

CS 320 Midterm (Version 1)

Total: 100 pts

CS 320 Course Staff

October 16, 2025

Name: _____

BU ID: _____

Instructions

- Please write your name and BU ID.
- This midterm 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 1 hr 15 mins.
- 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 Wrong Functions [40 pts]

Recall the typing rule for functions:

$$\frac{\Gamma, x : \tau_1 \vdash e : \tau_2}{\Gamma \vdash \text{fun } x \rightarrow e : \tau_1 \rightarrow \tau_2} \text{FUN}$$

Suppose we made a mistake when writing down this typing rule, and forgot to require the type of x in the context of the premise to be the same as the argument type of $\text{fun } x \rightarrow e$ in the conclusion (the difference in the rule is boxed).

$$\frac{\Gamma, \boxed{x : \tau'_1} \vdash e : \tau_2}{\Gamma \vdash \text{fun } x \rightarrow e : \tau_1 \rightarrow \tau_2} \text{FUNWRONG}$$

One important property of a programming language is called **preservation**: given any well-typed expression e_0 , if e_0 has type τ_0 , then it must evaluate to a value of the same type τ_0 . In this problem, you will show that a programming language with the rule FUNWRONG violates this property. You'll need (some of) the following typing rules.

$$\begin{array}{c} \boxed{\text{n is an int literal}} \\ \Gamma \vdash n : \text{int} \end{array} \text{INTLIT} \qquad \frac{}{\Gamma \vdash \text{true} : \text{bool}} \text{TRUELIT} \qquad \frac{\boxed{x : \tau \in \Gamma}}{\Gamma \vdash x : \tau} \text{VAR}$$

$$\frac{\Gamma \vdash e_1 : \tau_2 \rightarrow \tau \quad \Gamma \vdash e_2 : \tau_2}{\Gamma \vdash e_1 e_2 : \tau} \text{APP}$$

And you'll need (some of) the following semantics rules.

$$\begin{array}{c} \boxed{\text{n is an int literal}} \\ n \Downarrow n \end{array} \text{INTLITEVAL} \qquad \frac{}{\text{true} \Downarrow \top} \text{TRUELITEVAL} \qquad \frac{}{\text{fun } x \rightarrow e \Downarrow (\lambda x. e)} \text{FUN EVAL}$$

$$\frac{e_1 \Downarrow (\lambda x. e) \quad e_2 \Downarrow v_2 \quad \boxed{e' = [v_2/x]e} \quad e' \Downarrow v}{e_1 e_2 \Downarrow v} \text{APPEVAL}$$

Problem 1 (20 pts) Determine an expression e_0 such that $\emptyset \vdash e_0 : \text{bool}$ and $e_0 \Downarrow 42$ according to the above rules (including FUNWRONG). You must provide:

- (i) a typing derivation of $\emptyset \vdash e_0 : \text{bool}$ and
- (ii) a semantic derivation of $e_0 \Downarrow 42$.

Solution. Here's how to arrive at the expression: your expression must contain a function in it (otherwise you won't be able to use the FUNWRONG rule). But since your expression must have type **bool**, it cannot just be a function; the function must be applied to something. The easiest way to do this is to have an expression of the form $e_1 e_2$ where e_1 is a function which is applied to e_2 .

The next step is to notice the FUNWRONG rule: how it adds x of an arbitrary type to the context. So, the function body of e_1 must use x , so the simplest possible way to do that is if e_1 is $\text{fun } x \rightarrow x$.

Finally, the last step: what can we apply this function to so that it returns 42? Thus, e_2 must be 42. Hence, e_0 must be $(\text{fun } x \rightarrow x) 42$.

Typing Derivation:

$$\frac{\frac{x : \text{bool} \vdash x : \text{bool}}{\emptyset \vdash \text{fun } x \rightarrow x : \text{int} \rightarrow \text{bool}} \text{FUNWRONG} \quad \emptyset \vdash 42 : \text{int}}{\emptyset \vdash (\text{fun } x \rightarrow x) 42 : \text{bool}} \text{APP}$$

Semantic Derivation:

$$\frac{\frac{}{\text{fun } x \rightarrow x \Downarrow (\lambda x. x)} \text{FUN EVAL} \quad 42 \Downarrow 42 \quad 42 \Downarrow 42}{(\text{fun } x \rightarrow x) 42 \Downarrow 42} \text{APPEVAL}$$

Thus, the expression e_0 is shown to have type **bool**, while it evaluates to an integer 42.

