CSCI 3155: Midterm

Spring 2017: Thursday, March 2, 2017

- Please begin by acknowledging the CU Honor Code Pledge by signing in the space below the pledge.

  > *On my honor, as a University of Colorado at Boulder student, I will neither give nor receive unauthorized assistance on this work.*

  Signed: _____

- Please then print your name on this page in the space provided below. Please also write your name at the top of each page in case the pages become separated.
- You have 75 minutes for this exam.
- You may use both sides of a single 8.5" $\times$ 11" sheet of handwritten notes you created. Please turn in this page of notes with your exam—please make sure your name is on it if you would like it back. Otherwise, this exam is closed book, closed computer, and closed mobile device.
- This exam has 3 questions for a total of 113 points.
- Answer the questions in the spaces provided on the front side of the question sheets. You may use the back side for scratch work. The back side should not contain answers unless clearly marked and referenced.
- This exam has 8 pages, including 1 additional scratch page.

Name: _____

Question:	1	2	3	Total
Points:	58	16	39	113
Score:				

1. **Syntax**.

   (a) **Grammars**. Consider the following grammar:

$$\begin{array}{lcl} s & ::= & c \mid s \textbf{ if } e \textbf{ else } c \mid c\ s \\ c & ::= & ; \mid e \mid x \leftarrow e \\ e & ::= & f \mid f < e \mid \textbf{if } e \textbf{ then } e \textbf{ else } e \textbf{ fi} \\ f & ::= & n \mid x \end{array}$$

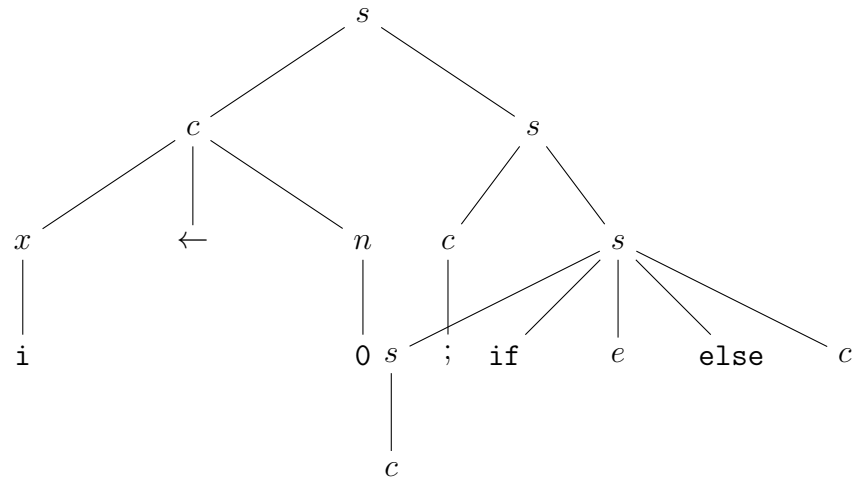
   The terminals n and x correspond to numbers (e.g., 0, 1, 42) and variable identifiers (e.g., `x`, `y`), respectively. The starting non-terminal is s.

   i. 8 points *Exercise 2.1.a.* Is the sentence

$$\texttt{i} \leftarrow \texttt{0};\ \texttt{j} \leftarrow \texttt{2}\ \textbf{if}\ \texttt{i} < \texttt{0}\ \textbf{else}\ \texttt{j} \leftarrow \texttt{3}$$

   in the language described by the above grammar? If so, give a **parse tree** for the sentence. If not, briefly explain why in no more than 1–2 sentences.

Solution: Yes.



   ii. 8 points *Exercise 2.1.b.* Is the above grammar ambiguous? If so, demonstrate the ambiguity by drawing two parse trees for the same sentence. If not, briefly explain why in no more than 1–2 sentences.

Solution: Yes, this grammar is ambiguous. Consider the sentence given above, that is, $\texttt{x} = \texttt{1};\ \textbf{fixmey} = \texttt{2z} < 0$. This sentence can be read as

$$(\texttt{i} \leftarrow \texttt{0};\ \texttt{j} \leftarrow \texttt{2})\ \textbf{if}\ \texttt{i} < \texttt{0}\ \textbf{else}\ \texttt{j} \leftarrow \texttt{3} \quad (1)$$

$$\texttt{i} \leftarrow \texttt{0};\ (\texttt{j} \leftarrow \texttt{2}\ \textbf{if}\ \texttt{i} < \texttt{0}\ \textbf{else}\ \texttt{j} \leftarrow \texttt{3}) \quad (2)$$

   iii. 5 points *Exercise 2.2.* What is the relative precedence of the following binary operators: '<' and '←' ? Briefly explain why in no more than 1–2 sentences.

