Class 07: Regular Expressions

Held: Friday, 9 September 2011

Summary: We consider regular expressions, a popular mechanism for describing sets of strings. Regular expressions are often used to define the tokens in a language.

Related Pages:

- EBoard.
- Lab: Regular Expressions.
- Reading: Aho et al., 3.3-3.4.

Notes:

- Reading for Monday: Aho et al., 3.6.
- Assignment for Wednesday: Tokenizing Pascal.
- Today’s lexing code should be in Examples/LA.
- EC for Robotics talk next Thursday.
- Assignment 1 returned. In order to receive your given grade, you must write a one- or two-paragraph summary of my comments. If you got an F on the assignment because you turned in non-compiling or non-working code, you may redo it for potential regrading (cap of C).

Overview:

- Lexing Example, Continued.
- Regular Expressions.
- Simplifying Regular Expressions.
- Common Shorthands.
- Sample Regular Expressions.
- Limitations.
- Using Regular Expressions for Describing Tokens.
- Lab.

Regular Expressions

- Regular expressions are mechanisms used to describe tokens.
- Each regular expression describes a language: a set of valid tokens.
  - We’ll denote the set of utterances (i.e., the language) a regular expression, re, denoted by L[re].
- Regular expressions can only describe relatively simple languages, but they permit very efficient membership tests and matching.
- Problem: need a language to describe regular expressions, but we’re only now learning how to describe languages, which leads to an interesting problem in recursion.
We have three concepts to consider

- The kind of regular expression (its concept)
- The way we write the regular expression (its denotation)
- The language it describes (its meaning)

As you might expect, if we treat this mathematically, the choice of denotation is independent of the underlying meaning.

- You will see different denotations for regular expressions
- We will not describe formally how we denote regular expressions, but it should be clear from context.
- In contrast, we will describe formally the relationship between a regular expression and the language it describes.

- Side note: In lexical analysis, we will associate a token type (or perhaps an action) with each regular expression.
- Like all languages, the language of regular expressions is based on an underlying alphabet, \( \Sigma \).
- There is a special symbol, \( \varepsilon \), not in \( \Sigma \).
  - \( \varepsilon \) is written as the greek letter epsilon
  - \( L[\varepsilon] \) is the set of the empty string, \{ "" \}.
- Any single symbol in \( \Sigma \) is a regular expression.
  - The regular expression for that symbol is that symbol
  - For each symbol \( s \) in \( \Sigma \), \( L[s] = \{ s \} \).
- The Concatenation of any two regular expressions is a regular expression denoting the combinations of strings from those two regular expressions.
  - If \( R \) and \( S \) are regular expressions, the concatenation of \( R \) and \( S \) is written \((R.S)\)
  - \( L[(R.S)] = \{ \text{concat}(x,y) \mid x \in L[R], y \in L[S] \} \)
- The concatenation of any two strings is the two strings written in sequence with no intervening space.
  - \( \text{concat}("hello", "world") = "helloworld" \)
- \( \varepsilon \) is the identity string for concatenation
  - \( \text{concat}(\varepsilon,s) = \text{concat}(s,\varepsilon) = s \)
- We sometimes will want shorthand
  - \((R.R)\) may also be written as \((R)^2\); \((R.R).(R)\) as \((R)^3\); and so on and so forth.
- The Alternation of any two regular expressions is a regular expression denoting strings in the language of either regular expression.
  - If \( R \) and \( S \) are regular expressions, the alternation of \( R \) and \( S \) is written \((R|S)\).
  - \( L[(R|S)] = L[R] \cup L[S] \)
- If \( R \) is a regular expression, then the Kleene Star of \( R \) is a regular expression.
  - If \( R \) is a regular expression, the Kleene star of \( R \) is written \((R^*)\).
  - \( L[(R^*)] = L[\varepsilon] \cup L[(R)^1] \cup L[(R)^2] \ldots \)
  - Alternately, \( L[(R^*)] = L[\varepsilon] \cup L[(R^*)] \cup L[(R^*)] \ldots \)
Simplifying Regular Expressions

