

CSCI 5535: Homework Assignment 4: Program Verification and Implementation

Fall 2023: Due Friday, November 17, 2023

This homework has two parts. The first considers a deductive system for thinking about program correctness. The second considers a semantics that is closer to a real machine implementation.

Recall the evaluation guideline from the course syllabus.

Both your ideas and also the clarity with which they are expressed matter—both in your English prose and your code!

We will consider the following criteria in our grading:

- How well does your submission answer the questions? *For example, a common mistake is to give an example when a question asks for an explanation. An example may be useful in your explanation, but it should not take the place of the explanation.*
- How clear is your submission? *If we cannot understand what you are trying to say, then we cannot give you points for it. Try reading your answer aloud to yourself or a friend; this technique is often a great way to identify holes in your reasoning. For code, not every program that "works" deserves full credit. We must be able to read and understand your intent. Make sure you state any preconditions or invariants for your functions.*

Submission Instructions. Typesetting is preferred but scanned, clearly legible handwritten write-ups are acceptable. Please no other formats—no .doc or .docx. You may use whatever tool you wish (e.g., \LaTeX , Word, markdown, plain text, pencil+paper) as long as it is legibly converted into a pdf.

1 Axiomatic Semantics: IMP

We continue to consider the same language **IMP** with the syntax chart:

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($e_1; e_2$)	$e_1 * e_2$	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			

As before, addresses a represent static memory store locations and are drawn from some unbounded set Addr and all memory locations only store numbers. A store σ is thus a mapping from addresses to numbers, written as follows:

$$\text{Store } \sigma ::= \cdot | \sigma, a \mapsto n$$

The semantics of **IMP** is as a formalized as before operationally, and we consider the Hoare rules for partial correctness as in Chapter 6 of *FSPL*.

- 1.1. **Program Correctness.** Prove using Hoare rules the following property: if we start the command while e do $a := a + 2$ in a state that satisfies the assertion even(a), then it terminates in a state satisfying even(a). That is, prove the the following judgment:

$$\{ \text{even}(a) \} \text{ while } e \text{ do } a := a + 2 \{ \text{even}(a) \}$$

Hint: your proof should *not* use induction.

- 1.2. **Hoare Rules.** Consider an extension to **IMP**

$$c ::= \text{do}(c_1; e) \text{ do } c_1 \text{ while } e \text{ at-least-once looping}$$

with a command for at-least-once looping. Extend the Hoare judgment form $\{ A \} c \{ B \}$ for this command.

2 Abstract Machines and Control Flow

In this section, we will consider a new implementation of language **PCF** based on abstract machines (i.e., **K** from Chapter 28 of *PFPL*).

One aspect of a structural small-step operational semantics (as we used in previous assignments) that seems wasteful from an implementation perspective is that we “forget” where we are reducing at each step. An abstract machine semantics makes explicit the “program counter” in its state.

2.1. Give a specification for **K** as a call-by-value language. That is, modify the definition of the judgments f frame and $s \longrightarrow s'$ from Section 28.1 of *PFPL*. You will also need to update the auxiliary frame-typing judgment $f : \tau \rightsquigarrow \tau'$ from Section 28.2 in order to state safety.

2.2. **Safety.**

- (a) Prove preservation: if s ok and $s \longrightarrow s'$, then s' ok.
- (b) Prove progress: if s ok, then either s final or $s \longrightarrow s'$ for some state s' .

2.3. **Extra Credit: Implementation.**

- (a) Implement call-by-value **K**. You do need not include previously implemented language features (though you may include some of them if you want).

First, we have some new syntactic forms:

```
frames  f      type frame
stacks  k      type stack = frame list
states  s      type state = Eval of stack * exp | Ret of stack * exp
```

Then, we will implement functions that define both the static and dynamic semantics of the language.

```
[e'/x]e      val subst : exp -> var -> exp -> exp
eval         val is_val : exp -> bool
ΔΓ ⊢ e : τ   val exp_typ : kindctx -> typctx -> exp -> typ option
s ⟶ s'       val kstep : state -> state
s initial    val initial : exp -> state
s final      val final : exp -> state
s final      val is_final : state -> bool
k <: τ       val stack_typ : stack -> typ option
f : τ ~> τ'   val frame_typ : frame -> typ -> typ option
s ok         val is_ok : state -> bool
s ↪ok s'     val steps_pap : state -> state
```

