2) \longmapsto \texttt{plus}(e_1;e_2')}$$

“times()”:

$$\frac{n_1 * n_2 = n}{\texttt{times}(\texttt{num}[n_1];\texttt{num}[n_2]) \longmapsto \texttt{num}[n]} \qquad \frac{e_1 \longmapsto e_1'}{\texttt{times}(e_1;e_2) \longmapsto \texttt{times}(e_1';e_2)}$$

$$\frac{e\ \mathsf{val} \qquad e_2 \longmapsto e_2'}{\texttt{times}(e_1;e_2) \longmapsto \texttt{times}(e_1;e_2')}$$

“eq()”:

$$\frac{n_1 == n_2 \vdash b}{\texttt{eq}(\texttt{num}[n_1];\texttt{num}[n_2]) \longmapsto \texttt{bool}[b]} \qquad \frac{e_1 \longmapsto e_1'}{\texttt{eq}(e_1;e_2) \longmapsto \texttt{eq}(e_1';e_2)} \qquad \frac{e\ \mathsf{val} \qquad e_2 \longmapsto e_2'}{\texttt{eq}(e_1;e_2) \longmapsto \texttt{eq}(e_1;e_2')}$$

“le()”:

$$\frac{n_1 <= n_2 \vdash b}{\texttt{le}(\texttt{num}[n_1];\texttt{num}[n_2]) \longmapsto \texttt{bool}[b]} \qquad \frac{e_1 \longmapsto e_1'}{\texttt{le}(e_1;e_2) \longmapsto \texttt{le}(e_1';e_2)} \qquad \frac{e\ \mathsf{val} \qquad e_2 \longmapsto e_2'}{\texttt{le}(e_1;e_2) \longmapsto \texttt{le}(e_1;e_2')}$$

“not()”:

$$\frac{n_1 !\ n_2 \ ||\ b_1 !\ b_2 \vdash b}{\texttt{not}(\texttt{num}[n_1])\texttt{num}[n_2] \ ||\ \texttt{not}(\texttt{bool}[b_1])\texttt{bool}[b_2] \longmapsto \texttt{bool}[b]} \qquad \frac{e_1 \longmapsto e_1'}{\texttt{not}(e_1)e_2 \longmapsto \texttt{not}(e_1')e_2}$$

$$\frac{e\ \mathsf{val} \qquad e_2 \longmapsto e_2'}{\texttt{not}(e_1)e_2 \longmapsto \texttt{not}(e_1)e_2'}$$

“and()”:

$$\frac{b_1 \ \wedge \ b_2 \vdash b}{\texttt{and}(\texttt{bool}[b_1];\texttt{bool}[b_2]) \longmapsto \texttt{bool}[b]} \qquad \frac{e_1 \longmapsto e_1'}{\texttt{and}(e_1;e_2) \longmapsto \texttt{and}(e_1';e_2)}$$

$$\frac{e \ \mathsf{val} \qquad e_2 \longmapsto e_2'}{\texttt{and}(e_1;e_2) \longmapsto \texttt{and}(e_1;e_2')}$$

"or()":

$$\frac{b_1 \ \vee \ b_2 \vdash b}{\texttt{or}(\texttt{bool}[b_1];\texttt{bool}[b_2]) \longmapsto \texttt{bool}[b]} \qquad \frac{e_1 \longmapsto e_1'}{\texttt{or}(e_1;e_2) \longmapsto \texttt{or}(e_1';e_2)}$$

$$\frac{e \ \mathsf{val} \qquad e_2 \longmapsto e_2'}{\texttt{or}(e_1;e_2) \longmapsto \texttt{or}(e_1;e_2')}$$

For "c final":

"set()":

$$\frac{\texttt{addr}[a] = \texttt{num}[n]}{\texttt{set}[a](e) \longmapsto e} \qquad \frac{\texttt{addr}[a] \ \mathsf{val} \qquad e \longmapsto e'}{\texttt{set}[a](e) \longmapsto e'}$$

