Haskell Intro for Regular Expressions

Today

- Get started with Haskell
- Use Haskell to recognize a regular expression

Tonight

- PA1 is due!
- HW1 will be posted.

Haskell Input and Output

```
-- Main0.hs
--
-- Null compiler that just snarfs the input and spits it out.
--
-- compilation:
--      ghc --make -O2 Main0.hs -o mjc
--
-- usage:
--      ./mjc < infile
--      ./mjc < infile > outfile
--

module Main where

main = do
  file_as_str <- getContents
  print file_as_str
```

Regular Expression Example in Haskell

abba example in Haskell

- Play with the REPL (read eval print loop), the interpreter

Modify the finite state machine slightly.

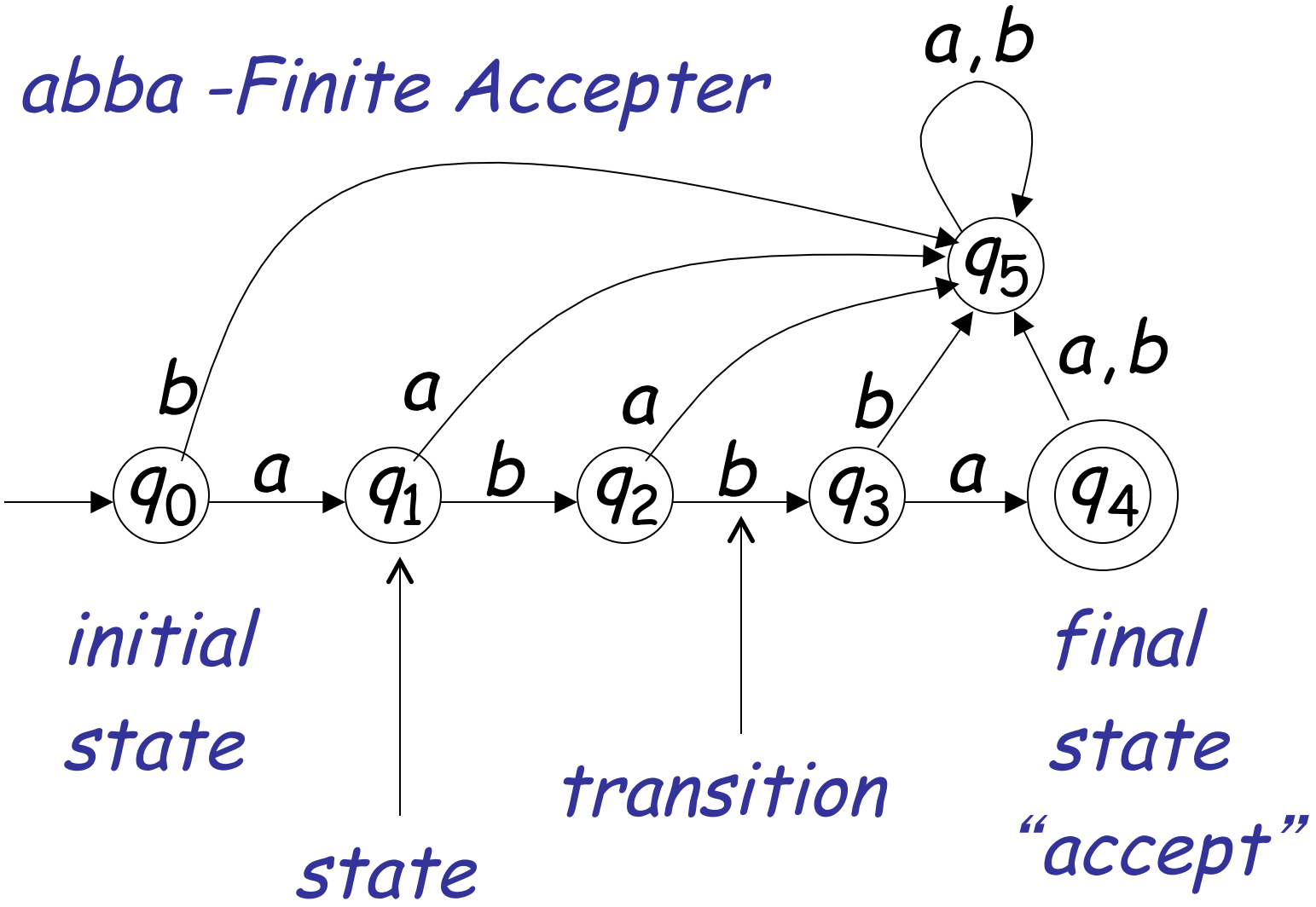
- How does the Haskell table code change?
- What regular expression is that equivalent to?

Will post source code. Also try out Kush's examples in subversion and Haskell recitation.

Good description of Haskell I/O in

- Chapter 7 of Real World Haskell book.
- Also really like Bartosz Milewski's Basics of Haskell.

State Transition Graph



Recursive Definition for Specifying Regular Expressions

Primitive regular expressions: $\varnothing,\ \varepsilon,\ \alpha$

where $\alpha \in \Sigma$, *some alphabet*

Given regular expressions r_1 *and* r_2

$r_1 \mid r_2$

$r_1\ r_2$

$r_1\ *$

(r_1)

} *Are regular expressions*

Complications

1. "1234" is an NUMBER but what about the "123" in "1234" or the "23", etc. Also, the scanner must recognize many tokens, not one, only stopping at end of file.

2. "if" is a keyword or reserved word IF, but "if" is also defined by the reg. exp. for identifier ID. We want to recognize IF.

3. We want to discard white space and comments.

4. "123" is a NUMBER but so is "235" and so is "0", just as "a" is an ID and so is "bcd", we want to recognize a token, but add attributes to it.

Complications 1

1. "1234" is an NUMBER but what about the "123" in "1234" or the "23", etc. Also, the scanner must recognize many tokens, not one, only stopping at end of file. So:

recognize the largest string defined by some regular expression,

only stop getting more input if there is no more match. This introduces the need to reconsider a character, as it is the first of the next token

e.g. *fname(a,bcd);*

would be scanned as

ID OPEN ID COMMA ID CLOSE SEMI EOF

scanning *fname* would consume (, which would be put back and then recognized as OPEN

Complication 2

2. "if" is a keyword or reserved word IF, but "if" is also defined by the reg. exp. for identifier ID, we want to recognize IF, so

Have some way of determining which token (IF or ID) is recognized.

This can be done using priority, e.g. in scanner generators an earlier definition has a higher priority than a later one.

By putting the definition for IF before the definition for ID in the input for the scanner generator, we get the desired result.

What about the string "ifyouleavemenow"?

Complication 3

3. we want to discard white space and comments and not bother the parser with these. So:

in scanner generators, we can

specify, using a regular expression, white space e.g. [\t\n]

and return no token, i.e. move to the next

specify comments using a (NASTY) regular expression and again return no token, move to the next

Complication 4

4. "123" is a NUMBER but so is "235" and so is "0", just as "a" is an ID and so is "bcd", we want to recognize a token, but add attributes to it. So,

Scanners return Symbols, not tokens.

A Symbol is a (token, tokenValue) pair,

e.g. (NUMBER,123) or (ID,"a").

Often more information is added to a symbol, e.g. line number and position (as we will do in MeggyJava)