Solution: < has the highest precedence, followed by $\leftarrow$, because the c non-terminal matches the string before any e non-terminal in the grammar, so $f < e$ substrings will always be matched lower in the parse tree.

iv. *Lab 2.3.c.* The grammar on the previous page is repeated here for your reference:

$$\begin{array}{rcl} s & ::= & c \mid s \textbf{ if } e \textbf{ else } c \mid c\ s \\ c & ::= & \ ; \mid e \mid x \leftarrow e \\ e & ::= & f \mid f < e \mid \textbf{if } e \textbf{ then } e \textbf{ else } e \textbf{ fi} \\ f & ::= & n \mid x \end{array}$$

α) [5 points] Is the ';' operator left associative, right associative, or neither? Briefly explain why in no more than 1–2 sentences.

Solution: ; is neither. The c rule is neither left nor right recursive.

β) [5 points] Is the '<' operator left associative, right associative, or neither? Briefly explain why in no more than 1–2 sentences.

Solution: = is right associative. The production with = is right recursive.

(b) [10 points] **Resolving Ambiguity**. *Exercise 2.2.* Recall the ambiguous grammar from Exercise 2:

$$e ::= n \mid \textbf{if } (e)\ e \textbf{ else } e \mid e + n$$

that can be witnessed by two parse trees for the expression **if** (0) 1 **else** $2 + 42$. Rewrite the above grammar into another grammar that accepts the same set of strings but is unambiguous. Resolve the ambiguity of the grammar by making + higher precedence than **if-else** (i.e., **if** (0) 1 **else** $2 + 42$ parses as **if** (0) 1 **else** $(2 + 42)$). Briefly explain why your refactored grammar accepts the same set of strings as the original grammar in no more than 1–2 sentences.

Solution:

$$\begin{array}{rcl} e & ::= & t \mid \textbf{if } (e)\ e \textbf{ else } e \\ t & ::= & n \mid t + n \end{array}$$

(c) **Inductive Definitions**. For this question, consider languages over the alphabet $\Sigma \stackrel{\text{def}}{=} \{0, 1\}$ consisting of a 0 and a 1.

i. [8 points] *Lab 2.3.c.* Let us define a *palindrome* as a string that is identical to its reversal. For example, "`0`", "`00`", "`010`" are palindromes. Complete the following grammar so that the language described by P (i.e., $\mathcal{L}(P)$) is the language of palindromes over Σ.

$$P ::= B \mid \boxed{\text{0B0} \mid \text{1B1}}$$
$$B ::= 0 \mid 1 \mid \boxed{\varepsilon}$$

Use only the above non-terminals, but you may use as many productions as you need.

ii. [9 points] *Lab 2.2.e.* Consider the following grammar that defines bit-strings s and expressions over bit-strings e:

$$e ::= {\sim}s$$
$$s ::= \varepsilon \mid s0 \mid s1$$

Let us write $e \Downarrow s$ for the judgment form that gives a semantics to bit-string expressions where ${\sim}s$ corresponds to bit-wise complement. For example, the following judgments should hold:

$${\sim}0 \Downarrow 1 \qquad {\sim}1 \Downarrow 0 \qquad {\sim}01 \Downarrow 10$$

Define this judgment form with a set of inference rules. Hint: you should have an inference rule for each production of s.

Solution:

$$\frac{}{{\sim}\epsilon \Downarrow \epsilon}\ \textsc{Epsilon} \qquad \frac{{\sim}s \Downarrow s'}{{\sim}s1 \Downarrow s'0}\ \textsc{OneFlip} \qquad \frac{{\sim}s \Downarrow s'}{{\sim}s0 \Downarrow s'1}\ \textsc{ZeroFlip}$$

2. **Small-Step Substitution Semantics**.

