Applications of Regular Expressions

Unix RE's

Text Processing

Lexical Analysis

Some Applications

- RE's appear in many systems, often private software that needs a simple language to describe sequences of events.
- We'll use Junglee as an example, then talk about text processing and lexical analysis.

Junglee

- Started in the mid-90's by three of my students, Ashish Gupta, Anand Rajaraman, and Venky Harinarayan.
- Goal was to integrate information from Web pages.
- Bought by Amazon when Yahoo! hired them to build a comparison shopper for books.

Integrating Want Ads

- Junglee's first contract was to integrate on-line want ads into a queryable table.
- Each company organized its employment pages differently.
  - Worse: the organization typically changed weekly.

Junglee's Solution

- They developed a regular-expression language for navigating within a page and among pages.
- Input symbols were:
  - Letters, for forming words like "salary".
  - HTML tags, for following structure of page.
  - Links, to jump between pages.

Junglee's Solution – (2)

- Engineers could then write RE's to describe how to find key information at a Web site.
  - E.g., position title, salary, requirements,...
- Because they had a little language, they could incorporate new sites quickly, and they could modify their strategy when the site changed.

RE-Based Software Architecture

- Junglee used a common form of architecture:
  - Use RE's plus actions (arbitrary code) as your input language.
  - Compile into a DFA or simulated NFA.
  - Each accepting state is associated with an action, which is executed when that state is entered.

UNIX Regular Expressions

- UNIX, from the beginning, used regular expressions in many places, including the "grep" command.
  - Grep = "Global (search for a) Regular Expression and Print."
- Most UNIX commands use an extended RE notation that still defines only regular languages.

UNIX RE Notation

- $[a_1a_2...a_n]$ is shorthand for $a_1+a_2+...+a_n$.
- *Ranges* indicated by first-dash-last and brackets.
  - Order is ASCII.
  - Examples: [a-z] = "any lower-case letter," [a-zA-Z] = "any letter."
- Dot = "any character."

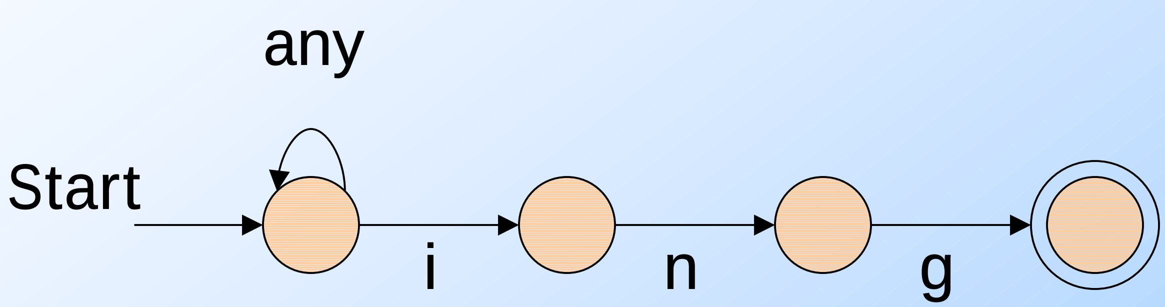
UNIX RE Notation – (2)

- | is used for union instead of +.
- But + has a meaning: "one or more of."
  - E+ = EE*.
  - Example: [a-z]+ = "one or more lower-case letters.
- ? = "zero or one of."
  - E? = E + ϵ.
  - Example: [ab]? = "an optional *a* or *b*."

Example: Text Processing

- Remember our DFA for recognizing strings that end in "ing"?
- It was rather tricky.
- But the RE for such strings is easy: **.*ing** where the dot is the UNIX "any".
- Even an NFA is easy (next slide).

NFA for "Ends in *ing*"



Lexical Analysis

- The first thing a compiler does is break a program into *tokens* = substrings that together represent a unit.
  - Examples: identifiers, reserved words like "if," meaningful single characters like ";" or "+", multicharacter operators like "<=".

Lexical Analysis – (2)

- Using a tool like Lex or Flex, one can write a regular expression for each different kind of token.
- Example: in UNIX notation, identifiers are something like [A-Za-z][A-Za-z0-9]*.
- Each RE has an associated action.
  - Example: return a code for the token found.

Tricks for Combining Tokens

- There are some ambiguities that need to be resolved as we convert RE's to a DFA.
- Examples:
  1. "if" looks like an identifier, but it is a reserved word.
  2. < might be a comparison operator, but if followed by =, then the token is <=.

Tricks – (2)

- Convert the RE for each token to an ϵ–NFA.
  - Each has its own final state.
- Combine these all by introducing a new start state with ϵ-transitions to the start states of each ϵ–NFA.
- Then convert to a DFA.

Tricks – (3)

- If a DFA state has several final states among its members, give them priority.
- Example: Give all reserved words priority over identifiers, so if the DFA arrives at a state that contains final states for the “if” ϵ–NFA as well as for the identifier ϵ–NFA, if declares “if”, not identifier.

Tricks – (4)

- It's a bit more complicated, because the DFA has to have an additional power.
- It must be able to read an input symbol and then, when it accepts, put that symbol back on the input to be read later.

Example: Put-Back

- Suppose “<” is the first input symbol.
- Read the next input symbol.
  - If it is “=”, accept and declare the token is <=.
  - If it is anything else, put it back and declare the token is <.

Example: Put-Back – (2)

- Suppose “if” has been read from the input.
- Read the next input symbol.
  - If it is a letter or digit, continue processing.
    - You did not have reserved word “if”; you are working on an identifier.
  - Otherwise, put it back and declare the token is “if”.