CS 476 Midterm

Sandeep Joshi

TOTAL POINTS

100 / 100

QUESTION 1

Problem 1 20 pts

1.1 1a 10 / 10

+ **0 pts** graded
- ✓ + **1 pts** O production 1 takes string
- ✓ + **2 pts** O production 2 takes int * string
- ✓ + **1 pts** I production 1 takes ast_O * ast_O
- ✓ + **1 pts** I production 2 takes ast_O * ast_O
- ✓ + **2 pts** I production 3 takes ast_I * int
- ✓ + **2 pts** no extra productions
- ✓ + **1 pts** mostly correct syntax

1.2 1b 10 / 10

+ **0 pts** graded
- ✓ + **1 pts** root: do
- ✓ + **2 pts** L: take
- ✓ + **1 pts** R: 5
- ✓ + **2 pts** LL: pieces
- ✓ + **1 pts** LR: box
- ✓ + **1 pts** LLL: 3
- ✓ + **1 pts** LLR: paper
- ✓ + **1 pts** correct types/tree structure

QUESTION 2

2 Problem 2 25 / 25

+ **0 pts** graded
- ✓ + **5 pts** assignment rule
- ✓ + **3 pts** expression has type int
- ✓ + **5 pts** if-then-else rule
- ✓ + **3 pts** bool rule
- ✓ + **3 pts** num rule
- ✓ + **3 pts** var rule
- ✓ + **3 pts** proof tree structure

- **1 pts** Extra rules

- **1 pts** ill-formed rule applications

- **2 pts** missing one top line

- **3 pts** missing multiple top lines

1 No rule needed here

QUESTION 3

3 Problem 3 15 / 15

+ **0 pts** graded
- ✓ + **3 pts** typecheck e
- ✓ + **3 pts** typecheck e1
- ✓ + **3 pts** typecheck e2
- ✓ + **3 pts** e must be Tbool
- ✓ + **3 pts** e1 and e2 must be t

+ **2 pts** e1 and e2 same type, but not t

- **3 pts** only typechecks one side

QUESTION 4

4 Problem 4 15 / 15

+ **0 pts** graded
- ✓ + **5 pts** if-then-else-command rule
- ✓ + **5 pts** correct next step
- ✓ + **5 pts** evaluate true

- **2 pts** missing top line

- **2 pts** Extra rules

+ **3 pts** (if others 0) Correct tree structure

+ **2.5 pts** false above the line, true below

+ **2 pts** turned true into false

QUESTION 5

5 Problem 5 25 / 25

- ✓ + **0 pts** graded
- ✓ + **5 pts** gave small-step or hybrid rules
- ✓ + **7 pts** handles c0 and c1 cases
- ✓ + **5 pts** handles default case
- ✓ + **5 pts** evaluates e correctly
- ✓ + **3 pts** default case only applies when e is not 0 or 1

+ **3 pts** evaluates e only in non-default cases

- **2 pts** Didn't use switch in bottom rule

- **2 pts** extra conditions

+ **3 pts** has the right idea intuitively

CS 476 Fall 2018 Midterm

Name:	SANDEEP JOSHI
NetID:	sjoshi37

- You have **50 minutes** to complete this exam.
- This is a **closed-book** exam.
- Do not share anything with other students. Do not talk to other students. Do not look other students' exams. Do not expose your exam to easy viewing by other students. Violation of any of these rules will count as cheating.
- If you believe there is an error or an ambiguous question, you may seek clarification from the instructor. Please speak quietly or write your question out.
- Including this cover sheet and rules at the end, there are 8 pages to the exam, including one blank page for workspace. Please verify that you have all 8 pages.
- Please write your name and NetID in the spaces above, and also in the provided space at the top of every sheet.
- Show your work. Partial credit will be given for incomplete answers.
- If you finish with time remaining, check your work!

