University of Massachusetts Lowell — Comp 3010: Organization of Programming Languages

Assignment 4

DUE: Tuesday, Oct 15, 2024, 11:59PM

Submission instructions: We are using GradeScope for PDF submission and the coding assignment. That is, you must submit your work as a PDF document. If you are not able to submit your assignment, please contact course staff. Some points to note:

- Your submission must be a PDF. We encourage you to use LaTeX (see below). If you produce your assignment using a different tool, please make sure your assignment is legible and follows these submission instructions.
- The first page of your assignment should state your name, UML ID, and a list of collaborators (if any), and *all other pages of your other pages should not contain any identifying information*, i.e., they should not contain your name.
- We will provide a LaTeX template you can use to write the answers to your assignment.

1 Equivalence of Semantics (40 points)

Recall the grammar for the calculator language used from HW2 and HW3. In this question, you will prove the equivalence of large-step and multi-step semantics.

NOTE: You have to use/fix your small-step and large-step inference rules from your HW2 and HW3 and use them in the proof. You **must** copy/write all the inference rules again for reference. During grading, we will not refer to your previous "HW2" or "HW3" submissions. As such, your current submission must be self-contained.

$$\begin{array}{lcl} n & \in & \mathbb{Z} \\ a & ::= & n \mid a_1 + a_2 \mid a_1 \times a_2 \\ b & ::= & \textbf{true} \mid \textbf{false} \mid a = a \mid a \neq a \\ & & \mid a \leq a \mid a > a \mid \neg b \mid b \&\& b \\ v & ::= & n \mid \textbf{true} \mid \textbf{false} \end{array}$$

(a) Prove the following theorem using induction.

Theorem (Equivalence of Semantics for Arithmetic Expressions)**.** *For all expressions a, integers n, we have:*

$$a \Downarrow n \iff a \longrightarrow^* n.$$

You need to prove the theorem in both the directions. That is, you need to provide proofs for the following statements.

$$\text{If } a \Downarrow n \text{ then } a \longrightarrow^* n$$

and

$$\text{If } a \longrightarrow^* n \text{ then } a \Downarrow n.$$

Colloquially, we refer to the former statement as forward direction and the latter as the reverse direction.

(i) What is the set that you will induct on in the forward direction?

(ii) What is the property that you will induct on in the forward direction? Formally state the property.

Hint: See Lecture 04 for an example formal statement.

(iii) Write the inductive reasoning principle for the forward direction.

(iv) Provide the actual proof for both forward and reverse directions.

Proof. Proof should go here. □

(b) Prove the following theorem using induction.

Theorem (Equivalence of Semantics for Boolean Expressions)**.** *For all expressions a, values $v \in \{\textsf{true}, \textsf{false}\}$, we have:*

$$b \Downarrow v \iff b \longrightarrow^* v.$$

(i) What is the set that you will induct on in the forward direction?

(ii) What is the property that you will induct on in the forward direction? Formally state the property.

(iii) Write the inductive reasoning principle for the forward direction.

(iv) Provide the proof for both forward and reverse directions. Note that you have to handle the proof for all cases. You cannot state "same as above" or "similar to the case".

Hint: When you handle cases that involve comparison between arithmetic expressions, you may need to invoke the lemmas proved in the previous part of the question.

Proof. Proof should go here. □

2 OCaml Program (10 points)

In this question, you will test the theory you have developed in the previous question. That is, you will test that multi-step semantics and large-step semantics are equivalent for arithmetic and boolean expressions.

In "hw2.ml" and "hw3.ml", you have implemented `sstep_aexp`, `sstep_bexp`, `lstep_aexp` and `lstep_bexp`, respectively. In this question, you will write a program `multi_step_aexp` and `multi_step_bexp` that compute the multi-step of arithmetic and boolean expressions, respectively. Submit the resulting file as "hw4.ml".

Empirically, test your implementation to check whether the value returned by `multi_sstep_aexp` is same as that of `lstep_aexp`, and `multi_sstep_bexp` is same as that of `lstep_bexp`.

Note the following:

(a) You can reuse/copy or fix your implementation of `sstep_bexp`, `sstep_aexp`, `lstep_aexp` and `lstep_bexp` from "hw2.ml" and "hw3.ml". **OR** you can also create a new implementation, however, you **must not** use the solutions posted for "hw2.ml" and "hw3.ml".

(b) Your "hw4.ml" should contain all the definitions. That is, it should define the following as described in HW2 and HW3, and additionally, it should define the functions related to multi-step relation. Recall that a multi-step function must invoke the small-step function until it evaluates to a value. You can refer to the multi-step interpreter that we developed in the class.

- `aexp`
- `bexp`
- `sstep_aexp`
- `sstep_bexp`
- `lstep_aexp`
- `lstep_bexp`

- multi_step_aexp
- multi_step_bexp

For reference, I have provided the signature of the new functions. The old functions have exactly the same signatures as described in "HW2" and "HW3".

```
(* Implement the following function by removing the exception.
*
* val multi_step_aexp : aexp -> aexp
*
* It should take an arithmetic expression a and return a
   arithmetic value v
* such that a multi-steps to the value v.
*)
let multi_step_aexp a = raise NoRuleApplies


(* Implement the following function by removing the exception
*
* val multi_step_bexp : bexp -> bexp
*
* It should take a boolean expression b and return a boolean
   value v
* such that b multi-steps to the value v.
*)
let multi_step_bexp b = raise NoRuleApplies
```