CSE216
Programming Abstraction

Instructor: Zhoulai Fu

State University of New York, Korea

Our paths have crossed for a reason...

Your career and us

- Our school wants to see you succeed
- You may hope to find a good job after graduation, or
- You may hope to find a good Master/Phd graduate school
- In either case, you need a resume that looks unique
- I am involved two Korea - US projects at SUNY-K, contributing to DARPA (US Defense)
    - 1. Programming Languages & Software Security E-BOSS Program: https://www.darpa.mil/program/enhanced-sbom-for-optimized-software-sustainment
    - 2. Programming Languages & Machine Learning TRACTOR Program: https://www.darpa.mil/program/translating-all-c-to-rust
- Together with you, we hope to produce great research work to make your resume shine
- Method 1. U3/U4 -> CSE487: Get credits, but can only do twice
- Method 2. Master -> RA: Get paid
- Prerequisite: Strong motivation, math/programming; CSE216 A-ish

Review

Lambda calculus

Small-1

- (λ x.x) a
- (λ x.y) a
- (λ x.xy) a
- (λ x. yx) a
- (λ x. xx) a
- (λ x. yy) a

Small-2

- (λ x.x) a b
- (λ x.y) a b
- (λ x.xy) a b
- (λ x. yx) a b
- (λ x. xx) a b
- (λ x. yy) a b

Small-3

- (λ x.x) λ a. b
- (λ x.y) λ a. b
- (λ x.xy) λ a. b
- (λ x. yx) λ a. b
- (λ x. xx) λ a. b
- (λ x. yy) λ a. b

Small-4

- (λ x.x) x
- (λ x.y) x
- (λ x.xy) x
- (λ x. yx) x
- (λ x. xx) x
- (λ x. yy) x

Small-5

- (λ x.x) x y
- (λ x.y) x y
- (λ x.xy) x y
- (λ x. yx) x y
- (λ x. xx) x y
- (λ x. yy) x y

Exercise: beta reduction

- (λz.z) (λz.z z) (λz.z q)

Exercise: beta reduction

- (λs.λq.s q q) (λa.a) b

Exercise: beta reduction

- (λs.λq.s q q) (λq.q) q

Exercise: beta reduction

- ((λs.s s) (λq.q)) (λq.q)

Exercise: beta reduction

- (λ x.λy.x) x y

Exercise: beta reduction

- (λ x. λ y. λ z. y (w y x)) λ s. λ z. z

Exercises:
Context-Free Grammar

Exercise 3

Given the grammar G with the following productions:

- $S \to aSb$
- $S \to \epsilon$

Determine the language $L(G)$ generated by G.

Exercise 4

Given the grammar G with the productions:

- $S \to aSa$
- $S \to bSb$
- $S \to \epsilon$

What is the language $L(G)$?

Exercise 5

1

Consider the grammar G defined as:

- $S \to aS$
- $S \to Sb$
- $S \to \epsilon$

Define the language $L(G)$.

2. Find a word that does not belong to L(G)

Exercise 6

Create a grammar that generates the language of all strings of the form:

- a language containing only the words "dog", "cat", and “fish”.

Exercise 7

Create a grammar that generates the language of all strings of the form:

- "a^n", where n ≥ 0.

Exercise 8

Create a grammar that generates the language of all strings of the form:

- "a^n b^m", where n, m ≥ 0.

Exercise 9

Create a grammar that generates the language of all strings of the form:

- "a^n b^n", where n ≥ 0.

Exercise 10

Create a grammar that generates the language of all strings of the form:

- all strings over {a, b} that start with 'a' and end with 'b'.

Ocaml

- Make sure you have no problem solving **#1-#10** in 99 problems: https://v2.ocaml.org/learn/tutorials/99problems.html

A harder one

Some programming languages (like Python) allow us to quickly *slice* a list based on two integers `i` and `j`, to return the sublist from index `i` (inclusive) and `j` (not inclusive). We want such a slicing function in OCaml as well.

Write a function **`slice`** as follows: given a list and two indices, `i` and `j`, extract the slice of the list containing the elements from the $\texttt{i}^{\text{th}}$ (inclusive) to the $\texttt{j}^{\text{th}}$ (not inclusive) positions in the original list.

```
# slice ["a";"b";"c";"d";"e";"f";"g";"h"] 2 6;;
- : string list = ["c"; "d"; "e"; "f"]
```

Invalid index arguments should be handled *gracefully*. For example,

```
# slice ["a";"b";"c";"d";"e";"f";"g";"h"] 3 2;;
- : string list = []
# slice ["a";"b";"c";"d";"e";"f";"g";"h"] 3 20;
- : string list = ["d";"e";"f";"g";"h"];
```

You do *not*, however, need to worry about handling negative indices.

Some Ocaml mock questions

(9) A *predicate* is a function that returns a boolean value. In OCaml, the type of such a function can be expressed as `'a -> bool`. Write a function `remove_if: 'a list -> ('a -> bool) -> 'a list`, which takes a list and a predicate, and removes all the elements that satisfy the predicate. Below is an example.

```
 # remove_if [1;2;3;4;5] (fun x -> x mod 2 = 1);;
- : int list = [2; 4]
```

(10) Find the last two elements of a list. `last_two: ' a list -> ('a * 'a) Option`. Below are two examples.

```
# last_two ["a"; "b"; "c"; "d"];;
- : (string * string) option = Some ("c", "d")
# last_two ["a"];;
- : (string * string) option = None
```

Problem 4. Ocaml Application (points = 15)

Given the definition of a binary tree:

```
type 'a btree =
  | Empty
  | Node of 'a * 'a btree * 'a btree
```

(1) Implement a function `height : 'a btree -> int` that takes a binary tree and returns its height. The height of a binary tree is defined as the length of the longest path from the root to a leaf. An empty tree has height 0, and a tree with one node has height 1. For example, given the following tree:

```
let mytree = Node (1, Node (2, Empty, Empty), Node (3, Node (4, Empty,
Empty), Empty))
```

The function `height mytree` should return `3`. The tree looks like:

```
  1
 / \
2   3
   /
  4
```

(2) ** Implement a function `is_balanced : 'a btree -> bool` that checks whether a binary tree is balanced. A binary tree is *balanced* if it is Empty, or the height of the two subtrees of every node differ less than or equal to one. For example, given the following tree:

```
let mytree2 = Node (1, Node (2, Node (3, Empty, Empty), Empty), Empty)
```

The function `is_balanced mytree` should return `true`, while `is_balanced mytree2` should return `false`. The tree looks like:

```
    1
   /
  2
 /
3
```

You can use the built-in function `abs` in this question.