Question	Points	Score
1	20	
2	25	
3	15	
4	15	
5	25	
Total:	100	

CS 476 Fall 2018 Midterm Name: sjoshi23 SANDEEP JOSHI

Problem 1. (20 points)

Consider the following BNF grammar:

O ::= ⟨ident⟩ | ⟨#⟩ pieces of ⟨ident⟩
I ::= take O from O | put O in O | do I ⟨#⟩ times

(a) (10 points) Write OCaml datatypes `ast_O` and `ast_I` that encode the abstract syntax trees of O and I respectively. You may represent ⟨ident⟩ with the `string` type and ⟨#⟩ with the `int` type.

ast_O = Item of string | Pieces of int * string

ast_I = Takefrom of ast_O * ast_O | Putin of ast_O * ast_O |
Do of ast_I * int

(b) (10 points) Write the instance of the `ast_I` type corresponding to the AST for the term

do (take 3 pieces of paper from box) 5 times

If you prefer, you can instead draw the AST for the term.

Do (Takefrom(Pieces (3, "paper"), Item("box")), 5)

Problem 2. (25 points)

The typing rules for a simple imperative language are given in Appendix A. Write a proof tree for the judgment

$$\Gamma \vdash \text{x := if false then 5 else x : ok}$$

given that $\Gamma(\text{x}) = \text{int}$.

$$\dfrac{\dfrac{\Gamma(x) = int}{\Gamma \vdash x : int} \quad \cancel{\Gamma(x) = int} \quad \dfrac{\dfrac{\text{false is a boolean}}{\Gamma \vdash false : bool} \quad \dfrac{\text{5 is a number}}{\Gamma \vdash 5 : int} \quad \cancel{\Gamma(x) = int} \quad \dfrac{\Gamma(x) = int}{\Gamma \vdash x : int}}{\Gamma \vdash \text{if false then 5 else } x : int}}{\Gamma \vdash x := \text{if false then 5 else } x : ok}$$

Problem 3. (15 points)

Suppose you were writing a type-checking function `typecheck_exp : context -> exp -> typ -> bool` that takes a type context `gamma`, an expression `e`, and a type `t` that is either `Tint` or `Tbool`, and returns `true` if gamma $\vdash$ e : t and `false` otherwise. Fill in the skeleton below by translating the typing rule for the if-then-else expression into OCaml code.

```
let rec typecheck_exp (gamma : context) (e : exp) (t : typ) : bool =
  match e with
  | If (e, e1, e2) -> (match typecheck-exp gamma e Tbool with
                       ~~| true -> typecheck-exp e1 t && typecheck-exp~~
                       | true -> (typecheck-exp gamma e1 t && typecheck-exp gamma e2 t)
                       ~~| false ->~~
                       | _ -> false )
```

Problem 4. (15 points)

The operational semantics rules for a simple imperative language are shown in Section B. Write a proof tree for the next step that

(if true then y := 5 else skip, {y = 3})

takes.

$$\dfrac{\dfrac{\text{true is a boolean}}{(\text{true}, \{y=3\}) \Downarrow \text{true}}}{(\text{if true then } y := 5 \text{ else skip}, \{y=3\}) \to (y := 5, \{y=3\})}$$

Problem 5. (25 points)
Write small-step operational semantics rules for the construct

switch e { case 0: c_0; case 1: c_1; default: c_2 }

executes c_0 if the value of e is 0, c_1 if the value of e is 1, and c_2 if the value of e is anything else.
Hint: Think about which existing construct is most similar to switch, and look at its rules.

~~0 is a number~~

$$\frac{(e, \sigma) \Downarrow 0}{(\text{switch } e \{\text{case } 0: c_0;\ \text{case } 1: c_1;\ \text{default}: c_2\}, \sigma) \rightarrow (c_0, \sigma)}$$

~~1 is a number~~

$$\frac{(e, \sigma) \Downarrow 1}{(\text{switch } e \{\text{case } 0: c_0;\ \text{case } 1: c_1;\ \text{default}: c_2\}, \sigma) \rightarrow (c_1, \sigma)}$$

~~v is a number~~

$$\frac{(e, \sigma) \Downarrow v \quad ((v = 0), \sigma) \Downarrow \text{false} \quad ((v = 1), \sigma) \Downarrow \text{false}}{(\text{switch } e \{\text{case } 0: c_0;\ \text{case } 1: c_1;\ \text{default}: c_2\}, \sigma) \rightarrow (c_2, \sigma)}$$

