CS536 Lecture 5

Tuesday 3 February 2015

Reminders:

- P1 Part 2 due last night, late days in effect.
- HW1 due tonight, HW2 assigned.
- Reading up

Last class:

- Regular Expressions
- Attaching Actions to States

Today:

- Full Scanner
- JLex

Actions: Review

FSMs only check for membership. A scanner needs to:

- Recognize a stream of many different tokens using the longest match.
- Know what *kind* of token was matched

Types of actions:

- Return a token
- Report an error
- Put a character “back”

Example: Pascal identifiers

eof symbol

Our first (full) Scanner

Consider a language consisting of two simple statements:

- Assignments: `ID = expr`
- Increments: `ID += expr`

where `expr` is of the form

- `ID + ID`
- `ID ^ ID`
- `ID < ID`
- `ID <= ID`

and identifiers ID follow C/C++ conventions.

Tokens:

Token	Regular Expression
ASSIGN	“=”
INCR	“+=”
PLUS	“+”
EXP	“^”
LT	“<”
LEQ	“<=”
ID	(letter|'_')(letter|digit|'_')*

Full DFA

State transition table

	=	+	^	<	_	letter	digit	EOF	other
s_0									
A									
B									
C									

Scanner pseudo-code

```
do
        read character
        update state and perform action if any
        if action was to return a token
                go back to start state
while
        not (stuck or EOF)
```

Scanner *generators*

Lex: Unix scanner generator, outputs C code for scanner

Flex: Faster Lex

JLex: Java version of Lex (we'll use this)

JLex:

Declarative, not procedural

Input: JLex specification (set of regular expressions + associated actions) in a file `<xxx>.jlex` .

Output: Java source code for the scanner. `<xxx>.jlex.java`

On compilation, this produces `YYLex.class`

JLex specification format

3 sections:

- User code section
- Directives
- Regular Expressions + Actions (rules section)

  separated by `%%`

Example

Rules section:

- Format is `<regex> { code }` where `<regex>` is a regular expression for a single token. Can use macros defined earlier (surround with curly braces).
- Regex operators: `| * + ? ( )`
- Characters represent themselves (except special chars e.g. `\n`, `\t`, `^`, `$`) Characters inside " " represent themselves (except \")
- Character class operators: `-` (range), `^` (not), `\` (escape)
- `.` (dot) matches anything

Example

Compiling and Running

.jlex files

(set the CLASSPATH variable)

```
java Jlex.Main <xxx>.jlex
javac <xxx>.jlex.java
```

Driver file

```
javac Driver.java
java Driver sample_input.txt
```

Example

Adding more functionality