Strings and Patterns

String Processing Design Issues

Size of alphabet: the alphabet is the set of characters in the string language
Boolean alphabet = {0,1}
simple English alphabet = {a,b,c,...,x,y z}
Larger alphabets require more effort for matching

Redundancy: strings with high character redundancy require more comparisons
e.g.: "abbaabbacdcdcd" is more redundant than "thisstringistoolong"

Processor/data-structure: Sometimes it is easier not to decrement the pointer index (i.e. not to back up). Some languages do not support intertwined functional recursion.

Brute-force-match

```
pattern[0..3] =   "bcde"
string[0..14] =   "abcdeabcdeabcde"
```

Compare the first character in the pattern to the first character in the string. If it matches, compare the rest of the pattern to the string, one character at a time. When it doesn't match, compare the next character in the string to the beginning of the pattern. Repeat until the string is exhausted.

```
pattern[0..M]
string[0..N]
brute-match[string]  =
  i := 0;  j := 0;
  loop until j=M or i=N
    if pattern[j] = string[i]
      then  i++; j++                    */increment if matching
      else  i := i-j+1; j := 0          */reset if not matching
  if j=M                                */pattern match found
    then return (i-M)                   */location of start of match
    else return i                       */end of string
```

N*M comparisons worst-case, but average case is almost always N+M

KMP-match and Boyer-Moore-match

KMP-match is the same as brute-force-match, except when there is a mismatch, it backs up the string pointer as little as possible by using the knowledge that the examined string characters before the mismatch do not match the pattern.

Boyer-Moore-match is the same as KMP-match, except that it makes the comparisons from right to left, thus assuring that the maximum number of mismatching characters can be skipped. If the front of the pattern does not match, then you can only skip ahead one character in

the string, but if the end of the pattern does not match, then the entire length of the pattern can be skipped. Boyer-Moore requires reverse traversal of an array, which is expensive in some implementations.

Pattern-matching Languages

Expand the concept of a pattern to include different types of matches. These three define a *regular language*:

Concatenation: characters must be adjacent (standard)

Or: a character location in the pattern may have more than one acceptable match.
`A(B+C)D` matches both `ABD` and `ACD`

Closure: a pattern may be repeated any number of times (including zero)
`A* = AAA... (AB)* = ABABAB...`

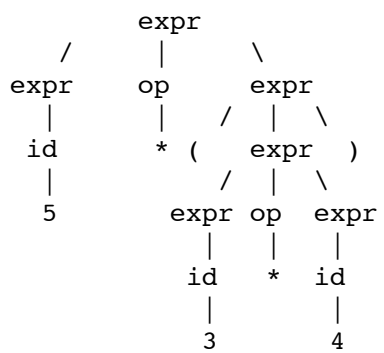
Parsing a context-free grammar

Parsers convert a string input into a tree, with tree-leaves forming the words/characters and internal nodes describing the type of expression.

Example, simple arithmetic expressions:

```
expr  -->   expr  op  expr
expr  -->  ( expr )
expr  -->  - expr
expr  -->  id
op  -->  +
op  -->  -
op  -->  *
op  -->  /
op  -->  ^
```

Parse 5 * (3 + 4):



Example, Backus-Naur form for a regular language:

```
<expression>  ::=  <term>  |  <term>  <expression>
<term>  ::=  <factor>  |  <factor><term>
<factor>  ::=  (<expression>)  |  v  |  (<expression>)*  |  v*
```

Top-down Parsing

The input expression (in the above example, `5*(3+4)`) is processed one character at a time. The parser calls each production rule recursively until either the entire expression is accepted or a syntax error is identified.

Example of a parser/recognizer:

```
input[0..6]          */in the example, N=7 for the seven characters in "5*(3+4)"
i = 0
expression =
      case input[i]
            "("         i++;
                              expression;
                              if input[i] = ")" then i++ else ERROR
            "-"         operator;
                        expression
            id          i++           */if id has more than one character
                                      */then need to process id length here
            otherwise   expression;
                        operator;
                        expression

operator =
      if  member[ input[i], { "+",  "-",  "*", "/", "^" } ]
            then i++
            else ERROR
```

Pattern Variables

Some programming languages allow patterns as variables. Identifiers within the pattern control decomposition and construction.

Record:

```
((first-name last-name) (address-digits street-name city state) (phone))
```

Example:

```
Get-component [((_ _) (_ _ city_ _) (_)),my-record]
      returns city = <my-city>

Change-city [record-pattern, new-city]
      returns ((_ _) (_ _ new-city _) (_))
```