

Midterm Exam

CAS CS 320: Concepts of Programming Languages

February 19, 2026

Name: Nathan Mull

BUID: 12345678

- ▷ 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 misconduct).
- ▷ Unless stated otherwise, you should only need the rules provided **in the problem statement** for any derivation.
- ▷ 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.

Problem #	Points
1	6
2	4
3	4
4	14

This page is intentionally left blank.

1 OCaml Programming

(6 points) Implement the function `val nub : ('a * 'b) list -> ('a * 'b) list` so that `nub l` is the result of removing entries from `l` until every key is associated with at most one value. In the case that `l` has multiple entries with the same key, the *leftmost* one should be kept and the remaining should be removed. The order of the remaining elements should be maintained.

```
let _ = assert (nub [(1, "1"); (2, "2")] = [(1, "1"); (2, "2")])
let _ = assert (nub [(1, "1"); (2, "2"); (1, "3")] = [(1, "1"); (2, "2")])
let _ = assert (nub [(1, "1"); (2, "2"); (1, "3"); (1, "4")] = [(1, "1"); (2, "2")])
let _ = assert (nub [(2, "2"); (1, "3"); (1, "4"); (2, "5")] = [(2, "2"); (1, "3")])
```

You may use the function `List.rev` and basic constructors like `(::)` and `[]`, but you otherwise may not use any functions from the standard library.

Hint: First implement the function `val remove_key : 'a -> ('a * 'b) list -> ('a * 'b) list` so that `remove_key k l` is the result of removing *all* entries of `l` with the key `k`. For partial credit, implement `nub` assuming access to `remove_key`.

```
let rec nub l =
  match l with
  | [] -> []
  | (k,v)::es -> (k,v)::nub (remove_key k es)
```

```
let remove_key k l =
  let rec loop acc l =
    match l with
    | [] -> List.rev acc
    | (k',v')::es ->
      if k' = k
      then loop acc es
      else loop ((k',v')::acc) es
  in loop [] l
```

(Problem 1 Continued)

2 Polymorphism

(4 points) Determine an OCaml expression with the following polymorphic type:

```
('a -> 'b, 'c -> 'b) result -> ('a * 'c) -> 'b
```

Your expression may not include `assert false`. Recall that the type `('a, 'b) result` is defined as follows.

```
type ('a, 'b) result =  
  | Ok of 'a  
  | Error of 'b
```

```
fun r (x,y) ->  
  match r with  
  | Ok f -> f x  
  | Error g -> g y
```

3 Gap Buffers

(4 points) A *gap buffer* is data structure common in the implementation of text editors. It's like a list, but with an interface for moving and accessing a cursor location. Below is a toy OCaml implementation of a gap buffer.

```
type 'a cur = (* PART A *)
type 'a gbuf = (* PART A *)

let empty : 'a gbuf = { lft = []; rgt = End }

let get_cur (c : 'a cur) : 'a option =
  match c with
  | End -> None
  | Cur (x, _) -> Some x

let get (g : 'a gbuf) : 'a option = get_cur g.rgt

let insert (x : 'a) (g : 'a gbuf) : 'a gbuf = { lft = x :: g.lft; rgt = g.rgt }

let move_left (g : 'a gbuf) : 'a gbuf =
  match g.lft with
  | [] -> g
  | x :: xs ->
    let r =
      match g.rgt with
      | End -> []
      | Cur (y, ys) -> y :: ys
    in
    { lft = xs; rgt = Cur (x, r) }

let ex = get (move_left (move_left (insert 3 (insert 2 (insert 1 empty))))) (* Part B *)
```

A. Determine the types `'a cur` and `'a gbuf`, assuming all pattern matches above are exhaustive.

type 'a cur =
| End
| Cur of 'a * 'a list

type 'a gbuf = {
 lft : 'a list;
 rgt : 'a cur;
}

(Problem 3A Continued)

B. (2 point, Extra Credit) Determine the value of `ex`. Explain your answer.

Some 2

get returns whats under the cursor
if it's not at the end of the buffer.
insert pushes the cursor to the right.

`[]` $\Rightarrow$

`[1]` $\Rightarrow$

`[1, 2]` $\Rightarrow$

`[1, 2, 3]` $\Rightarrow$

`[1, 2, 3]` $\Rightarrow$ `[1, 2, 3]`

4 Options and Results, Formally

The following are the syntax, typing and semantic rules for options with a simple form of pattern matching.