~~v is a number~~

A Typing Rules for a Simple Imperative Language

$$\frac{(n \text{ is a number})}{\Gamma \vdash n : \text{int}} \qquad \frac{(b \text{ is a boolean})}{\Gamma \vdash b : \text{bool}} \qquad \frac{(\Gamma(x) = \tau)}{\Gamma \vdash x : \tau}$$

$$\frac{\Gamma \vdash e_1 : \text{int} \quad \Gamma \vdash e_2 : \text{int}}{\Gamma \vdash e_1 \oplus e_2 : \text{int}} \text{ where } \oplus \text{ is an arithmetic operator}$$

$$\frac{\Gamma \vdash e_1 : \text{bool} \quad \Gamma \vdash e_2 : \text{bool}}{\Gamma \vdash e_1 \otimes e_2 : \text{bool}} \text{ where } \otimes \text{ is a boolean operator} \qquad \frac{\Gamma \vdash e_1 : \tau \quad \Gamma \vdash e_2 : \tau}{\Gamma \vdash e_1 = e_2 : \text{bool}}$$

$$\frac{\Gamma \vdash e : \text{bool} \quad \Gamma \vdash e_1 : \tau \quad \Gamma \vdash e_2 : \tau}{\text{if } e \text{ then } e_1 \text{ else } e_2 : \tau}$$

$$\frac{(\Gamma(x) = \tau) \quad \Gamma \vdash e : \tau}{\Gamma \vdash x := e : \text{ok}} \qquad \frac{\Gamma \vdash c_1 : \text{ok} \quad \Gamma \vdash c_2 : \text{ok}}{\Gamma \vdash c_1; c_2 : \text{ok}}$$

$$\frac{\Gamma \vdash e : \text{bool} \quad \Gamma \vdash c_1 : \text{ok} \quad \Gamma \vdash c_2 : \text{ok}}{\Gamma \vdash \text{if } e \text{ then } c_1 \text{ else } c_2 : \text{ok}} \qquad \frac{\Gamma \vdash e : \text{bool} \quad \Gamma \vdash c : \text{ok}}{\Gamma \vdash \text{while } e \text{ do } c : \text{ok}}$$

B Operational Semantics for a Simple Imperative Language

$$\frac{(n \text{ is a number})}{(n, \sigma) \Downarrow n} \qquad \frac{(b \text{ is a boolean})}{(b, \sigma) \Downarrow b} \qquad \frac{(\sigma(x) = v)}{(x, \sigma) \Downarrow v}$$

$$\frac{(e_1, \sigma) \Downarrow v_1 \quad (e_2, \sigma) \Downarrow v_2 \quad (v_1 \oplus v_2 = v)}{(e_1 \oplus e_2, \sigma) \Downarrow v} \text{ where } \oplus \text{ is an arithmetic or boolean operator}$$

$$\frac{(e, \sigma) \Downarrow \text{true} \quad (e_1, \sigma) \Downarrow v}{(\text{if } e \text{ then } e_1 \text{ else } e_2, \sigma) \Downarrow v} \qquad \frac{(e, \sigma) \Downarrow \text{false} \quad (e_2, \sigma) \Downarrow v}{(\text{if } e \text{ then } e_1 \text{ else } e_2, \sigma) \Downarrow v}$$

$$\frac{(e, \sigma) \Downarrow v}{(x := e, \sigma) \to (\text{skip}, \sigma[x \mapsto v])}$$

$$\frac{(c_1, \sigma) \to (c_1', \sigma')}{(c_1; c_2, \sigma) \to (c_1'; c_2, \sigma')} \qquad \frac{}{(\text{skip}; c_2, \sigma) \to (c_2, \sigma)}$$

$$\frac{(e, \sigma) \Downarrow \text{true}}{(\text{if } e \text{ then } c_1 \text{ else } c_2, \sigma) \to (c_1, \sigma)} \qquad \frac{(e, \sigma) \Downarrow \text{false}}{(\text{if } e \text{ then } c_1 \text{ else } c_2, \sigma) \to (c_2, \sigma)}$$

$$\frac{}{(\text{while } e \text{ do } c, \sigma) \to (\text{if } e \text{ then } (c; \text{while } e \text{ do } c) \text{ else skip}, \sigma)}$$

C Scratch Space