CS 131 Discussion 1

Winter 2015

General Info

Professor Paul Eggert

Office Hours: Mondays 10:00–11:00 and Thursdays 13:30–14:30
Boelter 4532J

Email: eggert@cs.ucla.edu

Yun Lu (TA)

Office Hours (tentative): Mondays 14:30–16:30 at Boelter 2432

Email: yunalu@ucla.edu

Class website

http://www.cs.ucla.edu/classes/winter15/cs131/

Piazza

piazza.com/ucla/winter2016/cs131

Homework and project submission

CCLE

Grading

6 Homeworks (30%)

2 OCaml, 1 Prolog, 1 Java, 1 Scheme, 1 TBD

1 Project (10%)

Python

1 Midterm (20%)

Thursday, Feb 05 during lecture

1 Final (40%)

Monday, March 16 8:00am to 11:00am

OCaml Crash Course

OCaml Basic Types

(Based on TryOCaml: http://try.ocamlpro.com/)

Numbers

```
# 1+2;; (* Integer addition *)
- : int = 3
# 1. +. 2.;; (* Floating-point addition *)
- : float = 3.
```

String

```
# "Mary";;
- : string = "Mary"
```

Character

```
# 'a';;
- : char = 'a'
```

Lists and Tuples

List

```
# [ 42; 1; 55 ];;
- : int list = [42; 1; 55]
```

Tuple

```
# (1,"HELLO!");;
- : int * string = (1, "HELLO!")
```

List of tuples

```
# [1,'a';2,'b';3,'c'];;
- : (int * char) list = [(1, 'a'); (2, 'b'); (3, 'c')]
```

List operations

Reverse a list

```
# List.rev [1;2;3;4];;
- : int list = [4; 3; 2; 1]
```

Check if a list contains an element

```
# List.mem 2 [1;2;3];;
- : bool = true
```

Concatenate two lists

```
# []@[1;2];;
- : int list = [1; 2]
# [1;2]@[3];;
- : int list = [1; 2; 3]
```

For more list operations

http://caml.inria.fr/pub/docs/manual-ocaml/libref/List.html

If-Then-Else

Comparison operators are >, <, <=, =, !=

```
# if 1>2 then "1>2" else "1<=2";;
- : string = "1<=2"

# if 2=2 then "2=2" else "2!=2"
- : string = "2=2"
```

Cannot have just If-Then

```
# if 1<2 then true;;
Characters 12-16:
  if 1<2 then true;;
                ^^^^
Error: The variant type unit has no constructor true
```

Functions

Defining a function

```
# let add x y = x+y;;
val add : int -> int -> int = <fun>
```

Calling a function

```
# add 1 2;;
- : int = 3
```

Passing a function as an argument

```
# let foo f x = f x;;
val foo : ('a -> 'b) -> 'a -> 'b = <fun>
# foo List.rev [1;2;3];;
- : int list = [3; 2; 1]
```

Matching

Simple Example

```
# let string_of_int x = match x with
    | 0 -> "zero"
    | 1 -> "one"
    | 2 -> "two"
    | _ -> "many";;
val string_of_int : 'a -> string = <fun>
```

In general

```
match [EXPRESSION] with
| [PATTERN1] -> [VALUE 1]
| [PATTERN2] -> [VALUE 2]
...
```

Matching with Lists

Find the first element of the list

```
# let firstElem x = match x with
  | h::t -> Some h
  | _ -> None;;
val firstElem : 'a list -> 'a option = <fun>

# firstElem [1;2;3];;
- : int option = Some 1
```

Find the second element of the list

```
# let secondElem x = match x with
  | h1::h2::t -> Some h2
  | _ -> None;;
val secondElem : 'a list -> 'a option = <fun>

# secondElem [1;2;3];;
- : int option = Some 2
```

Matching with Lists and Tuples

Another Example

```
# let rec findThree x = match x with
   | (a,_)::t -> if a=3 then "Found 3" else (findThree t)
   | _ -> "Not found";;
val findThree : (int * 'a) list -> string = <fun>
```