"skip()":

$$\frac{\texttt{addr}[a] = \texttt{num}[n]}{\texttt{skip} \longmapsto a}$$

"seq()":

$$\frac{c \longmapsto \texttt{ok}}{\texttt{seq}(c_1;c_2) \longmapsto c_2} \qquad \frac{c_1 \longmapsto c_1'}{\texttt{seq}(c_1;c_2) \longmapsto \texttt{seq}(c_1';c_2)} \qquad \frac{c_2 \longmapsto c_2'}{\texttt{seq}(c_1;c_2) \longmapsto \texttt{seq}(c_1;c_2')}$$

"if()":

$$\frac{\texttt{if}\ e\ \texttt{then}\ c_1\ \texttt{else}\ c_2}{\texttt{if}(e;c_1;c_2) \longmapsto c_1 \vee c_2} \qquad \frac{e \ \mathsf{val} \qquad e \longmapsto e'}{\texttt{if}(e;c_1;c_2) \longmapsto \texttt{if}(e';c_1;c_2)} \qquad \frac{e \ \mathsf{val} \qquad c_1 \longmapsto c_1'}{\texttt{if}(e;c_1;c_2) \longmapsto \texttt{if}(e;c_1';c_2)}$$

$$\frac{e \ \mathsf{val} \qquad c_2 \longmapsto c_2'}{\texttt{if}(e;c_1;c_2) \longmapsto \texttt{if}(e;c_1;c_2')}$$

"while()":

$$\frac{\texttt{while}\ e\ \texttt{do}\ c}{\texttt{while}(e;c) \longmapsto c} \qquad \frac{e = \texttt{bool}[b] \qquad e \ \mathsf{val}}{\texttt{while}(e;c) \longmapsto \texttt{while}(e;c)} \qquad \frac{e \longmapsto e'}{\texttt{while}(e;c) \longmapsto \texttt{while}(e';c)}$$

$$\frac{c \longmapsto c'}{\texttt{while}(e;c) \longmapsto \texttt{while}(e;c')}$$

(b) Define the small-step semantics.

For "*e*":

$$\frac{}{\langle a,\sigma\rangle \longmapsto \sigma(a)} \qquad \frac{}{\langle n,\sigma\rangle \longmapsto n} \qquad \frac{}{\langle \texttt{true},\sigma\rangle \longmapsto \texttt{true}} \qquad \frac{}{\langle \texttt{false},\sigma\rangle \longmapsto \texttt{false}}$$

$$\frac{\langle a_0,\sigma\rangle \longmapsto n_0 \qquad \langle a_1,\sigma\rangle \longmapsto n_1}{\langle n_0 - n_1,\sigma\rangle \longmapsto n} \qquad \frac{\langle a_0,\sigma\rangle \longmapsto n_0 \qquad \langle a_1,\sigma\rangle \longmapsto n_1}{\langle n_0 \times n_1,\sigma\rangle \longmapsto n}$$

$$\frac{\langle a_0,\sigma\rangle \longmapsto n \qquad \langle a_1,\sigma\rangle \longmapsto m}{\langle n = m,\sigma\rangle \longmapsto \texttt{true}} \qquad \frac{\langle a_0,\sigma\rangle \longmapsto n \qquad \langle a_1,\sigma\rangle \longmapsto m}{\langle n = m,\sigma\rangle \longmapsto \texttt{false}}$$

$$\frac{\langle a_0,\sigma\rangle \longmapsto n \qquad \langle a_1,\sigma\rangle \longmapsto m}{\langle n \le m,\sigma\rangle \longmapsto \texttt{true}} \qquad \frac{\langle a_0,\sigma\rangle \longmapsto n \qquad \langle a_1,\sigma\rangle \longmapsto m}{\langle n \le m,\sigma\rangle \longmapsto \texttt{false}}$$

