

# CS 320 Final Exam

Total: 100 pts

CS 320 Course Staff

Name: \_\_\_\_\_

BU ID: \_\_\_\_\_

## Instructions

- Please write your name and BU ID.
- This exam has 5 problems.
- Please read all problems before writing your solutions. You may find some problems easier than others.
- The total duration of the exam is 2 hrs.
- The typing/semantic rules that you need for each problem are provided **on the same page**. For these rules, sides conditions appear in boxes.

*This page is intentionally left blank.*

# 1 OCaml Programming

Recall the definition of rose trees:

```
type 'a rtree = Node of 'a * 'a rtree list
```

In this problem, you can use any of the following functions from the standard library:

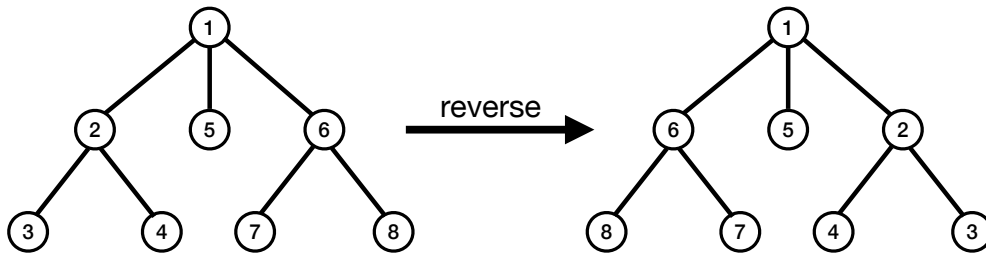
```
val List.rev : 'a list -> 'a list
val List.map : ('a -> 'b) -> 'a list -> 'b list
val List.fold_right : ('a -> 'acc -> 'acc) -> 'a list -> 'acc -> 'acc
val List.fold_left : ('acc -> 'a -> 'acc) -> 'acc -> 'a list -> 'acc
```

You cannot use any other functions from the standard library.

**Problem 1 (20 pts)** *Implement the function*

```
val reverse : 'a rtree -> 'a rtree
```

so that it reverses the order of nodes, i.e., reverse  $t$  is the mirror image of  $t$  about the  $y$ -axis, e.g.,



Equivalently, the following assertion holds:

```
let n x l = Node (x, l)
let test =
  reverse (n 1 [n 2 [n 3 []; n 4 []]; n 5 []; n 6 [n 7 []; n 8 []]])
  =
    n 1 [n 6 [n 8 []; n 7 []]; n 5 []; n 2 [n 4 []; n 3 []]]
let _ = assert test
```

**Solution.**

**Solution.** (Contd.)

## 2 Formal Grammar

Consider the following grammar for a toy imperative language with references.

$$\begin{aligned} \langle p \rangle &::= \langle w \rangle = \langle e \rangle \langle p \rangle \mid \langle w \rangle = \langle e \rangle \mid \langle e \rangle \\ \langle e \rangle &::= \langle e2 \rangle \langle o \rangle \langle e \rangle \mid \langle e2 \rangle \\ \langle e2 \rangle &::= \& \langle w \rangle \mid \langle w \rangle \mid ( \langle e \rangle ) \mid 1 \mid 2 \mid 3 \\ \langle o \rangle &::= + \mid - \mid * \mid / \\ \langle w \rangle &::= * \langle w \rangle \mid x \mid y \mid z \end{aligned}$$

This grammar expresses that a program ( $\langle p \rangle$ ) is a sequence of assignment statements ( $\langle w \rangle = \langle e \rangle$ ) ending with an optional expression ( $\langle e \rangle$ ).<sup>1</sup> For example,  $x = 1 \ y = \&x \ x + *y$  is a well-formed program. Note that it can be written more naturally as:<sup>2</sup>

```
x = 1
y = &x
x + *y
```

**Problem 2 (20 pts)** *Demonstrate that the above grammar is ambiguous by giving a program that has multiple parse trees. Write down the program and draw two distinct parse trees for it.*

**Solution.**

---

<sup>1</sup>This is similar to how Rust works.

<sup>2</sup>The meaning of this program doesn't matter for this problem, but we might understand this program to be saying:  $x$  is assigned to 1, and then  $y$  is assigned to a reference to  $x$ , and then the output of the program is  $x$  plus the value that  $y$  refers to, i.e.,  $y$  is *dereferenced*.