Another important property of a programming language is called **progress**: given any well-typed expression e_0 , it must evaluate to a value. Formally, if $\emptyset \vdash e_0 : \tau_0$, then there must exist a value v_0 such that $e_0 \Downarrow v_0$. In this problem, you will show that a programming language with the rule FUNWRONG violates this property as well. You'll need (some of) the following typing rules.

$$\begin{array}{c}
\boxed{x : \tau \in \Gamma} \text{ VAR} \quad \frac{\Gamma, x : \tau'_1 \vdash e : \tau_2}{\Gamma \vdash \text{fun } x \rightarrow e : \tau_1 \rightarrow \tau_2} \text{ FUNWRONG} \quad \frac{\Gamma \vdash e_1 : \tau_2 \rightarrow \tau \quad \Gamma \vdash e_2 : \tau_2}{\Gamma \vdash e_1 e_2 : \tau} \text{ APP} \\
\\
\frac{\Gamma \vdash e : \text{bool} \quad \Gamma \vdash e_1 : \tau \quad \Gamma \vdash e_2 : \tau}{\Gamma \vdash \text{if } e \text{ then } e_1 \text{ else } e_2 : \tau} \text{ IF} \quad \frac{\Gamma \vdash e_1 : \text{int} \quad \Gamma \vdash e_2 : \text{int}}{\Gamma \vdash e_1 + e_2 : \text{int}} \text{ ADDINT} \\
\\
\frac{}{\Gamma \vdash \text{true} : \text{bool}} \text{ TRUELIT} \quad \frac{}{\Gamma \vdash \text{false} : \text{bool}} \text{ FALSELIT} \quad \frac{\boxed{n \text{ is an int literal}}}{\Gamma \vdash n : \text{int}} \text{ INTLIT}
\end{array}$$

You'll also need (some of) the following semantics rules.

$$\begin{array}{c}
\frac{}{\text{fun } x \rightarrow e \Downarrow (\lambda x. e)} \text{ FUNEVAL} \quad \frac{e_1 \Downarrow (\lambda x. e) \quad e_2 \Downarrow v_2 \quad \boxed{e' = [v_2/x]e} \quad e' \Downarrow v}{e_1 e_2 \Downarrow v} \text{ APPEVAL} \\
\\
\frac{e \Downarrow \top \quad e_1 \Downarrow v_1}{\text{if } e \text{ then } e_1 \text{ else } e_2 \Downarrow v_1} \text{ IFTRUEEVAL} \quad \frac{e \Downarrow \perp \quad e_2 \Downarrow v_2}{\text{if } e \text{ then } e_1 \text{ else } e_2 \Downarrow v_2} \text{ IFFALSEEVAL} \\
\\
\frac{e_1 \Downarrow v_1 \quad e_2 \Downarrow v_2 \quad \boxed{v = v_1 + v_2}}{e_1 + e_2 \Downarrow v} \text{ ADDINTEVAL} \quad \frac{\boxed{n \text{ is an int literal}}}{n \Downarrow n} \text{ INTLITEVAL} \\
\\
\frac{}{\text{false} \Downarrow \perp} \text{ FALSELITEVAL} \quad \frac{}{\text{true} \Downarrow \top} \text{ TRUELITEVAL}
\end{array}$$

Problem 2 (20 pts) Determine a type τ_0 and a **well-typed** expression e_0 of type τ_0 such that there does not exist a value v_0 with a valid derivation of $e_0 \Downarrow v_0$. Additionally, provide

(i) a typing derivation of $\emptyset \vdash e_0 : \tau_0$ and

(ii) a brief English explanation of why your choice of e_0 does not evaluate to a value.

Solution. Here's how to arrive at the expression: again your expression must contain a function in it (otherwise you won't be able to use the FUNWRONG rule). But since your expression must not be able to evaluate, it cannot just be a function expression (remember a function directly evaluates to itself); so again, the function must be applied to something. Hence, the expression again should have the form $e_1 e_2$ where e_1 is a function which is applied to e_2 .

Now, this time we want an expression that does not evaluate and we have already seen how we can interchange booleans and integers in the typing derivation. So, what if we decided to have an expression where booleans are added (see the hint about addition being in the rules provided).

So, there are many possibilities now, and here's one of them: let's make e_1 be $\text{fun } x \rightarrow x + x$. Even though the function is expecting integers, we can give it a boolean as an argument, due to FUNWRONG. So, let's make e_2 to be true .

Hence, e_0 must be $(\text{fun } x \rightarrow x + x) \text{ true}$.

