Regular Expressions in Java
Grammars

March 26–27, 2012
CS 60: Principles of Computer Science

Assignment 7 (Quantities) due tonight.
Assignment 8 (Full Unicalc) due next Monday.
Regular Expression Ingredients

Regular expressions are a formal way of describing simple patterns. A regular expression can be:

✓ The empty string (sometimes written $\epsilon$ or $\lambda$)
✓ A single character (e.g., $a$ or $0$)
✓ Concatenation: $r_1 r_2$
✓ Alternative: $r_1 | r_2$
✓ Repetition (“Kleene Star”): $r_1^*$

In practice we use lots of abbreviations, e.g.,

\[
[a-e] := (a|b|c|d|e) \\
r? := r | \epsilon \\
r^+ := r r^*
\]
Application: Regular Expressions and Search

Unix’s `egrep` command does line-by-line search for text matching a regular expression.

- `egrep 'hh' /usr/share/dict/words`
- `egrep 'y.*y' /usr/share/dict/words`
- `egrep '(xq|hq)' /usr/share/dict/words`
- `egrep '^y.*y$' /usr/share/dict/words`
I have a Racket implementation of unicalc, with an extensive database:

- `(list 'km (make-QL 1.0 '(kilo\text{meter}) '()))`
- `(list 'kph (make-QL 1.0 '(kilo\text{meter}) '(hour)))`
- `(list 'kwh (make-QL 1.0 '(hour kilowatt) '()))`
- `(list 'l (make-QL 1.0 '(liter) '()))`

I wanted Java code to construct a database:

```java
db.put("km", new Quantity(1.0, 0.0, Arrays.asList("kilo\text{meter}"), Collections.<String>emptyList()));
```

```java
db.put("kph", new Quantity(1.0, 0.0, Arrays.asList("kilo\text{meter}"), Arrays.asList("hour")));
```

```java
db.put("kwh", new Quantity(1.0, 0.0, Arrays.asList("hour", "kilowatt"), Collections.<String>emptyList()));
```

```java
db.put("l", new Quantity(1.0, 0.0, Arrays.asList("liter"), Collections.<String>emptyList()));
```
My Plan

Write Java code that loops over the lines of a file. For each line:

- Search for occurrences of the Racket pattern
  \[(\text{list } '\text{joule} \ (\text{make-QL 1.0 '}(\text{kg m m}) ' (s s)))\]

- For each occurrence, extract the four important parts, and generate appropriate Java code, e.g.,
  - Quotation marks and commas
  - new Quantity,
  - Calls to `db.put` and `Arrays.asList`
READING LINES FROM A FILE

```java
import java.util.Scanner;
import java.io.FileInputStream;

try {
    FileInputStream fin = new FileInputStream(fileName);
    Scanner scan = new Scanner(fin);

    // Loop over the lines
    while ( scan.hasNextLine() ) {
        String line = scan.nextLine();
        ...
    }
}

} catch (IOException e) {
    System.out.println("Could not open " + filename)
}
```
## Looping Over Pattern-Matches in a String

// importing java.util.* isn't enough
import java.util.regex.Pattern;
import java.util.regex.Matcher;

Pattern p = Pattern.compile("[0-9]+"); // Only once
...

// To loop over all matches of Pattern p in one string:
// Step 1: Say which string we want to search
Matcher m = p.matcher(myString);

// Step 2: Loop over all matches.
while (m.find()) {
    String matchedSubstring = m.group();
    ...
}
Extracting Subparts of a Match

When you find a match (`m.find()`), rather than asking for the entire matched substring

```java
m.group();
```

you can ask for the subpart matched by the $k$-th parenthesized part of the regular expression

```java
m.group(k)
```

```java
Pattern p2 = Pattern.compile("([A-Za-z]+) : ([0-9]+) ");
Matcher m2 = p2.matcher("Student Pat : 42 units");
m2.find();
String match = m2.group();  // "Pat : 42"
String name = m2.group(1);  // "Pat"
String num = m2.group(2);   // "42"
```
BACK TO OUR EXAMPLE

Pattern linePat = Pattern.compile(???

...  
// Get ready to start searching this string
Matcher m = linePat.matcher(line);

// Loop over all pattern matches (database entries) in
while (m.find()) {
    String unitName = m.group(1);
    String valueString = m.group(2);
    String numeratorSymbols = m.group(3);
    String denominatorSymbols = m.group(4);

    // ... print a line of Java...
}

//}
**Exercise: Constructing A Regular Expression**

What should my regular expression be, to match an entire Racket line with the correct 4 parts parenthesized?

\[
\begin{align*}
\text{(list 'joule (make-QL 1.0 '([kg m m] [s s])))} \\
\text{(list 'atomic_mass_unit (make-QL 1.0 '([dalton] [()])))}
\end{align*}
\]

**Recall:**
- \(| = \text{“or,”} \)
- \(\cdots* = \text{“0 or more,”} \)
- \(\cdots+ = \text{“1 or more,”} \)
- \(. = \text{“any character except \n”} \)
- \([\text{abc}] = \text{“a or b or c,”} \)
- \([^{\text{abc}}] = \text{“any single character except a or b or c”} \)

Also in Java:
- \(\d = \text{“any digit,”} \)
- \(\s = \text{“any whitespace character (space, tab, ...),”} \)
- \(\S = \text{any “word” character (letter, number, underscore)} \)

See [http://docs.oracle.com/javase/6/docs/api/java/util/regex/Pattern.html](http://docs.oracle.com/javase/6/docs/api/java/util/regex/Pattern.html) for more.
Grammars and Parsing

March 26–27, 2012
CS 60: Principles of Computer Science
TRADITIONAL LEXING AND PARSING

\[(x - 32) \geq 7 \times y2\]
**SPECIFYING SYNTAX VIA CFGs**

A *context-free grammar* is a set of rules for producing a set of strings (a *language*).

\[
S \rightarrow V + S \mid V \\
V \rightarrow 0 \mid 1 \mid 2 \mid \cdots \mid 9
\]

**Ingredients:**

✓ Nonterminals: \( S, V \)
✓ Terminals: \(+, 0, 1, 2, \ldots, 9\)
✓ Production rules: (see above)
✓ Where to start: \( S \)

Show how to produce 4 starting from \( S \).
Show how to produce 4 + 5 starting from \( S \).
What other strings can we produce?
Using Structure to Clarify Meaning

He gave her cat food.
**Parse Trees**

The *parse tree* of a string makes explicit how a string was produced:

- ✓ Root is the start symbol
- ✓ When we apply a rule, items on the right-hand-side become children

Parse trees for 4 and 4+5?

\[
S \rightarrow V + S \mid V \\
V \rightarrow 0 \mid 1 \mid 2 \mid \cdots \mid 9
\]
**Parse Trees**

Show the parse tree for $9 - 3 + 2$

\[
S \rightarrow V + S \mid V - S \mid V \\
V \rightarrow 0 \mid 1 \mid 2 \mid \cdots \mid 9
\]