**Repeated Grammar:**

$\langle p \rangle ::= \langle w \rangle = \langle e \rangle \langle p \rangle \mid \langle w \rangle = \langle e \rangle \mid \langle e \rangle$   
 $\langle e \rangle ::= \langle e_2 \rangle \langle o \rangle \langle e \rangle \mid \langle e_2 \rangle$   
 $\langle e_2 \rangle ::= \& \langle w \rangle \mid \langle w \rangle \mid ( \langle e \rangle ) \mid 1 \mid 2 \mid 3$   
 $\langle o \rangle ::= + \mid - \mid * \mid /$   
 $\langle w \rangle ::= * \langle w \rangle \mid x \mid y \mid z$

**Solution.** (Contd.)

### 3 Message Passing Semantics

Recall the following constructs for sending and receiving messages on channels:

- ‘let  $(tx, rx) = \text{new\_channel}$  in  $e$ ’ creates a new channel with two ends: the variable  $tx$  is the sender’s end, which can be used to send messages on the channel, and the variable  $rx$  is the receiver’s end, which can be used to receive messages on the channel;
- ‘send  $e_1$  on  $tx$  in  $e$ ’ evaluates the expression  $e_1$  and sends its value on  $tx$ ;
- ‘let  $x = \text{recv } rx$  in  $e$ ’ receives a value on  $rx$  and binds it to the variable  $x$ .

In this problem, you’ll be looking at the *semantics* of message passing. We take judgments to be of the form  $\langle \mathcal{E}, e \rangle \Downarrow v$ , where the environment  $\mathcal{E}$  not only contains variable-value bindings, but also channel-buffer bindings. Formally,  $\mathcal{E}$  is additionally allowed to have bindings of the form  $(tx, rx) \mapsto l$  where  $l$  is a list of values. Whenever a message is sent on  $tx$ , it gets added to the back of this list, and whenever a message is received on  $rx$ , it gets removed from the front of the list (i.e., the list acts as a queue).

The semantic rules for these constructs are as follows:

$$\frac{\langle \mathcal{E}[(tx, rx) \mapsto []], e \rangle \Downarrow v}{\langle \mathcal{E}, \text{let } (tx, rx) = \text{new\_channel in } e \rangle \Downarrow v} \text{NEWCHAN}$$

$$\frac{\langle \mathcal{E}, e_1 \rangle \Downarrow v_1 \quad \boxed{\mathcal{E}((tx, rx)) = \ell} \quad \langle \mathcal{E}[(tx, rx) \mapsto \ell @ [v_1]], e \rangle \Downarrow v}{\langle \mathcal{E}, \text{send } e_1 \text{ on } tx \text{ in } e \rangle \Downarrow v} \text{SEND}$$

$$\frac{\boxed{\mathcal{E}((tx, rx)) = v_1 :: \ell} \quad \langle \mathcal{E}[(tx, rx) \mapsto \ell][x \mapsto v_1], e \rangle \Downarrow v}{\langle \mathcal{E}, \text{let } x = \text{recv } rx \text{ in } e \rangle \Downarrow v} \text{RECV}$$

**Additional Rules Needed:**

$$\frac{\boxed{\mathcal{E}(x) = v}}{\langle \mathcal{E}, x \rangle \Downarrow v} \text{VAR} \qquad \frac{\boxed{n \text{ is an integer literal}}}{\langle \mathcal{E}, n \rangle \Downarrow n} \text{INTLIT}$$

**Problem 3 (20 pts)** Provide a derivation demonstrating that the following expression evaluates to 5.

```

let (tx, rx) = new_channel in
  send 5 on tx in
  let x = recv rx in
  x