Typing Derivation:

$$\frac{\frac{\frac{x : \text{int} \vdash x : \text{int} \text{ VAR} \quad x : \text{int} \vdash x : \text{int} \text{ VAR}}{x : \text{int} \vdash x + x : \text{int}} \text{ ADDINT}}{\emptyset \vdash \text{fun } x \rightarrow x + x : \text{bool} \rightarrow \text{int}} \text{ FUNWRONG} \quad \frac{}{\emptyset \vdash \text{true} : \text{bool}} \text{ TRUELIT}}{\emptyset \vdash (\text{fun } x \rightarrow x + x) \text{ true} : \text{int}} \text{ APP}$$

But it is easy to see that this cannot evaluate because, once we substitute `true` for x in the function body, there are no rules to evaluate `true + true`.

2 OCaml Programming [40 pts]

Recall rose trees from Assignment 5:

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

For the following problem, you can use arithmetic operators from the standard library and any of the following higher-order functions:

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

Problem 3 (20 pts) *Implement the function*

```
val sum : int rtree -> int
```

so that `sum t` is the sum of all the elements at the nodes of `t`. For example, the following assertion holds:

```
let _ = assert (sum (Node 1 [Node 2 []; Node 3 []]) = 6)
```

Solution.

```
let rec sum t =
  match t with
  | Node(x, l) -> x + List.fold_right (fun x s -> sum x + s) l 0
```

Problem 4 (20 pts) *Determine the smallest type `task` such that the following function is well-typed.*

```
let output (t : task) (x : int) =  
  match t with  
  | Compute(f) -> f x  
  | Double(l) -> List.map (fun e -> 2 * e) l  
  | Filter(p, l) -> List.filter (p x) l  
  | Append(l) -> List.fold_right (fun a b -> a::b) l.second l.first
```

The following signatures are provided for your benefit:

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

Solution.

```
type task =  
  | Compute of (int -> int list)  
  | Double of int list  
  | Filter of (int -> int -> bool) * int list  
  | Append of {first : int list; second : int list}
```

3 Message Passing [20 pts]

Suppose we introduce a new construct to our language for sending and receiving messages on channels. We introduce three new syntactic constructs:

- ‘let $(tx, rx) = \text{new_channel}$ in e ’: create 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 ’: evaluate expression e_1 and send its value on tx .
- ‘let $x = \text{recv } rx$ in e ’: receive a value on rx and bind it to variable x .

We introduce two types: τ sender and τ receiver for the sender’s and receiver’s end of the channel respectively. We also provide the typing rules for these constructs:

$$\frac{\Gamma, tx : \tau_1 \text{ sender}, rx : \tau_1 \text{ receiver} \vdash e : \tau_2}{\Gamma \vdash \text{let } (tx, rx) = \text{new_channel in } e : \tau_2} \text{NEWCHAN}$$

$$\frac{\Gamma \vdash tx : \tau_1 \text{ sender} \quad \Gamma \vdash e_1 : \tau_1 \quad \Gamma \vdash e : \tau_2}{\Gamma \vdash \text{send } e_1 \text{ on } tx \text{ in } e : \tau_2} \text{SEND}$$

$$\frac{\Gamma \vdash rx : \tau_1 \text{ receiver} \quad \Gamma, x : \tau_1 \vdash e : \tau_2}{\Gamma \vdash \text{let } x = \text{recv } rx \text{ in } e : \tau_2} \text{RECV}$$

You’ll need the following typing rules.

$$\frac{\boxed{n \text{ is an int literal}}}{\Gamma \vdash n : \text{int}} \text{INTLIT}$$

$$\frac{\boxed{x : \tau \in \Gamma}}{\Gamma \vdash x : \tau} \text{VAR}$$

Problem 5 (20 pts) Based on these rules, provide a derivation for the following program having type int under the empty context $\emptyset$. You should use Δ as shorthand for the context $\{tx : \text{int sender}, rx : \text{int receiver}\}$. Otherwise, you should use no shorthand in your derivation.

```

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

```

Solution.

$$\frac{\frac{\frac{\frac{\Delta \vdash tx : \text{int sender}}{\text{VAR}} \quad \frac{\Delta \vdash 5 : \text{int}}{\text{INTLIT}} \quad \frac{\frac{\frac{\Delta \vdash rx : \text{int receiver}}{\text{VAR}} \quad \frac{\Delta, x : \text{int} \vdash x : \text{int}}{\text{VAR}}}{\Delta \vdash \text{let } x = \text{recv } rx \text{ in } x : \text{int}} \text{RECV}}{\Delta \vdash \text{send 5 on tx in let } x = \text{recv } rx \text{ in } x : \text{int}} \text{SEND}}{\emptyset \vdash \text{let } (tx, rx) = \text{new_channel in send 5 on tx in let } x = \text{recv } rx \text{ in } x : \text{int}} \text{NEWCHAN}$$