CS 131 Discussion 2

Winter 2015

Announcements

- **My office hours**
  Mondays 14:30–16:30 at Boelter 2432
- **Homework 1**
  Due Thursday, Jan 15 at 23:55
- **Please ask technical questions on Piazza**
  piazza.com/ucla/winter2016/cs131
- **Homework 2**
  Due Monday, Jan 26 at 23:55

Homework 2

A (Slightly) Different Representation of a Grammar

Recall Homework 1:

<u>Grammar</u> : A starting symbol, and a set of rules that describe what symbols can be derived from a non-terminal symbol

Homework 1:

```
(Sentence,
    [...;
    (Sentence, [N Quiet]);
    (Sentence, [N Grunt]);
    (Sentence, [N Shout);
    ...])
```

Homework 2:

```
(Sentence,
    function
        | …
        | Sentence -> [[N Quiet];
                        [N Grunt];
                        [N Shout]]
        | …)
```

Production function

Warm-Up Exercise

- Convert Homework 1- style grammar to Homework 2-style grammar
  - **(convert_grammar aksub_grammar)** should return a Homework 2-style grammar that is equivalent to aksub_grammar

- Test on the grammars in Homework 1

Derivations

Phrase/Fragment : A list of terminal symbols
Derivation : A list of rules that describe how to derive a phrase from a nonterminal symbol (the “starting symbol”)

Derivation of 3+4

```
[Expr, [N Term; N Binop; N Expr];
 Term, [N Num];
 Num, [T "3"];
 Binop, [T "+"];
 Expr, [N Term];
 Term, [N Num];
 Num, [T "4"]]
```

Matching Prefix

Prefix : [], [1], [1;2], [1;2;3] are prefixes of [1;2;3] (not necessarily in this order)
Suffix : [], [3], [2;3], [1;2;3] are suffixes of [1;2;3] (not necessarily in this order)
Matching Prefix: A prefix of a fragment, for which there exists a derivation

Example:
Find all matching prefixes of **3+$2-6** in order

Answer:
First matching prefix: 3+$2-6
Second matching prefix: 3+$2
Third matching prefix: 3

Acceptor

- A function which takes a **rule list** (generally a derivation) and a **list of symbols** (generally a suffix of a fragment) and returns:
  - None, if the acceptor rejects
  - Some x
    - x generally has the form of (ruleList, listOfSymbols)
    - Generally, ruleList and listOfSymbols will be the same as the arguments, but not necessarily

Example:

```
# let accept_empty_suffix derivation = function
  | [] -> Some (derivation, [])
  | _ -> None;;
# accept_empty_suffix derivation ["RULE1"; "RULE2"] [];;
- : (string list * 'a list) option = Some (["RULE1"; "RULE2"], [])
```

Summary so far

Phrase/Fragment : A list of terminal symbols

Derivation : A list of rules that describe how to derive a phrase from a nonterminal symbol (the “starting symbol”)

Prefix : [], [1], [1;2], [1;2;3] are prefixes of [1;2;3] (not necessarily in this order)

Suffix : [], [3], [2;3], [1;2;3] are suffixes of [1;2;3] (not necessarily in this order)

Matching Prefix : A prefix of a fragment, for which there exists a derivation

Acceptor : A function which takes a **rule list** (generally a derivation) and a **list of symbols** (generally a suffix of another fragment) and returns **None** (if reject) or **Some x**

Main Problem for Homework 2

- Write the function “parse_prefix”:
  (parse_prefix **grammar acceptor fragment**)

Here’s what the function does:

1. Look for the next matching prefix
2. If matching prefix not found, return None
3. Else, call acceptor with the suffix, and the derivation of the prefix
4. If acceptor returns None, go back to 1.
5. Else, return whatever the acceptor returns

Example:
(parse_prefix
awkish_grammar accept_all [“9+2”])

Outputs:

```
Some ( [Expr, [N Term; N Binop; N Expr];
        Term, [N Num];
        Num, [T "9"];
        Binop, [T "+"];
        Expr, [N Term];
        Term, [N Num];
        Num, [T "2"]],
      [])
```

Another Example

(parse_prefix awkish_grammar accept_all ["9"; "+"; "$"; "1"; "+"])

Outputs:

```
Some
   ([(Expr, [N Term; N Binop; N Expr]); (Term, [N Num]); (Num, [T "9"]);
     (Binop, [T "+"]); (Expr, [N Term]); (Term, [N Lvalue]);
     (Lvalue, [T "$"; N Expr]); (Expr, [N Term]); (Term, [N Num]);
     (Num, [T "1"])],
    ["+"])
```

Matchers

See the “Hint” section of http://www.cs.ucla.edu/classes/fall06/cs131/hw/hw2.html

Matcher : a function that takes a **fragment** and an **acceptor** and

1. Find next matching prefix
2. If no prefix found, return None, else
3. Call the acceptor with the suffix
4. If acceptor returns None, go back to 1.,
   else return whatever acceptor returned

Example:

```
let match_empty frag accept = accept frag;;
match_empty [1;2;3] (fun x -> x);;
```

Another Matcher

```
# let rec match_junk k frag accept =
  match accept frag with
  | None ->
    (if k = 0 then None
    else match frag with
        | [] -> None
        | _::tail -> match_junk (k-1) tail accept)
  | ok -> ok;;


# match_junk 1 [1;2;3] (function | [3] -> Some "Accepted" | _ -> None);;
# match_junk 2 [1;2;3] (function | [3] -> Some "Accepted" | _ -> None);;
```

- match_junk matches any prefix of size k or less

Complicated Example

```
# let rec make_or_matcher make_a_matcher = function
    | [] -> (fun frag accept -> None)
    | head::tail ->
          let head_matcher = make_a_matcher head
          and tail_matcher = make_or_matcher make_a_matcher tail
          in (fun frag accept ->
                let ormatch = head_matcher frag accept
                        in match ormatch with
                             | None -> tail_matcher frag accept
                             | _ -> ormatch);;

  val make_or_matcher :   ('a -> 'b -> 'c -> 'd option) -> 'a list -> 'b
  -> 'c -> 'd option = <fun>
```

```
# let m1 = make_or_matcher (fun h -> h)
                            [(fun frag accept -> accept frag)];;
val m1 : '_a -> ('_a -> '_b option) -> '_b option = <fun>

# m1 [1;2;3] (function | [1;2;3] -> Some "Accepted" | _ -> None);;

# m1 [1;2;3] (fun frag -> match frag with
                                [2;3] -> Some "Accept"
                                | _ -> None);;
```

```
# let m2 = make_or_matcher
               (fun h -> h)
               [(fun frag accept -> accept frag);
                (fun frag accept -> match frag with
                                     h::t -> accept t
                                    | _ -> None)];;
val m2 : '_a list -> ('_a list -> '_b option) -> '_b option = <fun>

# m2 [1;2;3] (fun frag -> match frag with
                                [2;3] -> Some "Accept"
                                | _ -> None);;
```