

# Practice Midterm Exam

CAS CS 320: Principles of Programming Languages

Name:

BUID:

- ▷ You will have approximately 75 minutes to complete this exam. Make sure to read every question, some are easier than others.
- ▷ Do not remove any pages from the exam.
- ▷ Make very clear what your final solution for each problem is (e.g., by surrounding it in a box). We reserve the right to mark off points if we cannot tell what your final solution is.
- ▷ You must show your work on all problems unless otherwise specified. A solution without work will be considered incorrect (and will be investigated for potential academic dishonesty).
- ▷ Unless stated otherwise, you should only need the rules provided **in that problem** for your derivations.
- ▷ We will not look at any work on the pages marked “*This page is intentionally left blank.*” You should use these pages for scratch work.

*This page is intentionally left blank.*

# 1 Bad Inference Rules

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  $\tau_0$ , then it must evaluate to a value of the same type  $\tau_0$ . 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.

$$\begin{array}{c} \frac{\text{n is an integer literal}}{\Gamma \vdash n : \text{int}} \text{INTLIT} \qquad \frac{}{\Gamma \vdash \text{true} : \text{bool}} \text{TRUE} \qquad \frac{x : \tau \in \Gamma}{\Gamma \vdash x : \tau} \text{VAR} \\[10pt] \frac{\Gamma \vdash e_1 : \tau_2 \rightarrow \tau \quad \Gamma \vdash e_2 : \tau_2}{\Gamma \vdash e_1 e_2 : \tau} \text{APP} \end{array}$$

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

$$\begin{array}{c} \frac{\text{n is an integer literal for } n}{n \Downarrow n} \text{INTLITE} \qquad \frac{}{\text{true} \Downarrow \top} \text{TRUEE} \qquad \frac{}{\text{fun } x \rightarrow e \Downarrow \lambda x.e} \text{FUNE} \\[10pt] \frac{e_1 \Downarrow \lambda x.e \quad e_2 \Downarrow v_2 \quad e' = [v_2/x]e \quad e' \Downarrow v}{e_1 e_2 \Downarrow v} \text{APPE} \end{array}$$

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

- A. a typing derivation of  $\{\} \vdash e_0 : \text{bool}$  and
- B. a semantic derivation of  $e_0 \Downarrow 42$ .

The problem is repeated here for convenience.

$$\begin{array}{c}
\frac{\text{n is an integer literal}}{\Gamma \vdash \text{n} : \text{int}} \text{INTLIT} \qquad \frac{}{\Gamma \vdash \text{true} : \text{bool}} \text{TRUE} \qquad \frac{x : \tau \in \Gamma}{\Gamma \vdash x : \tau} \text{VAR} \\
\\
\frac{\Gamma \vdash e_1 : \tau_2 \rightarrow \tau \quad \Gamma \vdash e_2 : \tau_2}{\Gamma \vdash e_1 \ e_2 : \tau} \text{APP} \qquad \frac{\text{n is an integer literal for } n}{n \Downarrow n} \text{INTLITE} \qquad \frac{}{\text{true} \Downarrow \top} \text{TRUEE} \\
\\
\frac{}{\text{fun } x \rightarrow e \Downarrow \lambda x.e} \text{FUNE} \qquad \frac{e_1 \Downarrow \lambda x.e \quad e_2 \Downarrow v_2 \quad e' = [v_2/x]e \quad e' \Downarrow v}{e_1 \ e_2 \Downarrow v} \text{APPE}
\end{array}$$

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

- A. a typing derivation of  $\{\} \vdash e_0 : \text{bool}$  and
- B. a semantic derivation of  $e_0 \Downarrow 42$ .

## 2 OCaml Programming

Implement the function `val lookup : ('a * 'b) list -> 'a -> 'b option` so that `lookup l a` is

- ▷ `Some b` if there is pair of the form `(a, b)` in `l`. If there are multiple such pairs, the output should be the value associated with the *rightmost* such pair in `l`;
- ▷ `None` if there is no such pair.

