Session 4

Examples and applications of regular expressions

Strings with an odd Number of 1's

- Focusing on the first 1:
  - We start with with 1: 0^*10^*
  - Subtrings with a pair of 1's and any number of 0's: $(10^*10^*)^*$
  - The concatenation: $0^*10^*(10^*10^*)^*$
- Focusing on the first 1, but also in the second substring:
  - $0^*1(0^*10^*10^*)^*0^*$
- Focusing on the last 1:
  - $(0^*10^*1)^*0^*10^*$
- Focusing on the 1 in the middle:
  - $0^*(10^*10^*)^*1^*(0^*10^*1)0^*$
- But not:
  - $(10^*10^*)^*10^*$
  - We need to allow the initial 0's, so: $0^*(10^*10^*)^*10^*$

Strings of length 6 or less

- A very concrete way:
  - $\Lambda + 0 + 1 + 00 + 01 + 10 + 11 + \ldots + 111110 + 111111$
- Lets try to do a little better:
  - First, think of strings of length 6 exactly:
    - $(0 + 1)(0 + 1)(0 + 1)(0 + 1)(0 + 1)(0 + 1)$
  - Then, think of the exponential notation:
    - $(0 + 1)^6$
  - Final, allow strings of length less than 6:
    - $(0 + 1 + \Lambda)^6$

Strings ending in 1 with no "00"

- $L = \{x \in \{0, 1\}^* \mid x$ ends with 1 and does not contain 00$\}$
- No 0 can follow a 0: 0 is either at the end or followed by 1
- But x ends with 1
- So, x is either 1 or copies of 01: $(1 + 01)^*$
  - $\{1, 01, 11, 101, 011, 0101\ldots\}$
- However, this does allow Λ, which does not end in 1 and has no 00
  - This can be fixed with: $(1 + 01)^*1$
- But now, 01 is not in the language, so:
    - $(1 + 01)^*(1 + 01)$
    - $(1 + 01)^+$

The language of C identifiers

- Let's l and d stand for letter and digit respectively
  - l stands for $a + b + \ldots + z + A + B + \ldots + Z$
  - d stands for $0 + 1 + 2 + \ldots + 9$
- An identifier in C is a string of length 1 or more containing letters, digits and underscore ("_"):

  $(l + \_)(l + d + \_)^*$
- Examples:
  - "cis625", "cis_625", "cis6_2_5", "_625"

Real Literals in Pascal

- Notation:
  - l stands for $a + b + \ldots + z + A + B + \ldots + Z$
  - d stands for $0 + 1 + 2 + \ldots + 9$
  - s stands for "sign" (shorthand for $\Lambda + a + m$, where a is plus and m is minus)
  - p stands for "point"
  - E is a symbol of Σ
- Real literals: $sd^+(pd^+ + pd^+Esd^+ + Esd^+)$
- Examples: +6.25, 6.25, -6.25E+2, 6.25E-2, -2E2

Applications of Regular Expressions

- Provide a “picture” of a pattern that we want to recognize
- They can be “compiled” into determinist automata, which can be modeled to recognize patterns in texts
- Two important applications:
  - Lexical analyzers
  - Texts search in Internet

UNIX Notation for *RE*

- Σ = The set of ASCII characters
- The *grep* command:
  - *Global (search for) Regular Expressions and Print*
- Short hand definitions: Character classes
  - The dot “.” stands for any character
  - [$a_1a_2\ldots a_k$] stands for the *RE*: $a_1 + a_2 + \ldots + a_k$
    - e.g. The characters used for comparison in C: [<>=!]
  - [*x-y*] stand for range definitions:
    - e.g. [A-Za-z0-9] stands for the set of all letters and digits
    - A minus sign “–” is placed first or last (to avoid confusion):
      [–+.0-9] is the set {-, +, . , 0…9}
  - For reserved characters of UNIX, we use the backslash \
    [0-9\.] is the set of digits and the dot (not any character)

UNIX Notation for *RE*

- Meaning of UNIX operators
  - | is used in place of +
  - The operator ? means zero or one of: **R?** = Λ + *R*
  - + means one or more: **R+** = RR^*
  - The operator {*n*} means *n* copies of: **R{5}** = *RRRRR*
  - * in UNIX has the usual meaning (not a superscript!)
- Also:
  - [:digit:] stands for [0-9] (not necessarily in ASCII)
  - [:alpha:] stands for [A-Za-z]
  - [:alnum:] stands for [A-Za-z0-9]
- Operators precedence is as usual (with ?, + and {*n*} treated like *)
- UNIX extensions to name and refer to previous strings that have matched a pattern (allowing to the recognition of non-regular languages) are not considered here!

Lexical Analyzers

- *Lexical analyzer*: the part of a compiler that scans the source code and identifies *tokens* (i.e. basic or atomic symbols, or entries to the symbol’s table)
  - Keywords
  - Identifiers (names, variables, etc.)
- *Lexical-analyzer generator*:
  - UNIX’s *lex* (*flex* in GNU)
  - Accepts a list of regular expressions each followed by a a bracketed piece of code, indicating what to do when an instance of the token described by the *RE* is found
- Advantages:
  - A high level description of a lexical analyzer
  - Automatic generation of complicated code
  - Easy to create and modify

Example

- Partial input to the *lex* command:

```
Else                    {return(ELSE)}
[A-Za-z][A-Za-z0-9]*    {code to Enter identifier
                          in Symbol table;
                          return(Id); }
>=                      {return(GE);}
=                       {return(QE);}
```

…integers, floating-point, character strings, etc.

- Conversion of regular expressions to an automata for processing the corresponding strings

Finding patterns in texts

- *RE* are useful for describing searches for interesting patterns
- Descriptions of vaguely defined class of patterns in texts
- Patters that are hard to define…
- Easy to specify and modify

Example: Detecting addresses in web-pages

- First: the street address (UNIX Notation)
  - Street|St\.|Ave\.|Road|Rd\.
- Next: Name of the street
  - [A-Z][a-z]* (e.g. Island)
- But… what about streets with two or more names?
  - '[A-Z][a-z]*([A-Z][a-z]*)*' (e.g. Road Island Av.)
- Next: House numbers
  - String of digits… probably followed by letters as in "123A Main St."
  - [0-9]+[A-Z]?
- The full expression:
  - '[0-9]+[A-Z]? [A-Z][a-z]*([A-Z][a-z]*)* (Street|St\.|Ave\.|Road|Rd\.)

Dr. Luis Pineda, IIMAS, UNAM & OSU-CIS, 2003

Example: Detecting addresses in web-pages

- But what about
  - Streets with a different name: "Boulevar", "Place",…
  - Streets with ordinal abbreviations: 42nd St.
  - Post-Office boxes or rural-delivery routes
  - Streets names that do not end with "Street", like *El Camino Real* in Silicon Valley (Spanish name for The Royal Road)
    - *El Camino Real Road*?
    - *2000 El Camino Real*
- It is really a knowledge engineering task!
- We can appreciate the power of Regular expressions
  - Expressive
  - Economical

Dr. Luis Pineda, IIMAS, UNAM & OSU-CIS, 2003