```

**Solution.**

Repeated Rules:

$$\begin{array}{c}
 \boxed{\mathcal{E}(x) = v} \text{ VAR} \\
 \langle \mathcal{E}, x \rangle \Downarrow v \\
 \\
 \boxed{n \text{ is an integer literal}} \text{ INTLIT} \\
 \langle \mathcal{E}, n \rangle \Downarrow n \\
 \\
 \frac{\langle \mathcal{E}[(tx, rx) \mapsto []], e \rangle \Downarrow v}{\langle \mathcal{E}, \text{let } (tx, rx) = \text{new\_channel in } e \rangle \Downarrow v} \text{ NEWCHAN} \\
 \\
 \frac{\langle \mathcal{E}, e_1 \rangle \Downarrow v_1 \quad \boxed{\mathcal{E}((tx, rx)) = \ell} \quad \langle \mathcal{E}[(tx, rx) \mapsto \ell @ [v_1]], e \rangle \Downarrow v}{\langle \mathcal{E}, \text{send } e_1 \text{ on } tx \text{ in } e \rangle \Downarrow v} \text{ SEND} \\
 \\
 \frac{\boxed{\mathcal{E}((tx, rx)) = v_1 :: \ell} \quad \langle \mathcal{E}[(tx, rx) \mapsto \ell][x \mapsto v_1], e \rangle \Downarrow v}{\langle \mathcal{E}, \text{let } x = \text{recv } rx \text{ in } e \rangle \Downarrow v} \text{ RECV}
 \end{array}$$

Repeated Expression:

```

let (tx, rx) = new_channel in
  send 5 on tx in
  let x = recv rx in
  x

```

Solution. (Contd.)

## 4 Polymorphism

**Problem 4 (15 pts)** *Determine an OCaml expression with the following type.*

$((('a \rightarrow 'b) \rightarrow 'b) \rightarrow 'c) \rightarrow 'a \rightarrow 'c$

**Solution.**

**Problem 5 (25 pts)** Determine the principle type of the following expression.

$$\text{fun } x \rightarrow \text{fun } y \rightarrow x \ y \ y$$

To receive full credit, you must provide:

1. A derivation of the judgment  $\cdot \vdash \text{fun } x \rightarrow \text{fun } y \rightarrow x \ y \ y : \tau \dashv \mathcal{C}$  for some type  $\tau$  and constraints  $\mathcal{C}$ . You may write your derivation in compact form.
2. A step-by-step demonstration of the solving of constraints  $\mathcal{C}$  using the unification algorithm we discussed in class.
3. The principle type of the above expression, clearly demarcated in your solution.

This problem requires the following rules:

$$\frac{(x : \sigma) \in \Gamma \quad \boxed{\sigma \text{ is a monotype}}}{\Gamma \vdash x : \sigma \dashv \emptyset} \text{VARMONO} \qquad \frac{\boxed{\alpha \text{ is fresh}} \quad \Gamma, x : \alpha \vdash e : \tau \dashv \mathcal{C}}{\Gamma \vdash \text{fun } x \rightarrow e : \alpha \rightarrow \tau \dashv \mathcal{C}} \text{FUN}$$

$$\frac{\Gamma \vdash e_1 : \tau_1 \dashv \mathcal{C}_1 \quad \Gamma \vdash e_2 : \tau_2 \dashv \mathcal{C}_2 \quad \boxed{\alpha \text{ is fresh}}}{\Gamma \vdash e_1 \ e_2 : \alpha \dashv \tau_1 \doteq \tau_2 \rightarrow \alpha, \mathcal{C}_1, \mathcal{C}_2} \text{APP}$$

**Solution.**

**Rules Repeated:**

$$\frac{(x : \sigma) \in \Gamma \quad \boxed{\sigma \text{ is a monotype}}}{\Gamma \vdash x : \sigma \dashv \emptyset} \text{VARMONO} \qquad \frac{\boxed{\alpha \text{ is fresh}} \quad \Gamma, x : \alpha \vdash e : \tau \dashv \mathcal{C}}{\Gamma \vdash \text{fun } x \rightarrow e : \alpha \rightarrow \tau \dashv \mathcal{C}} \text{FUN}$$

$$\frac{\Gamma \vdash e_1 : \tau_1 \dashv \mathcal{C}_1 \quad \Gamma \vdash e_2 : \tau_2 \dashv \mathcal{C}_2 \quad \boxed{\alpha \text{ is fresh}}}{\Gamma \vdash e_1 e_2 : \alpha \dashv \tau_1 \doteq \tau_2 \rightarrow \alpha, \mathcal{C}_1, \mathcal{C}_2} \text{APP}$$

**Expression Repeated:**

$\text{fun } x \rightarrow \text{fun } y \rightarrow x y y$

**Solution.** (Contd.)

*This page is intentionally left blank.*