For example, the following assertions hold.

```
let _ = assert (lookup [] 2 = None)
let _ = assert (lookup [(2, "2")] 2 = Some "2")
let _ = assert (lookup [(2, "2"); (3, "3"); (2, "4"); (3, "5")] 2 = Some "4")
let _ = assert (lookup [(2, "2"); (3, "3"); (2, "4"); (3, "5")] 4 = None)
```

You *cannot* use anything from the standard library except for comparison operators and constructors, e.g., `(::)` and `[]`.

### 3 Algebraic Data Types

Determine the smallest type `stuff` such that the following function is well-typed.

```
let do_thing (s : 'a stuff) (x : int) =  
  match s with  
  | Foo f -> f x  
  | Bar (y, z) -> (if y then y else y) :: z  
  | Baz l -> l.first (l.second (l.third = x))
```

## 4 Optional Binding

The programming language Swift has a feature called *optional binding*, which allows us to conditionally bind the inner value of an option. In this problem, we'll be looking at typing and semantic rules for a variant of this. We introduce **let?**-expressions, a version of **let**-expressions specialized to options. They have the following syntax:

$\langle \text{expr} \rangle ::= \text{let? } \langle \text{expr} \rangle = \langle \text{expr} \rangle \text{ in } \langle \text{expr} \rangle$

Make sure to read the following rules carefully. The new **let?**-expressions are similar to **let**-expressions, but not identical.

A. Consider the following typing rules.

$$\begin{array}{c}
 \frac{x : \tau \in \Gamma}{\Gamma \vdash x : \tau} \text{VAR} \qquad \frac{n \text{ is an integer literal}}{\Gamma \vdash n : \text{int}} \text{INTLIT} \qquad \frac{\Gamma \vdash e_1 : \text{int} \quad \Gamma \vdash e_2 : \text{int}}{\Gamma \vdash e_1 + e_2 : \text{int}} \text{ADDINT} \\
 \\
 \frac{}{\Gamma \vdash \text{None} : \tau \text{ option}} \text{NONE} \qquad \frac{\Gamma \vdash e : \tau}{\Gamma \vdash \text{Some } e : \tau \text{ option}} \text{SOME} \\
 \\
 \frac{\Gamma \vdash e_1 : \tau' \text{ option} \quad \Gamma, x : \tau' \vdash e_2 : \tau \text{ option}}{\Gamma \vdash \text{let? } x = e_1 \text{ in } e_2 : \tau \text{ option}} \text{LETOPT}
 \end{array}$$

Write a derivation of the following typing judgment.

$\{\} \vdash \text{let? } x = \text{Some } 2 \text{ in } \text{Some } (x + 2) : \text{int option}$

B. Consider the semantic rules. Note that we introduce an option value for the result of evaluating an option.

$$\begin{array}{c}
\frac{\text{n is an integer literal for } n}{n \Downarrow n} \text{INTLITE} \qquad \frac{e_1 \Downarrow v_1 \quad e_2 \Downarrow v_2 \quad v = v_1 + v_2}{e_1 + e_2 \Downarrow v} \text{ADDITE} \\
\\
\frac{}{\text{None} \Downarrow \text{None}} \text{NONE} \qquad \frac{e \Downarrow v}{\text{Some } e \Downarrow \text{Some}(v)} \text{NONE} \\
\\
\frac{e_1 \Downarrow \text{Some}(v_1) \quad e' = [v_1/x]e_2 \quad e' \Downarrow v}{\text{let? } x = e_1 \text{ in } e_2 \Downarrow v} \text{LETOPTSOME} \quad \frac{e_1 \Downarrow \text{None}}{\text{let? } x = e_1 \text{ in } e_2 \Downarrow \text{None}} \text{LETOPTNONE}
\end{array}$$

Determine the value  $v$  such that the following semantic judgment is derivable, and then write its derivation.

$$\text{let? } x = \text{Some } 2 \text{ in let? } y = \text{None in Some } (x + y) \Downarrow v$$