Regular Expressions

Based on slides of Aaron Deever

Regular Expressions

- Another means to describe languages accepted by Finite Automata.

- In some books, regular languages, **by definition,** are described using regular expressions.

(Sipser defines a regular language as one recognized by a finite automaton)

Regular Expressions

- A <u>regular expression</u> describes a language using only the set operations of:
  - Union
  - Concatenation
  - Kleene Star

Regular Expressions

- Regular expressions are the mechanism by which regular languages are described:
  - Take the "set operation" definition of the language and:
    - Replace {} with ()
    - (Some definitions also replace ∪ with +)
  - And you have a regular expression

  - (We've used the "set operation" way of representing languages a bunch already)
    - In examples in class and in the homework, you were asked to start with simple languages like {0}, {1}, {ε} and ∅ and build up more complicated languages using just those along with the regular operations of union, concatenation and Kleene star.

Regular Expressions

Language Set Notation	Regular Exp. Notation
{ε}	ε
{0}{1}{1}	011
{0,1}	0 ∪ 1
{0, 01}	0 ∪ 01
$\{110\}^*\{0,1\}$	$(110)^*(0 \cup 1)$
$\{10, 11, 01\}^*$	$(10 \cup 11 \cup 01)^*$
$\{0, 11\}^*(\{11\}^* \cup \{101, \varepsilon\})$	$(0 \cup 11)^*((11)^* \cup 101 \cup \varepsilon)$

Regular Expression Definition

- R is a regular expression if R equals

Base cases

1. $\emptyset$ (representing the empty language)
2. ε (representing the language $\{\varepsilon\}$)
3. a, for each $a \in \Sigma$, (representing the language $\{a\}$)

Recursive cases

4. $(R_1 \cup R_2)$ where R_1 and R_2 are regular expressions
5. (R_1R_2) where R_1 and R_2 are regular expressions
6. $(R_1)^*$ where R_1 is a regular expression

Regular Expressions

- Some shorthand
  - If we apply precedence to the operators, we can relax the full parenthesized definition:
    - Kleene star has highest precedence
    - Concatenation has middle precedence
    - Union has lowest precedence
  - Thus
    - $a \cup b^{*}c$ is the same as $(a \cup ((b^{*})c))$
    - $(a \cup b)^{*}$ is not the same as $a \cup b^{*}$

Regular Expressions

- More shorthand
  - Union over entire alphabet can be represented as Σ
  - Example: $\Sigma = \{a,b,c\}$
    - $(a \cup b \cup c) = \Sigma$
    - $(a \cup b \cup c)^* = \Sigma^*$

Regular Expressions

- More shorthand
  - Equating regular expressions.
    - Two regular expressions are considered equal if they describe the same language
    - $1^*1^* = 1^*$
    - $(a \cup b)^* \neq a \cup b^*$
- For convenience
  - $R^+ = RR^*$
  - R^k = concatenation of R, k times

Regular Expressions

- Note that:
  - A regular expression is <u>not</u> a language
  - A regular expression is used to <u>describe</u> a language.

  - It is incorrect to say that for a language L,
    - $L = (a \cup b \cup c)^*$
  - But it's okay to say that L is described by
    - $(a \cup b \cup c)^*$

Examples

- $\{ w \in \{0,1\}^* \mid w$ contains the substring 001 $\}$
  - $(0 \cup 1)^*001(0 \cup 1)^* = \Sigma^*001\Sigma^*$
- $\{ w \in \{0,1\}^* \mid |w|$ is divisible by 2 or 3 $\}$
  - $((0 \cup 1)(0 \cup 1))^* \cup ((0 \cup 1)(0 \cup 1)(0 \cup 1))^*$
  - $((0 \cup 1)^2)^* \cup ((0 \cup 1)^3)^* = (\Sigma\Sigma)^* \cup (\Sigma\Sigma\Sigma)^*$
- $\{ w \in \{0,1\}^* \mid w$ does not contain two consecutive 0's$\}$
  - $(0 \cup \epsilon)(1^+0)^*1^*$
- $\{ w \in \{0,1\}^* \mid |w| < 4 \}$
  - $\epsilon \cup 0 \cup 1 \cup (0 \cup 1)^2 \cup (0 \cup 1)^3 = \epsilon \cup \Sigma \cup \Sigma^2 \cup \Sigma^3$

Examples

- All finite languages can be described by regular expressions
  - How?
  - A finite language is a finite set of strings
    - Each string is just a concatenation of symbols in the alphabet
      - Each symbol in the alphabet is a regular expression
        Concatenation is allowed in building up regular expressions
    - The language is the union of these strings
  - Example:
    - L = {a, aa, aba, aca}
    - R = a ∪ aa ∪ aba ∪ aca