(a) Consider a dynamically-typed language with numbers and functions:

$$\begin{array}{rcl} e & ::= & n \mid x \mid (x) \Rightarrow e_1 \mid e_1(e_2) \mid \mathsf{typeerror} \\ v & ::= & n \mid (x) \Rightarrow e_1 \end{array}$$

where n and x correspond to numbers (e.g., 0, 1, 42) and variable identifiers (e.g., `x`, `y`), respectively. The values v of this language are numbers n and function values $(x) \Rightarrow e_1$. The expression $e_1(e_2)$ corresponds to function call.

A one-step reduction judgment for this language has the following form: $e \longrightarrow e'$. Informally, this judgment says "Closed expression e takes one step of evaluation to expression e'." This judgment gives a small-step operational semantics for this language, which we define in this question.

i. 6 points *Exercise 3.1.a.* Suppose that SEARCH rules to evaluate call expressions $e_1(e_2)$ are as follows:

$$\frac{e_1 \longrightarrow e_1'}{e_1(v_2) \longrightarrow e_1'(v_2)} \qquad \frac{e_2 \longrightarrow e_2'}{e_1(e_2) \longrightarrow e_1(e_2')}$$

What is stated about the order of evaluation for call expressions in the above rules? Explain in 1–2 sentences.

Solution: These rule say that the evaluation of call expressions is right-to-left.

ii. 10 points *Lab 3.4.* Replace the SEARCH rules given in part i with new SEARCH rules that implement the opposite order of evaluation.

Solution:

$$\frac{e_1 \longrightarrow e_1'}{e_1(e_2) \longrightarrow e_1'(e_2)} \qquad \frac{e_2 \longrightarrow e_2'}{v_1(e_2) \longrightarrow v_1(e_2')}$$

3. **A Language for Boolean Expressions**

(a) We wish to create a language for boolean expressions called Bex, with constants `t` and `f`, and operations ! and $->$, and ().

i. 10 points **Grammar**
$->$ is right associative. ! has higher precedence than $->$. So `!t->t->f` should be read as `(!t)->(t->f)`.
Create a BNF grammar for this language. Include a rule for ()!

Solution:
```
E ::= V | V -> E
V ::= B | !V
B ::= t | f | (E)
```

ii. 10 points **Semantics and Judgment Form** We define a new judgment form:

$$E \Downarrow b$$

E is a Bex expression, and b is either `True` or `False`. For $\Downarrow$ the following should hold true:

$$!t \Downarrow False \qquad !f \Downarrow True$$
$$f-> f \Downarrow True \qquad f-> t \Downarrow True$$
$$t-> f \Downarrow False \qquad t-> t \Downarrow True$$

Observe that $f-> x \Downarrow True$ for any x. Write the inference rules for $\Downarrow$ Create a short-circuting rule for ->. A few have been given for you.

$$\textsc{TrueAxiom}\ \frac{}{t \Downarrow True} \qquad \textsc{FalseAxiom}\ \frac{}{f \Downarrow False} \qquad \textsc{Parens}\ \frac{e \Downarrow b}{(e) \Downarrow b}$$

Solution:

$$\textsc{Not}\ \frac{e \Downarrow b}{!e \Downarrow !b} \qquad \textsc{FalseImp}\ \frac{e \Downarrow False}{e-> f \Downarrow True} \qquad \textsc{TrueImp}\ \frac{e \Downarrow True, f \Downarrow b}{e-> f \Downarrow b}$$

(b) **Abstract Syntax Tree**

i. 6 points Create the case classes for a scala AST class Bex for this language.

sealed abstract class `Bex`

Solution:

```
object T extends Bex
object F extends Bex
object Not(b:Bex) extends Bex
object Imp(b:Bex, c:Bex)
```

ii. [3 points] Write the following string as a Bex object, the result of parsing the string with the grammar above.
!(T− > T)− > F− >!T

Solution:

```
Imp(Not(Imp(T, T)), Imp(F, Not(T)))
```

iii. [10 points] Evaluation
Write a scala eval function with type eval(e : Bex) : Bool. Implement Short-circuiting for -> on the first argument.

Solution:

```
eval(e:Bex) : Bool = e match {
  case T => True
  case F => False
  case Not(b) => !eval(b)
  case Imp(b, c) => eval(b) match {
    case F => True
    case T => eval(c)
  }
}
```

Additional Scratch Page