CS 131 Discussion 3

Winter 2015

Announcements

- **Homework 2**

  Due Monday, Jan 26 at 23:55

Recursive Types in OCaml

(From https://ocaml.org/learn/tutorials/data_types_and_matching.html)

Definition

```
# type 'a binary_tree =
    | Leaf of 'a
    | Tree of 'a binary_tree * 'a binary_tree;;
```

```
type 'a binary_tree = Leaf of 'a | Tree of 'a
binary_tree * 'a binary_tree
```

Examples

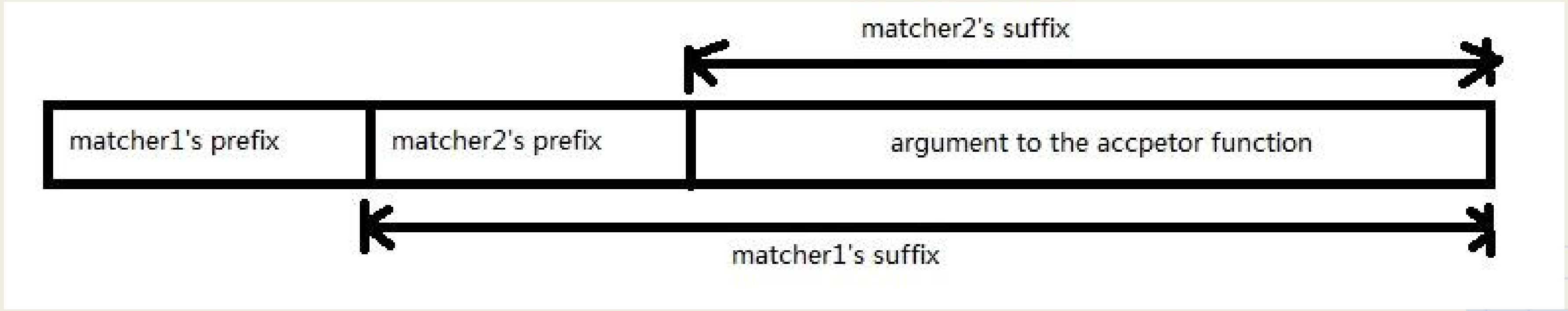
- `Leaf 3`
- `Tree (Leaf 3, Leaf 4)`
- `Tree (Tree (Leaf 3, Leaf 4), Leaf 5)`

-
  ```
  Tree (Tree (Leaf 3, Leaf 4),
        Tree (Tree (Leaf 3, Leaf 4), Leaf 5))
  ```

More Homework 2

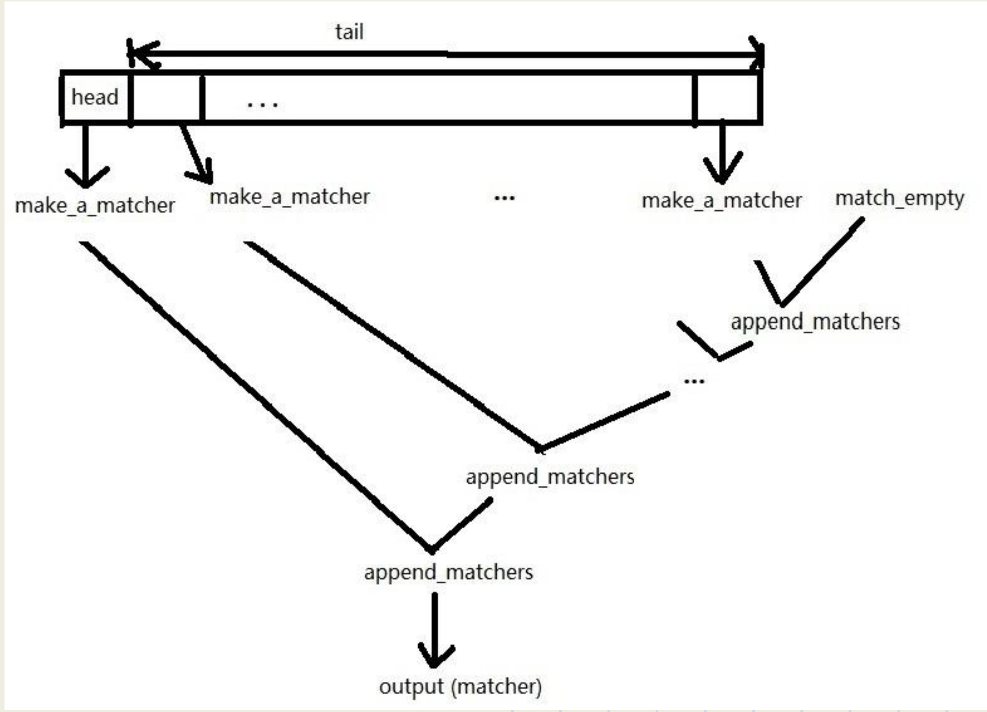
Appending matchers

```
# let append_matchers matcher1 matcher2 frag accept =
  matcher1 frag (fun frag1 -> matcher2 frag1 accept)
```



Making a list of Appended Matchers

```
let make_appended_matchers
        make_a_matcher ls =
  let rec mams = function
    | [] -> match_empty
    | head::tail ->
        append_matchers
             (make_a_matcher head)
             (mams tail)
   in mams ls
```



Match Star

Star (Kleene Star): `[STRING]`* means zero or more of `[STRING]`.
eg. `(ab)`*`c` = {`c, abc, ababc, abababc, …`}

<u>Example:</u>

matcher: a prefix is a matching prefix iff it is equal to **`ab`**
acceptor: accepts fragments that are equal to **`c`**

```
(match_star matcher acceptor)
```
creates a matcher that matches with **`(ab)`*****`c`**

Idea

Let **m** be the original matcher we call match_star with

1. Let the prefix be empty, and suffix be whole list
2. Call acceptor on suffix
3. If acceptor accepts, then return whatever acceptor returns
4. Else, find the next non-empty prefix of m. If no such prefix, return None
5. Go to 2

```
let rec match_star matcher frag accept =
  match accept frag with (*First try empty prefix*)
    | None ->
    matcher frag
        (fun frag1 ->
            if frag == frag1 (*get non-empty prefix only*)
            then None
            (*try another suffix*)
            else match_star matcher frag1 accept)
    | ok -> ok
```

Side note about ==

```
# let x = [1;2];;
val x : int list = [1; 2]
# let y = [1;2];;
val y : int list = [1; 2]
# x == y;;
- : bool = false
# x = y;;
- : bool = true
```

“Eager” Match

- Notice match_star tries to match the shortest number of repetitions first
  eg. Suppose a matching prefix is $(ab)^*$
  For ababab, we match empty string first, then ab, then abab, then ababab

- This is called a “**lazy**” match

- An “**eager**” match matches the longest possible match first.
  eg. for ababab, we match ababab first, then abab, then ab, then empty

“Eager” version of match_star

```
let rec eager_star matcher frag accept =
    match (matcher frag
            (fun frag1 ->
            if frag == frag1 then None  (*try to get the next non-empty prefix*)
            else
                (*is this the longest matching prefix?*)
                match (matcher frag1 accept_all) with
                | None -> (accept frag1)
                | ok -> match (eager_star matcher frag1 accept) with
                        | None -> None
                        | ok1 -> ok1 ))
    with
        | None -> accept frag (*try the empty prefix last*)
        | ok2 -> ok2
```