The $s \hookrightarrow_{\text{ok}} s'$ is the analogous iterate-step-with-preservation-and-progress for states.

$$\frac{s \text{ ok} \quad s \text{ final}}{s \hookrightarrow_{\text{ok}} s} \qquad \frac{s \text{ ok} \quad s \longrightarrow s' \quad s' \hookrightarrow_{\text{ok}} s''}{s \hookrightarrow_{\text{ok}} s''}$$

- (b) **Exceptions.** Extend your **K** machine with exceptions as in Section 29.2. You may choose `nat` for the type of the value carried by the exception.
- (c) **Continuations.** Extend your **K** machine with continuations as in Section 30.2. Implementing continuations is independent of implementing exceptions, so you may choose to do either or both. (Technically, you can encode exceptions with continuations.)

3 Final Project Preparation: Start Paper Drafting

3.1. **Reading Papers.** Follow some citations based on the papers you chose in Homework 2 and read in Homework 3. List at least three cited papers that seems relevant to follow up on. Include a citation along with a URL for each paper. For each of the additional papers, and for each question below, write two concise sentences:

- (a) Why did *you* select this cited paper?
- (b) What is the relation between the “main idea” of this cited paper and the “main idea” of the paper that cites it? You may want to skim the introductory and concluding bits of the cited paper along with the related work in the citing paper.

3.2. **Project Repository.** Create a project repository shared with your partner via GitHub classroom. Coordinate so that one person creates the repository, and the other person joins it. Give the link to your project repository here. You may find it more convenient to collaborate on the proposal and paper draft with your partner in your project repo going forward. Just make sure to include the proposal and paper draft into your pdf submission here for our review.

3.3. **Proposal Revision.** Finalize your class project plan. Write an updated explanation of your plan (expanding and revising as necessary), and what you hope to accomplish with your project by the end of the semester. That is, on what artifact do you want to be graded?

Here are questions that you should address in your project proposal.

- (a) Define the problem that you will solve as concretely as possible. Provide a scope of expected and potential results. Give a few example programs that exhibit the problem that you are trying to solve.
- (b) What is the general approach that you intend to use to solve the problem?
- (c) Why do you think that approach will solve the problem? What resources (papers, book chapters, etc.) do you plan to base your solution on? Is there one in particular that you plan to follow? What about your solution will be similar? What will be different?
- (d) How do you plan to demonstrate your idea?
- (e) How will you evaluate your idea? What will be the measurement for success?

3.4. **Paper Drafting.** Write a first draft of the following sections of your paper.

- (a) **Related Work.** Take the papers you have read thus far (from Homeworks 2, 3, and 4) that are relevant to your project and write a draft of your Related Work section. Your Related Work Section should compare-and-contrast your (expected) contribution to the related work it builds on (grouped in categories as appropriate). Your Related Work Section *should not* simply summarize what you have read.
- (b) **Abstract.** Make an attempt at writing your 4-sentence Abstract, answering the following 4 questions:
1. What is the (technical) *problem* you are addressing?
 2. Why is addressing this problem *important* (i.e., what broader concern motivates solving this problem)?
 3. Why is solving this problem *hard* (i.e., why is what you are doing advancing the state-of-the-art)?
 4. What is your (expected) *contribution* (to addressing the problem)?

Be as concrete and concise as possible. Aim for clarity with just 4 sentences. This activity is extremely hard, and it is extremely unlikely this will be your final version of your Abstract. But the process of repeatedly trying to get these ~4 sentences to your satisfaction will help you better understand the problem you're tackling.

- (c) **Introduction.** Make an attempt at expanding the sentences of your Abstract into paragraphs in your Introduction. Then, make an attempt to state your (expected) technical contributions as bullets at the end of your Introduction that outline the technical sections of your paper.
- (d) **Overview Figure.** Make an attempt at the key figure you will use to illustrate (by example) your contribution.

Take a look at Simon Peyton Jones's "How to Write a Great Research Paper"¹ for further advice on writing research papers.

4 Feedback

- 4.1. **Assignment Feedback.** Complete the survey on the linked from the moodle after completing this assignment. Any non-empty answer will receive full credit for this part.

¹Simon Peyton Jones. How to Write a Great Research Paper. <https://www.microsoft.com/en-us/research/academic-program/write-great-research-paper/>