Lab 6 Writeup

Michael Gohde

April 21, 2018

1 Overview

This document contains our responses to the questions in the Lab 6 writeup.

2 Responses to questions

1. *Do the survey* Done
2. *Continuations warm-up* Done
3. *Recursive Descent Parsing Problems*
   (a) *"Give a refactored version of the re grammar from Figure 1 that eliminates ambiguity in BNF.*

```
re::=atom | union | intersect | concat | not | star
union::=re '|' re
intersect::=re & re
concat::=re re
not::=~re
star::= re star
atom::= . | c | # | !
```

   (b) *Explain why a recursive descent parser following your grammar with left recursion would go into an infinite loop.* The given grammar allows for the elements of 're' to call 're'. This can create a circular condition in which an an element includes a 're' case, which includes that element, which includes a 're' case, etc.

(c) *Give a refactored version of the 're' grammar that replaces left associative binary operators with n-ary version using EBNF.*

```
re::=union
union::=intersect{'|' intersect}
intersect::=concat * intersect
concat::=not concat
not::=~> not | star
star::=atom {'*' | + | '?'}
atom::= . | c | # | ! | '(' re ')'
```

(d) *Give typing and small step operational semantic rules for regular expression literals and regular expression tests. Clearly and concisely explain how your rules enforce the constraints given above and any additional decisions made.*

The rules given enforce the fact that e1 will eventually reduce to a string (since other types cannot be searched this way) in addition to the fact that e2 reduces to an object of type RegExp. If neither constraint is met, then the rules will not apply and an error should be thrown.

The type rules specify that a regular expression will return a boolean.

```
TypeRegExp
T |- e1: string T |- e2: RegExp
-------------------------------
e1 RegExp e2 -> b: boolean

DoRegExp
b'=str1 RegExp r1
--------------------
str1 RegExp r1 -> b'

SearchRegExp1
e2 -> e2'
----------------------------
e1 RegExp e2 -> e1 RegExp e2'

SearchRegExp2
```

```
e1 -> e1', e2 RegExp
----------------------------
e1 RegExp e2 -> e1' RegExp e2
```