Examples

- For $\Sigma = \{0,1\}$, what language is described by the following:
  - $\varepsilon \cup 0 \cup 1 \cup (0 \cup 1)^*(00 \cup 10 \cup 11)$
- $L = \{x \in \{0,1\}^* \mid x \text{ does not end in } 01\}$
  - If x does not end in 01, then either
    - $|x| < 2$ or
    - x ends in 00, 10, or 11

Useful Properties of Regular Expressions

- For regular expressions L, M and N
  - Commutative
    - $L \cup M = M \cup L$
  - Associative
    - $(L \cup M) \cup N = L \cup (M \cup N)$
    - $(LM)N = L(MN)$
  - Identities
    - $\varnothing \cup L = L \cup \varnothing = L$
    - $\epsilon L = L\epsilon = L$
    - $\varnothing L = L\varnothing = \varnothing$
  - Distributive
    - $L(M \cup N) = LM \cup LN$
    - $(M \cup N)L = ML \cup NL$
  - Idempotent
    - $L \cup L = L$

Useful Properties of Regular Expressions

- For regular expressions L, ∅, ε
-
  - $(L^*)^* = L^*$
  - $\varnothing^* = \varepsilon$
  - $\varepsilon^* = \varepsilon$
  - $L^+ = LL^*$
  - $L^* = L^+ \cup \varepsilon$

Applications using regular expressions

Program	**(Original) Author**	**Version**	**Regex Engine**
awk	Aho, Weinberger, Kernighan	*generic*	DFA
new *awk*	Brian Kernighan	*generic*	DFA
GNU *awk*	Arnold Robbins	*recent*	Mostly DFA, some NFA
MKS *awk*	Mortice Kern Systems		POSIX NFA
mawk	Mike Brennan	*all*	POSIX NFA
egrep	Alfred Aho	*generic*	DFA
MKS *egrep*	Mortice Kern Systems		POSIX NFA
GNU Emacs	Richard Stallman	*all*	Trad. NFA (POSIX NFA available)
Expect	Don Libes	*all*	Traditional NFA
expr	Dick Haight	*generic*	Traditional NFA
grep	Ken Thompson	*generic*	Traditional NFA
GNU *grep*	Mike Haertel	Version 2.0	Mostly DFA, but some NFA
GNU *find*	GNU		Traditional NFA
lex	Mike Lesk	*generic*	DFA
flex	Vern Paxson	*all*	DFA
lex	Mortice Kern Systems		POSIX NFA
more	Eric Schienbrood	*generic*	Traditional NFA
less	Mark Nudelman		Variable (usually Trad. NFA)
Perl	Larry Wall	*all*	Traditional NFA
Python	Guido van Rossum	*all*	Traditional NFA
sed	Lee McMahon	*generic*	Traditional NFA
Tcl	John Ousterhout	*all*	Traditional NFA
vi	Bill Joy	*generic*	Traditional NFA

Practical Uses for Regular Expressions

- Python
  - [ ] for union
  - * for Kleen star
  - **abc** for concatenation
  - Example
    - $(a \cup b)^*c(\varepsilon \cup d \cup e)$
    - [ab]*c[de]

Regular Expressions in Practice

- How can we implement in code an algorithm to take as input a regular expression and an arbitrary string, and output whether that string is an element of the language described by the regular expression?

  - Parse the regular expression to convert it into an expression tree
  - Build the NFA piece by piece from the expression tree
  - Convert the NFA to a DFA using subset construction
  - Remove unreachable states (if necessary) by depth-first search from starting state (later we'll discuss a minimization algorithm to remove redundant states)
  - Run the string through the DFA to see if accepted

Regular Expressions in Practice

- Regular expression engines have grown to encompass more than just DFA implementations
  - Traditional NFA investigate all possible branches in a particular order
    - Reduces space complexity (number of states needed)
    - But can exponentially increase time complexity in worst case
    - Returns first matching string, so doesn't necessarily find longest

Regular Expressions in Practice

- Regular expression engines have grown to encompass more than just DFA implementations
  - POSIX NFA are similar to traditional NFA
    - But they continue backtracking to try all branches
    - Guarantee finding longest matching string
    - Run even slower than traditional NFA

Next

- Regular expressions and finite automata are equivalent in their ability to describe languages.
  - Every regular expression has a FA that accepts the language it describes
  - The language accepted by a FA can be described by some regular expression.
- The Kleene Theorem! (1956)