$$\frac{\langle a_0,\sigma\rangle \longmapsto \texttt{true} \qquad \langle a_1,\sigma\rangle \longmapsto \texttt{true}}{\langle n \wedge m,\sigma\rangle \longmapsto \texttt{true}} \qquad \frac{\langle a_0,\sigma\rangle \longmapsto \texttt{false} \qquad \langle a_1,\sigma\rangle \longmapsto \texttt{true}}{\langle n \wedge m,\sigma\rangle \longmapsto \texttt{false}}$$

$$\frac{\langle a_0,\sigma\rangle \longmapsto \texttt{true} \qquad \langle a_1,\sigma\rangle \longmapsto \texttt{false}}{\langle n \wedge m,\sigma\rangle \longmapsto \texttt{false}} \qquad \frac{\langle a_0,\sigma\rangle \longmapsto \texttt{true} \qquad \langle a_1,\sigma\rangle \longmapsto \texttt{true}}{\langle n \vee m,\sigma\rangle \longmapsto \texttt{true}}$$

$$\frac{\langle a_0,\sigma\rangle \longmapsto \texttt{true} \qquad \langle a_1,\sigma\rangle \longmapsto \texttt{false}}{\langle n \vee m,\sigma\rangle \longmapsto \texttt{true}} \qquad \frac{\langle a_0,\sigma\rangle \longmapsto \texttt{false} \qquad \langle a_1,\sigma\rangle \longmapsto \texttt{true}}{\langle n \vee m,\sigma\rangle \longmapsto \texttt{true}}$$

$$\frac{\langle a_0,\sigma\rangle \longmapsto \texttt{false} \qquad \langle a_1,\sigma\rangle \longmapsto \texttt{false}}{\langle n \vee m,\sigma\rangle \longmapsto \texttt{false}}$$

For "*c*":

$$\frac{\langle a,\sigma\rangle \longmapsto \sigma(a) \qquad \langle e,\sigma\rangle \longmapsto e}{\langle a,\sigma\rangle \longmapsto \langle e,\sigma\rangle} \qquad \frac{}{\langle \texttt{skip},\sigma\rangle \longmapsto \sigma}$$

$$\frac{\langle c_0,\sigma\rangle \longmapsto \sigma'' \qquad \langle c_1,\sigma''\rangle \longmapsto \sigma'}{\langle c_0\,;c_1,\sigma\rangle \longmapsto \sigma'} \qquad \frac{\langle b,\sigma\rangle \longmapsto \texttt{true} \qquad \langle c_0,\sigma\rangle \longmapsto \sigma'}{\langle \textbf{if}\ b\ \textbf{then}\ c_0\ \textbf{else}\ c_1,\sigma\rangle \longmapsto \sigma'}$$

$$\frac{\langle b,\sigma\rangle \longmapsto \texttt{false} \qquad \langle c_0,\sigma\rangle \longmapsto \sigma'}{\langle \textbf{if}\ b\ \textbf{then}\ c_0\ \textbf{else}\ c_1,\sigma\rangle \longmapsto \sigma'} \qquad \frac{\langle b,\sigma\rangle \longmapsto \texttt{false}}{\langle \textbf{while}\ b\ \textbf{do}\ c,\sigma\rangle \longmapsto \sigma}$$

$$\frac{\langle b,\sigma\rangle \longmapsto \texttt{true} \qquad \langle c,\sigma\rangle \longmapsto \sigma'' \qquad \langle \textbf{while}\ b\ \textbf{do}\ c,\sigma''\rangle \longmapsto \sigma'}{\langle \textbf{while}\ b\ \textbf{do}\ c,\sigma\rangle \longmapsto \sigma'}$$

(c) i. Canonical Form Lemmas

If e val and $e : \tau$, then

1. If $\tau = \texttt{num}$, then $e = \texttt{num}[n]$ for some number n.
2. If $\tau = \texttt{bool}$, then $e = \texttt{bool}[b]$ for some boolean b.

