Parsing Output

11-7-2011

Opening Discussion

- Minute essay comments:
  - Parsing HTML with a CF grammar.

Example CF Grammar

- Here is a CF grammar for math expressions:
  - expr ::= term { “+” term | “-” term }
  - term ::= factor { “*” factor | “/” factor }
  - factor ::= floatingPointNumber | “(“ expr “)”
- Use {} for 0 or more and [] for 0 or 1.
- Lots of languages here:
  - http://www.antlr.org/grammar/list

Scala Parsers

- import scala.util.parsing.combinator._
- class Arith extends JavaTokenParsers {
  - def expr:Parser[Any] = term~rep(“+”~term | “-”~term)
  - def term:Parser[Any] = factor~rep(“*”~factor | “/”~factor)
  - def factor:Parser[Any] = floatingPointNumber | “(“~expr~”)”
- }

Conversion Rules

- Put in a class that extends one of the Parsers.
  - Productions become methods.
  - Results are Parsers. Next class we'll see how to make it more specific than Any.
  - Consecutive symbols are adjoined with ~.
  - The {...} is replaced with rep(...).
  - The […] is replaced with opt(...).

Using the Parser

- Call parseAll or parse on your class.
- Takes two arguments:
  - First argument is the parser to use.
  - Second argument is the string to parse.
- Let's code this all up and see it in action.

Default Parser Output

- Strings match themselves.
- RegEx and tokens give strings.
- P~Q gives back ~(p,q), where p and q are the matches of P and Q.
- P | Q gives either p or q.
- rep(P) or repsep(P,seperator) give a list of p values.
- opt(P) gives an Option, either Some(p) or None.

Specifying Output

- You can override the default of P by using P ^^ f. The f is a function (or partial function) that takes the normal output of P.
- The output you get is f(p).
- Example uses:
  - floatingPointNumber ^^ (_.toDouble)
  - “true” ^^ (x=>true)
  - “(“~ident~”,”~ident~”)” ^^ { case “(“~i1~”,”~i2~”)” => (i1,i2) }

Ignoring Parts of the Parse

- In something like the last example shown, there are strings that are part of the parse that really don't impact the result.
- When you have this type of situation you can use ~> or <~ instead of just ~. The parse result will only include what the arrow points to.
  - “(“~>ident~”,”~ident<~”)” ^^ { case i1~”,”~i2 => (i1,i2) }

Our Code

- Let's work on putting this type of functionality in our formula code.
- We want to parse to a tree similar to what we produced with the recursive parser we wrote ourselves.
- With that we can make this alternate code functional.

Minute Essay

- What questions do you have about parsers, regex, or grammars?
- Next class we do spatial trees.
- IcP #7 is next class.