Syntax: (note that the symbol ‘|’ in the last line is part of the expression syntax and is distinct from the symbol ‘|’ which is slightly taller and narrower)

$$\begin{aligned} \text{<ty>} &::= \text{<ty> option} \\ \text{<expr>} &::= \text{None} \mid \text{Some <expr>} \\ &\mid \text{match <expr> with None -> <expr> \mid Some <var> -> <expr>} \end{aligned}$$

Typing:

$$\begin{aligned} &\frac{}{\Gamma \vdash \text{None} : \tau \text{ option}} \text{NONE} & \frac{\Gamma \vdash e : \tau}{\Gamma \vdash \text{Some } e : \tau \text{ option}} \text{SOME} \\ &\frac{\Gamma \vdash e_1 : \tau_1 \text{ option} \quad \Gamma \vdash e_2 : \tau \quad \Gamma, x : \tau_1 \vdash e_3 : \tau}{\Gamma \vdash \text{match } e_1 \text{ with None -> } e_2 \mid \text{Some } x \text{ -> } e_3 : \tau} \text{MATCHOPT} \end{aligned}$$

Semantics: (note the None and Some(·) values)

$$\begin{aligned} &\frac{}{\text{None} \Downarrow \text{None}} \text{NONEE} & \frac{e \Downarrow v}{\text{Some } e \Downarrow \text{Some}(v)} \text{SOME E} \\ &\frac{e_1 \Downarrow \text{None} \quad e_2 \Downarrow v}{\text{match } e_1 \text{ with None -> } e_2 \mid \text{Some } x \text{ -> } e_3 \Downarrow v} \text{MATCHOPTNONE} \\ &\frac{e_1 \Downarrow \text{Some}(v_1) \quad e = [v_1/x]e_3 \quad e \Downarrow v}{\text{match } e_1 \text{ with None -> } e_2 \mid \text{Some } x \text{ -> } e_3 \Downarrow v} \text{MATCHOPTSOME} \end{aligned}$$

The problem continues on the following page.

- A. (5 point) Determine a context Γ and a type τ such that the following judgment is derivable. Also provide the derivation. You must label all rule instances in your derivation with the corresponding rule name.

$$\Gamma \vdash \text{match } x \text{ with None} \rightarrow b \mid \text{Some } y \rightarrow y = 2 : \tau$$

You will need the following rules for this derivation.

$$\frac{(x, \tau) \in \Gamma}{\Gamma \vdash x : \tau} \text{VAR} \quad \frac{n \text{ is an integer literal}}{\Gamma \vdash n : \text{int}} \text{INTLIT} \quad \frac{\Gamma \vdash e_1 : \tau \quad \Gamma \vdash e_2 : \tau}{\Gamma \vdash e_1 = e_2 : \text{bool}} \text{EQ}$$

$$\frac{\Gamma \vdash e_1 : \tau_1 \text{ option} \quad \Gamma \vdash e_2 : \tau \quad \Gamma, x : \tau_1 \vdash e_3 : \tau}{\Gamma \vdash \text{match } e_1 \text{ with None} \rightarrow e_2 \mid \text{Some } x \rightarrow e_3 : \tau} \text{MATCHOPT}$$

$$\frac{\frac{\frac{}{\{x:\text{int option}, b:\text{bool}\} \vdash x:\text{int option}} \text{(var)}}{\{x:\text{int option}, b:\text{bool}\} \vdash x:\text{int option}} \quad \frac{\frac{}{\{x:\text{int option}, b:\text{bool}\} \vdash b:\text{bool}} \text{(var)}}{\{x:\text{int option}, b:\text{bool}\} \vdash b:\text{bool}} \quad \frac{\frac{\frac{}{\{x:\text{int option}, b:\text{bool}, y:\text{int}\} \vdash y:\text{int}} \text{(var)}}{\{x:\text{int option}, b:\text{bool}, y:\text{int}\} \vdash y:\text{int}} \quad \frac{\frac{}{\{x:\text{int option}, b:\text{bool}, y:\text{int}\} \vdash z:\text{int}} \text{(intLit)}}{\{x:\text{int option}, b:\text{bool}, y:\text{int}\} \vdash z:\text{int}}}{\frac{\{x:\text{int option}, b:\text{bool}, y:\text{int}\} \vdash y=z:\text{bool}} \text{(eq)}}{\{x:\text{int option}, b:\text{bool}\} \vdash \text{match } x \text{ with None} \rightarrow b \mid \text{Some } y \rightarrow y = 2 : \text{bool}} \text{(matchOpt)}$$

- B. (4 points) Results are like options except that the analogous “none” variant carries data. The following are the some of the syntax, typing, and semantic rules for results.

