CSE216
Foundations of Computer Science

Instructor: Zhoulai Fu

State University of New York, Korea

Some slides taken from Prof Clarkson CS3110. Thanks!
https://www.cs.cornell.edu/courses/cs3110/2014fa/lecture_notes.php

Midterm 2

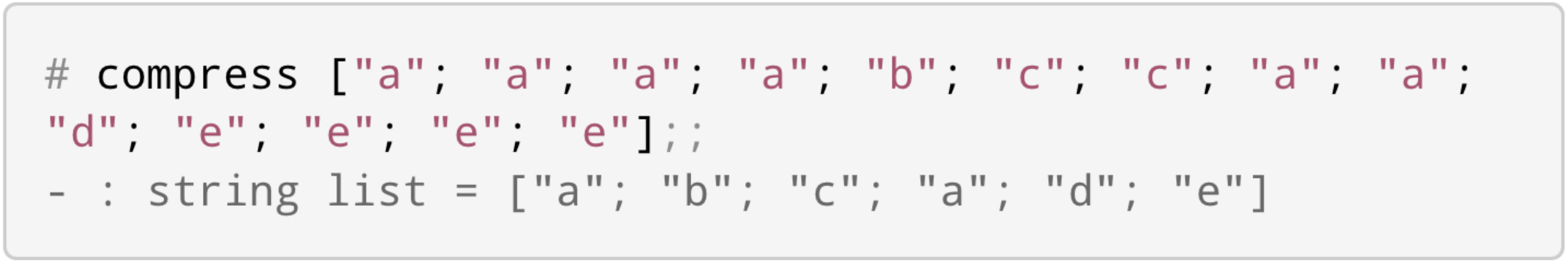
- Midterm 2 date: TBA. Immediately after our Ocaml part finishes, approximately 2 weeks later.
- Covering Lambda calculus + OCaml, not including things before

Plan

- A review question
- Pattern Matching in details

Question #8

- Eliminate consecutive duplicates of list elements.



```
# compress ["a"; "a"; "a"; "a"; "b"; "c"; "c"; "a"; "a";
"d"; "e"; "e"; "e"; "e"];;
- : string list = ["a"; "b"; "c"; "a"; "d"; "e"]
```

https://v2.ocaml.org/learn/tutorials/99problems.html

Pattern matching in details

Match expressions

- **Syntax**

```
match e with p1 -> e1 | p2 -> e2 | … | pn -> en
```

- **Evaluation:**
  - Evaluate `e` to a value `v`
  - If `pi` is the first pattern to match `v`, then evaluate `ei` to value `vi` and return `vi`
    - Note: pattern itself is **not** evaluated
  - Pattern *matches* value if it “looks like” the value
    - Pattern `Ci (x1 , … , xn)` matches value `Ci (v1 , … , vn)`
    - *Wildcard* pattern _ (i.e., underscore) matches any value

Typing

```
match e with p1 -> e1 | p2 -> e2 | … | pn -> en
```

- **Type-checking:**
  - If `e, p1..pn` have type `ta`
    and `e1..en` have type `tb`
    then entire match expression has type `tb`

Enhanced pattern Syntax

- Patterns can nest arbitrarily deep
  - (Just like expressions)
  - Easy-to-read, nested patterns can replace hard-to-read, nested **`match`** expressions

- Examples:
  - Pattern **`a::b::c::d`** matches all lists with >= 3 elements
  - Pattern **`a::b::c::[]`** matches all lists with 3 elements
  - Pattern **`((a,b),(c,d))::e`** matches all non-empty lists of pairs of pairs

example: zip 3 lists

```
let rec zip3 lists =
  match lists with
([],[],[]) -> []

|(hd1::tl1, hd2::tl2, hd3::tl3) ->
 (hd1, hd2, hd3)::zip3(tl1, tl2, tl3)

| _ -> raise (Failure “List length
mismatch”)
```

Exercise: unzip 3 lists

```
unzip3 : ('a * 'b * 'c) list -> 'a list * 'b list * 'c list
```

```
# unzip3 ([(3,6,8);(7,9,0)]) ;;
- : int list * int list * int list = ([3; 7], [6; 9], [8; 0])
```

Precise Definitions of Pattern Matching

Given a pattern $\mathbf{p}$ and a value $\mathbf{v}$, decide

– Does pattern match value?

– If so, what variable bindings are introduced?

Let's give an evaluation rule for each kind of pattern…

Precise Definitions of Pattern Matching (2)

- If **`p`** is a variable **`x`**, the match succeeds and **`x`** is bound to **`v`**
- If **`p`** is `_`, the match succeeds and no bindings are introduced
- If **`p`** is a constant *c*, the match succeeds if **`v`** is *c*. No bindings are introduced.

Precise Definitions of Pattern Matching (3)

- If **`p`** is **`C`**, the match succeeds if **`v`** is **`C`**. No bindings are introduced.
- If **`p`** is **`C p1`**, the match succeeds if **`v`** is **`C v1`** (i.e., the same constructor) and **`p1`** matches **`v1`**. The bindings are the bindings from the sub-match.

Precise Definitions of Pattern Matching (4)

- If **`p`** is **`(p1,…, pn)`** and **`v`** is **`(v1,…, vn)`**, the match succeeds if **`p1`** matches **`v1`**, and …, and **`pn`** matches **`vn`**. The bindings are the union of all bindings from the sub-matches.
  - The pattern **`(x1,…,xn)`** matches the tuple value **`(v1,…,vn)`**
- If **`p`** is **`{f1=p1; …; fn=pn}`** and **`v`** is **`{f1=v1; …; fn=vn}`**, the match succeeds if **`p1`** matches **`v1`**, and …, and **`pn`** matches **`vn`**. The bindings are the union of all bindings from the sub-matches.
  - (and fields can be reordered)
  - The pattern **`{f1=x1;…;fn=xn}`** matches the record value **`{f1=v1;…;fn=vn}`**

Exercise

- Using pattern matching, write three functions, one for each of the following properties. Your functions should return true if the input list has the property and false otherwise.

- 1. the list’s first element is “hello"

- 2. the list has exactly two or four elements; do not use the length function

- 3. the first two elements of the list are equal