- As you may have noted, we need a lot of parentheses in writing regular expressions.
- At one level, these parentheses are necessary to resolve ambiguities.
  - For example, is RS|T really ((R)(S))(T) or (R)((S)|(T))?
  - Does it matter?
  - Similarly, is RST really ((R)(S))(T) or (R)((S)(T))?
  - Does it matter?
  - Is RT* really ((R)(T))* or (R)((T)*)?
  - Does it matter?
- However, we can use the same techniques as we use with simple algebra to avoid these parentheses:
  - Assign precedence to operations.
    - Kleene star has the highest precedence.
    - Concatenation has the next highest precedence.
    - Alternation has the lowest precedence.
- We also tend to elide the concatenations symbol, just as we elide the multiplication symbol.
- Some examples
  - ab* is (a)((b)* and not ((a)(b))*
  - a|bc is (a)|((b)(c)) and not ((a)|(b))(c)
- When it’s appropriate, we assume that the operations are left-associative.
  - In RST, R and S bind first.

Regular Expression Shorthands

- There are a number of shorthands for regular expression.
- None add any expressive power and all can be “compiled” into normal regular expressions.
- A postfix plus sign (+) for “at least one instance”. a+ is (aa*).
- Brackets for alternation of larger sets of symbols. [abc] is a|b|c. [a-c] is also a|b|c.
- Negation of sets of symbols (which should work only if the set of symbols is finite).[^abc] is "anything except a, b, or c.". If sigma is "lowercase letters", [^abc] is shorthand for d|e|f|g|h|i|j|k|m|n|o|p|q|r|s|t|u|v|w|x|y|z
- A postfix question mark for “optional”. a? is shorthand for (a|\epsilon).
- A power represents repeated concatenation. a^n is shorthand for aa...a.
- Naming regular expressions and then using names within other regular expressions (provided the inclusion is not recursive).
  - We can translate any regular expression with names to a normal regular expression by substituting the named expression for the name.
- Some of the shorthands available in Perl and such really extend the power of regular expressions, and are not permitted in standard regular expressions.
  - “The thing matched in an earlier part of the pattern.”
  - “Not this regular expression”
  - ...
Sample Regular Expressions

- All strings of a’s and b’s: (a|b)*
- Strings of a’s and b’s with exactly one b: a*ba*
- Strings of one or more a’s: aa* (also a*a, also a*a*a)

Limitations

- Regular expressions, while powerful, also have some limitations.
- In particular, there are relatively simple languages you cannot describe with regular expressions.
- Here are a few:
  - Strings of a’s and b’s with equal numbers of a’s and b’s.
  - Palindromes

Using Regular Expressions for Lexical Analysis

- How do we use regular expressions for lexical analysis?
- We start by writing a regular expression for each nonterminal in the language.
- For example:

  BEGIN: [Bb][Ee][Gg][Ii][Nn]
  NUMBER: (+|-|epsilon)([1-9][0-9]*)|(0x[0-9]+)|([1-9].[0-9]+)|...
  IDENTIFIER: [a-zA-Z_][a-zA-Z_0-9]*
  OPEN: 
  CLOSE: 

- You may have noted that we need to be careful to distinguish the symbols used in building regular expressions from the similar characters.
  - Most regular expression implementations provide some mechanism for distinguishing the two.
- Note that we have only discussed how to describe tokens, not how a typical lexical analyzer uses those descriptions.
  - In particular, what happens if multiple regular expressions apply?
- Two general rules apply:
  - Use the longest possible match.
  - If two different patterns have an equally long match, use the one that appears first.
- Note that the first rule can make our matching less efficient, since we may have to “look ahead” a great deal to determine whether or not something matches a longer pattern and find that it doesn’t.
  - In practice, this problem rarely happens.
- In the next few classes, we’ll learn how to turn regular expression specifications into programs that match regular expressions or tokenize.
Lab on Regular Expressions

- You can ground this learning in laboratory work.
- We probably won’t have time during class, so you get to do it on your own if you’d like.