Syntax:

$\langle \text{ty} \rangle ::= (\langle \text{ty} \rangle , \langle \text{ty} \rangle) \text{ result}$
 $\langle \text{expr} \rangle ::= \text{Ok } \langle \text{expr} \rangle \mid \text{Error } \langle \text{expr} \rangle$
 $\quad \mid \text{ match } \langle \text{expr} \rangle \text{ with Ok } \langle \text{var} \rangle \rightarrow \langle \text{expr} \rangle \mid \text{Error } \langle \text{var} \rangle \rightarrow \langle \text{expr} \rangle$

Typing:

$$\frac{\Gamma \vdash e : \tau_1}{\Gamma \vdash \text{Ok } e : (\tau_1 , \tau_2) \text{ result}} \text{OK} \qquad \frac{\Gamma \vdash e : \tau_2}{\Gamma \vdash \text{Error } e : (\tau_1 , \tau_2) \text{ result}} \text{ERROR}$$

Semantics: (note the $\text{Ok}(\cdot)$ and $\text{Error}(\cdot)$ values)

$$\frac{e \Downarrow v}{\text{Ok } e \Downarrow \text{Ok}(v)} \text{OKE} \qquad \frac{e \Downarrow v}{\text{Error } e \Downarrow \text{Error}(v)} \text{ERROR E}$$

In analogy with options, write the typing and semantic rules for pattern matching with results.

Hint: As with options, there should be three rules total.

$$\frac{\Gamma \vdash e_1 : (\tau_1, \tau_2) \text{ result} \quad \Gamma, x : \tau_1 \vdash e_2 : \tau \quad \Gamma, y : \tau_2 \vdash e_3 : \tau}{\Gamma \vdash \text{match } e_1 \text{ with Ok } x \rightarrow e_2 \mid \text{Error } y \rightarrow e_3 : \tau} \text{(matchRes)}$$

$$\frac{e_1 \Downarrow \text{Ok}(v_1) \quad e = [v_1 / x] e_2 \quad e \Downarrow v}{\text{match } e_1 \text{ with Ok } x \rightarrow e_2 \mid \text{Error } y \rightarrow e_3 \Downarrow v} \text{(matchResOk)}$$

$$\frac{e_1 \Downarrow \text{Error}(v_1) \quad e = [v_1 / y] e_3 \quad e \Downarrow v}{\text{match } e_1 \text{ with Ok } x \rightarrow e_2 \mid \text{Error } y \rightarrow e_3 \Downarrow v} \text{(matchResErr)}$$

- C. (5 point) Determine a value v such that the following judgment is derivable. Also provide the derivation. You must label all rule instances in your derivation with the corresponding rule name.

match Error 3 with Ok $x \rightarrow x + 1$ | Error $y \rightarrow y + 2 \Downarrow v$

In addition to the rules you determined in the previous part, you'll need some of the following rules for this derivation.

$$\frac{\text{n is an integer literal for } n}{n \Downarrow n} \text{INTLITE}$$

$$\frac{e_1 \Downarrow v_1 \quad e_2 \Downarrow v_2 \quad v = v_1 + v_2}{e_1 + e_2 \Downarrow v} \text{ADDINTE}$$

$$\frac{e \Downarrow v}{\text{Ok } e \Downarrow \text{Ok}(v)} \text{OKE}$$

$$\frac{e \Downarrow v}{\text{Error } e \Downarrow \text{Error}(v)} \text{ERROR}$$

$$\begin{aligned} [3/y](y+2) &= 3+2 \\ 3+2 &= 5 \end{aligned} \quad \left(\text{not necessary for full credit} \right)$$

$$\frac{\frac{3 \Downarrow 3 \quad (\text{intLite})}{\text{Error } 3 \Downarrow \text{Error}(3)} \quad (\text{Error}) \quad \frac{\frac{3 \Downarrow 3 \quad (\text{intLite}) \quad \frac{2 \Downarrow 2 \quad (\text{intLite})}{3+2 \Downarrow 5} \quad (\text{intAddE})}{\text{match Error } 3 \text{ with Ok } x \rightarrow x+1 \mid \text{Error } y \rightarrow y+2 \Downarrow 5} \quad (\text{matchRule})$$

This page is intentionally left blank.