CSE 311: Foundations of Computing

Topic 9: Languages

Theoretical Computer Science

Strings

- An *alphabet* Σ is any finite set of characters

- The set Σ^* of *strings* over the alphabet Σ
  - example: {0,1}* is the set of *binary strings*

    0, 1, 00, 01, 10, 11, 000, 001, … and “”

- Σ^* is defined recursively by
  - **Basis:** $\varepsilon \in \Sigma^*$ (ε is the empty string, i.e., “”)
  - **Recursive:** if $w \in \Sigma^*$, $a \in \Sigma$, then $wa \in \Sigma^*$

Languages: Sets of Strings

- Subsets of strings are called *languages*
- Examples:
  - $\Sigma^* =$ All strings over alphabet Σ
  - Palindromes over Σ
  - Binary strings that don't have a 0 after a 1
  - Binary strings with an equal # of 0's and 1's
  - Legal variable names in Java/C/C++
  - Syntactically correct Java/C/C++ programs
  - Valid English sentences

Foreword on Intro to Theory C.S.

- Look at different ways of defining languages
- See which are more expressive than others
  - i.e., which can define more languages

- Later: connect ways of defining languages to different types of (restricted) computers
  - computers capable of recognizing those languages i.e., distinguishing strings in the language from not

- Consequence: computers that recognize more expressive languages are more powerful

Palindromes

Palindromes are strings that are the same when read backwards and forwards

Basis:

ε is a palindrome

any $a \in \Sigma$ is a palindrome

Recursive step:

If p is a palindrome,

then apa is a palindrome for every $a \in \Sigma$

Regular Expressions

Regular expressions over Σ

- **Basis:**
  - ε is a regular expression (could also include $\varnothing$)
  - a is a regular expression for any $a \in \Sigma$
- **Recursive step:**

  If **A** and **B** are regular expressions, then so are:

  - $A \cup B$
  - AB
  - A^*

Each Regular Expression is a "pattern"

ε matches only the **empty string**

a matches only the one-character string *a*

A ∪ **B** matches all strings that either **A** matches or **B** matches (or both)

AB matches all strings that have a first part that **A** matches followed by a second part that **B** matches

A* matches all strings that have any number of strings (even 0) that **A** matches, one after another (**ε** ∪ **A** ∪ **AA** ∪ **AAA** ∪ **…**)

Definition of the *language* matched by a regular expression

Language of a Regular Expression

The language defined by a regular expression:

$$\mathrm{L}(\varepsilon) = \{\varepsilon\}$$

$$\mathrm{L}(a) = \{a\}$$

$$\mathrm{L}(A \cup B) = L(A) \cup L(B)$$

$$\mathrm{L}(AB) = \{x \;:\; \exists y \in L(A), \exists z \in L(B)\ (x = y \bullet z)\}$$

$$\mathrm{L}(A^*) = \bigcup_{n=0}^{\infty} L(A^n)$$

A^n defined recursively by

$$A^0 = \emptyset$$

$$A^{n+1} = A^n A$$

Examples

001*

0*1*

Examples

001*

{00, 001, 0011, 00111, ...}

0*1*

Any number of 0's followed by any number of 1's

Examples

$(0 \cup 1)\, 0\, (0 \cup 1)\, 0$

$(0^*1^*)^*$

Examples

$(0 \cup 1)\, 0\, (0 \cup 1)\, 0$

{0000, 0010, 1000, 1010}

$(0^*1^*)^*$

All binary strings

Examples

- All binary strings that contain 0110

$$(\mathbf{0} \cup \mathbf{1})^* \, \mathbf{0110} \, (\mathbf{0} \cup \mathbf{1})^*$$

- All binary strings that begin with a string of doubled characters (00 or 11) followed by 01010 or 10001

$$(\mathbf{00} \cup \mathbf{11})^* \, (\mathbf{01010} \cup \mathbf{10001}) \, (\mathbf{0} \cup \mathbf{1})^*$$

Examples

- All binary strings that have an even # of 1's

  e.g., $0^*(10^*10^*)^*$

- All binary strings that *don't* contain 101

  e.g., $0^*(1 \cup 1000^*)^*(\varepsilon \cup 10)$

  at least two 0s between 1s

Finite languages vs Regular Expressions

- All finite languages have a regular expression.

    (a language is finite if its elements can be put into a list)

  Why?

- Given a list of strings s_1, s_2, ..., s_n

  Construct the regular expression

$$s_1 \cup s_2 \cup \ldots \cup s_n$$

(Could make this formal by induction on n)

Finite languages vs Regular Expressions

- Every regular expression that does not use *
  generates a finite language.

  Why?

- Prove by structural induction on the syntax of regular expressions!

Star-free implies finite

Let A be a regular expression that does not use *. Then L(A) is finite.

Proof: We proceed by structural induction on A.

Case ε: L(ε) = {ε}, which is finite

Case a: L(*a*) = {a}, which is finite

Case A ∪ B:

L(A ∪ B) = L(A) ∪ L(B)

By the IH, each is finite, so their union is finite.

Star-free implies finite

Let A be a regular expression that does not use *. Then L(A) is finite.

Proof: We proceed by structural induction on A.

Case AB:

$$\mathrm{L(AB)} = \{x \;:\; \exists y \in L(A), \exists z \in L(B)\ (x = y \bullet z)\}$$

By the IH, $\mathrm{L(A)}$ and $\mathrm{L(B)}$ are finite.

Every element of $\mathrm{L(AB)}$ is covered by a pair (y, z) where $y \in L(A)$ and $z \in L(B)$, so $\mathrm{L(AB)}$ is finite.

(No case for A*!)

Finite languages vs Regular Expressions

Key takeaways:

– Regular expressions can represent all finite languages

– To prove a language is represented by a regular expression, just describe the regular expression.

– Regular expressions are more powerful than finite languages (e.g., 0* is an infinite language)

– To prove something about *all* regular expressions, use structural induction on the syntax.

Regular Expressions in Practice

- Used to define the "tokens": e.g., legal variable names, keywords in programming languages and compilers
- Used in **`grep`**, a program that does pattern matching searches in UNIX/LINUX
- Pattern matching using regular expressions is an essential feature of PHP
- We can use regular expressions in programs to process strings!

Regular Expressions in Java

- Pattern p = Pattern.compile("a*b");
- Matcher m = p.matcher("aaaaab");
- boolean b = m.matches();

`[01]`	a 0 or a 1	`^` start of string	`$` end of string
`[0-9]`	any single digit	`\.` period	`\,` comma `\-` minus
`.`	any single character		

ab	a followed by b	(**AB**)
(a|b)	a or b	(**A** ∪ **B**)
a?	zero or one of a	(**A** ∪ ε)
a*	zero or more of a	**A***
a+	one or more of a	**AA***

- e.g. `^[\-+]?[0-9]*(\.|\,)?[0-9]+$`

  General form of decimal number e.g. 9.12 or -9,8 (Europe)

Limitations of Regular Expressions

- Not all languages can be specified by regular expressions
- Even some easy things like
  - Palindromes
  - Strings with equal number of 0's and 1's
- But also more complicated structures in programming languages
  - Matched parentheses
  - Properly formed arithmetic expressions
  - etc.