This is basically saying that our expressions and commands are well-typed, meaning, for 1 and 2, if our expression evaluates and the type is `num` or `bool` (respectively), then the evaluated expression e's output is also of that type.

ii. Progress and Preservation.

Progress: $\forall e \in \mathsf{Exp}$, either e val or $\exists\, e' : e \longmapsto e'$.

Progress says that our expressions are well-typed, and that we can transition from expression to expression knowing that the types of one expression are compatible with the others they may encounter. Progress enables a program to not "get stuck."

Preservation: If $e : \tau$ and $e \longmapsto e'$, then $e' : \tau$.

Preservation is the idea that our expressions are well-typed consistently. This allows us to use induction on our proofs because, if one instance of an expression has a type, than all other instances of that expression must have the same type.

iii. Prove progress and preservation for commands.

Progress: $\forall c \in \mathsf{Cmd}$, either c final or $\exists\, c' : c \longmapsto c'$.

Progress is proved by the statics in 1.1. For example, if $c_1 : \tau$ and $c_2 : \tau$, then $\texttt{seq}(c_1; c_2) : \tau$. Since all of our commands are well-typed, progress is achieved.

Preservation: If $c : \tau$ and $c \longmapsto c'$, then $c' : \tau$.

Preservation is proven with our dynamics. For example, since $c \longmapsto \texttt{ok}$ and $\texttt{seq}(c_1; c_2) \longmapsto c_2$, know that $\texttt{seq}(c_1; c_2) \longmapsto \texttt{seq}(c_1'; c_2)$ for all $c \in \mathsf{Cmd}$.

(d) Did not have time.

(e) Did not have time.

2 Language Implementation: ETPS

See `hw02.ml` and `test_hw02.ml`.

I was able to download and start learning OCaml via "https://v2.ocaml.org/learn/tutorials/a_first_hour_with_ocaml.html", but I did not have time to even start the unit tests even though I understand the concepts. It's the grammar of proving them in OCaml that took an undue amount of time.

3 Final Project Preparation: Pre-Proposal

3.1. I'm working with Matt Buchholz and we're thinking about how PL verification can be applied to LLMs to ensure their outputs are well-formed. Below is the synopsis.

Basic problem: Can a language model trained to perform some task with code (like e.g. GitHub Copilot's code 'auto-complete' feature) be improved with constraints on the well-formedness of their outputs.

We draw inspiration from related work in natural languages, where e.g. Tziafas et al. [**?**] show that pre-training a BERT-based model on a part-of-speech tagging-like task improves downstream performance on other tasks. We could apply a similar approach, but instead of using the grammatical parse of natural language, we could train the model on a similar task with the syntactic parse of code. Or, we could follow an approach similar to Han et al. [**?**], by using ILP formulations about the syntactic well-formedness of a program to constrain some of the model's optimization. Liu et al. [**?**] do something similar, formulating 'code execution' as a pre-training task, and demonstrating a model pre-trained on that task performs better on downstream tasks, such as natural language-to-code translation.

We're still in the exploratory phase of the project; it's unclear how feasible this project is, since we're not familiar with the availability of open-source models for working with code or datasets for any training or benchmarking tasks. (And we're not going to try to train something from scratch.) Though, Wang et al. [**?**] seems to point to (at least some) models and datasets being readily available, including the CodeT5+ model they introduce.

For context: we *did* try to scan the papers of top PL conferences, but found papers which were largely not interesting to us (or beyond our understanding of PL such that it's hard to gauge the difficulty or novelty of a paper's achievement). Since we both have backgrounds in AI, something at the intersection of PL and AI seems most appropriate.

A Syntax of 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`$(e_1;e_2)$	$e_1 \mathbin{\&\&} e_2$	conjunction
			`or`$(e_1;e_2)$	$e_1 \mathbin{\vert\vert} e_2$	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				