Output

```
# findThree [1,'a'; 2, 'b'; 3, 'c'];;
- : string = "Found 3"
# findThree [1,'a'; 2, 'b'; 4, 'c'];;
- : string = "Not found"
```

OCaml Type Definitions

Simple Example

```
# type suit =
   | Club
   | Diamond
   | Heart
   | Spades;;
```

Usage

```
# let x = Club;;
val x : suit = Club
```

Another Example

```
# type foo =
   | Nothing
   | IntPair of int * int
   | IntList of int list;;
```

Usage

```
# IntList [1;2;3];;
- : foo = IntList [1; 2; 3]
```

Matching with Types

Example:

```
# type ('n, 't) symbol = | N of 'n | T of 't;;

# match N "Hi" with
   | T a -> a
   | N a -> a;;
- : string = "Hi"
```

Polymorphism

Example from Homework 1

```
# type ('nonterminal, 'terminal) symbol =
  | N of 'nonterminal
  | T of 'terminal;;

# N "Sentence";;
- : (string, 'a) symbol = N "Sentence"
# N "Sentence" = N "Sentence";;
- : bool = true
```

BUT: you cannot compare N “Sentence” with N 1 since they are different types

In general

```
type ('[TYPE1], '[TYPE2], …) [TYPE NAME] =
| [VALUE1]
| [VALUE2]
...
```

Homework 1

Warm-Up Exercises

Fixed Point : A point x such that f x = x

Computed Fixed Point : (with respects to an initial point x) A point (f^N x) such that (f^{N+1} x) = (f^N x)

Periodic Point : A point x such that (f^P x) = x, where P = period
A fixed point is a periodic point with P = 1

Computed Periodic Point : (with respects to an initial point x and period p)
A point (f^N x) such that (f^{N+P} x) = (f^N x)

More on Warm-Up Exercises

Sets: OCaml lists allow duplicates, even though a mathematical set does not

equal_sets: For example [1;2;3] and [1;2;1;3] are equal

proper_subset: For example [1;1;2] is a proper subset of [1;2;3] but not of [1;2]

set_diff: For example set_diff [1;2;2] [2]. We will allow both the output [1;2] and [1]

Context-Free Grammar

Terminology:

Non-terminal : A symbol which you can replace with other symbols
Terminal : A symbol which you cannot replace with other symbols
Grammar : A starting symbol, and a set of rules that describe what symbols can be derived from a non-terminal symbol

Example of a Grammar

Example:

Symbols : S, A, B, a, b
Non-terminals : S, A, B
Terminals : a, b
Starting Symbol : S

Rules:
S -> A
S -> B
A -> aA
A -> a
B -> bB
B -> b

Can be abbreviated as:
S -> A | B
A -> aA | a
B -> bB | b

How to Derive:
aaa

1. S
2. A (apply rule: S -> A)
3. aA (apply rule: A -> aA)
4. aaA (apply rule: A -> aA)
5. aaa (apply rule: A -> a)

Blind Alley Rules

- Any rule from which it is impossible to derive a string of terminals (a string of terminals includes the empty string)
- Example:

Symbols : S, A, B, a, b
Non-terminals : S, A, B
Terminals : a, b
Starting symbol : S

Rules:
S -> A | B
A -> A | aB | aA | a
B -> B

What are/is the blind alley rule(s)?
S -> B
A -> aB
B -> B

Blind Alley Rule, Ex. 2

Example from Homework 1

Symbols : Conversation, Sentence, Grunt, Snore, Shout, Quiet, “ZZZ”, “khrgh”, “aooogah”, “,”

Non-terminals : Conversation, Sentence, Grunt, Snore, Shout, Quiet

Terminals : “ZZZ”, “khrgh”, “aooogah”, “,”

Starting symbol : Sentence

Rules:

Conversation -> Snore
| Sentence “,” Conversation

Sentence -> Quiet | Grunt | Shout

Grunt -> “khrgh”

Shout -> “aooogah”

Quiet -> empty

Blind Alley Rules:

Conversation -> Snore

Conversation -> Sentence “,” Conversation