CS 599 A1: Assignment 1

Due Thursday, February 6, 2025

Total: 100 pts

Ankush Das

- This assignment is due midnight on the above date and it must be submitted electronically on Gradescope. Please create an account on Gradescope, if you haven't already done so.
- Please use the template provided on the course webpage to typeset your assignment and please include your name and BU ID in the Author section (above).
- Although it is not recommended, you can submit handwritten answers that are scanned as a PDF and clearly legible.
- You are provided a tex file, named asgn1.tex. It contains environments called solution. Please enter your solutions inside these environments.

1 Booleans [34 pts]

Problem 1 (9 pts) *In class, we looked at a simple arithmetic expression language called LL1. Similarly, for this assignment, we will study a simple boolean expression language LH1. This language contains 3 expressions written using the following syntax:*

$$\textit{Expressions} \quad e ::= \mathsf{true} \mid \mathsf{false} \mid \mathsf{if}\ e\ \mathsf{then}\ e\ \mathsf{else}\ e$$

The first two expressions are just the values true and false respectively. The expression if b then e_1 else e_2 *describes the standard* if *behavior, i.e., if b evaluates to true, then the expression reduces to e_1, else if b evaluates to false, then the expression reduces to e_2.*

(5 pts) Write and explain each rule of semantics for LH1. Note that the semantics will be written using the two standard judgments: e value and $e \mapsto e'$.

(4 pts) Consider the following expression:

if (if true then false else true) then (if false then false else true) else (if true then true else false)

Evaluate this expression using the rules of the semantics you just defined above.

Problem 2 (25 pts) *Now, we will try to express booleans using λ-calculus. The general technique is to represent the two values, i.e., true and false using normal forms (or values) in λ-calculus. Furthermore, they should be closed, that is, not contain any free variables. Since we need to distinguish between the two values, we need to introduce two variables. We then define rather arbitrarily one to be true and the other to be false*

$$\mathsf{true} = \lambda x.\, \lambda y.\, x \qquad \mathsf{false} = \lambda x.\, \lambda y.\, y$$

(10 pts) Our first task is to express the if *expression defined in the previous problem in λ-calculus. To do this, let's consider two constants, T and E. T represents the expression for the 'then' branch, while E represents the expression for the 'else' branch. In other words, construct the expression* if b then T else E *in λ-calculus, i.e., using λ expressions, function applications, and variables.*

(5 pts) Verify that the expression you constructed is indeed equivalent to $\mathsf{if}\ b\ \mathsf{then}\ T\ \mathsf{else}\ E$. *To do this, suppose the expression you constructed above is* I. *Then,* $(\lambda b.\, I)\ \mathsf{true}$ *should evaluate to* T. *And similarly,* $(\lambda b.\, I)\ \mathsf{false}$ *should evaluate to* E, *where* true *and* false *are the expressions above. Evaluate these two expressions using the semantics rules of* λ*-calculus to verify that they indeed evaluate to* T *and* F *respectively.*

(10 pts) Use the expression above to construct the and *and* not *functions.*

- and *should have the form* $\lambda b_1.\, \lambda b_2. \ldots$. *It takes two booleans as parameters and returns a boolean.*
- not *should have the form* $\lambda b. \ldots$, *i.e., takes a boolean as a parameter and returns a boolean.*

2 More Arithmetic in LL1 [50 pts]

Problem 3 (30 pts) *Let's return to the LL1 language discussed and now add an expression for multiplication in the language, so the formal syntax is*

$$\textit{Expressions} \quad e ::= \overline{n} \mid e \oplus e \mid e \otimes e$$

We call this language LH2.

(5 pts) First, define the rules of (small-step) semantics for LH2.

(10 pts) You will notice there is some non-determinism in the semantics rules, i.e., for a given expression e, *it will evaluate to two different values based on the order in which the rules are applied. Give an example of such an expression* e *and show that it evaluates to two different values in two different derivations. Again, show each step of the derivation.*

(10 pts) Our goal is to eliminate this non-determinism. For this, we have to come up with evaluation precedence, i.e., the order in which operators are applied. We arbitrarily decide that multiplication has higher precedence than addition, i.e., ***all multiplication must occur before any addition****. Define the semantics rules for such an evaluation.* ***Hint:*** *It might help to define an auxiliary judgment:* $e\ \mathsf{mult{-}value}$, *which holds iff expression* e *contains no addition operators.*

(5 pts) Take the same expression e *defined in sub-task 2 and evaluate it using the semantics rules defined in sub-task 3. Is there still any non-determinism in these rules? Can* e *still evaluate to different values?*

Problem 4 (20 pts) *Now, let's add floating-point numbers to LL1 and a few more changes.*

$$\textit{Expressions} \quad e ::= \mathsf{true} \mid \mathsf{false} \mid \mathsf{if}\ e\ \mathsf{then}\ e\ \mathsf{else}\ e \mid \overline{n} \mid \boxed{\underline{n}} \mid \boxed{e \oplus e} \mid \mathsf{let}\ x = e\ \mathsf{in}\ e \mid x$$

$$\textit{Types} \quad \tau ::= \mathsf{bool} \mid \mathsf{int} \mid \boxed{\mathsf{float}}$$

As you can notice, there is a new type for floating-point expressions. In addition, we modify the behavior of the addition operation $\oplus$. *For integers and floats, the* $\oplus$ *operator is standard 'addition' with* ***the important caveat that the*** $\oplus$ ***operator can be used to add integers with floats****. Integers and floats can be on either side of the operator and adding an integer with a float will result in a float. We also introduce* $\underline{n}$ *to represent floating-point values. We call this language LH3.*

Solve the following problems for LH3:

(10 pts) Define the rules of the type system for LH3. Although not necessary, but try to use as few rules as possible.

(10 pts) Define the rules of the small-step semantics for LH3. To define the semantics, you can use the $+$ *operator to add integer/float values.*

3 More Practice with λ-calculus [16 pts]

Problem 5 (16 pts) *In this problem, fill in the blanks to make the following typing judgments in λ-calculus valid, or briefly explain that it is impossible to do so. This might require defining either (i) an expression with a given type, or (ii) the typing context for an expression, or (iii) the type of a given expression in a typing context, or (iv) a combination of the above. (2 pts each)*

1. $\cdot \vdash \boxed{} : \alpha \to \alpha$
2. $y : \beta \vdash \boxed{} : \alpha \to \beta$
3. $\cdot \vdash \boxed{} : \alpha \to \beta$
4. $\boxed{} \vdash x\ x : \boxed{}$
5. $\cdot \vdash \boxed{} : \alpha \to (\alpha \to \alpha)$
6. $\cdot \vdash \boxed{} : (\alpha \to \alpha) \to \alpha$
7. $\boxed{} \vdash \lambda x.\, x\ (x\ x) : \boxed{}$
8. $\cdot \vdash \lambda f.\, \lambda g.\, \lambda x.\, (f\ x)\ (g\ x) : (\alpha \to \boxed{}) \to (\alpha \to \boxed{}) \to (\alpha \to \boxed{})$

Note that for these problems, α, β, ... are (distinct) arbitrary types, i.e., they can be instantiated with any type. The expression you provide should typecheck for any arbitrary type. For example, for the first problem, if you write $\lambda x.\, x + 10$, then that expression does not work for an arbitrary type α; it only works if α is int*. So, $\lambda x.\, x + 10$ is not a valid answer.*