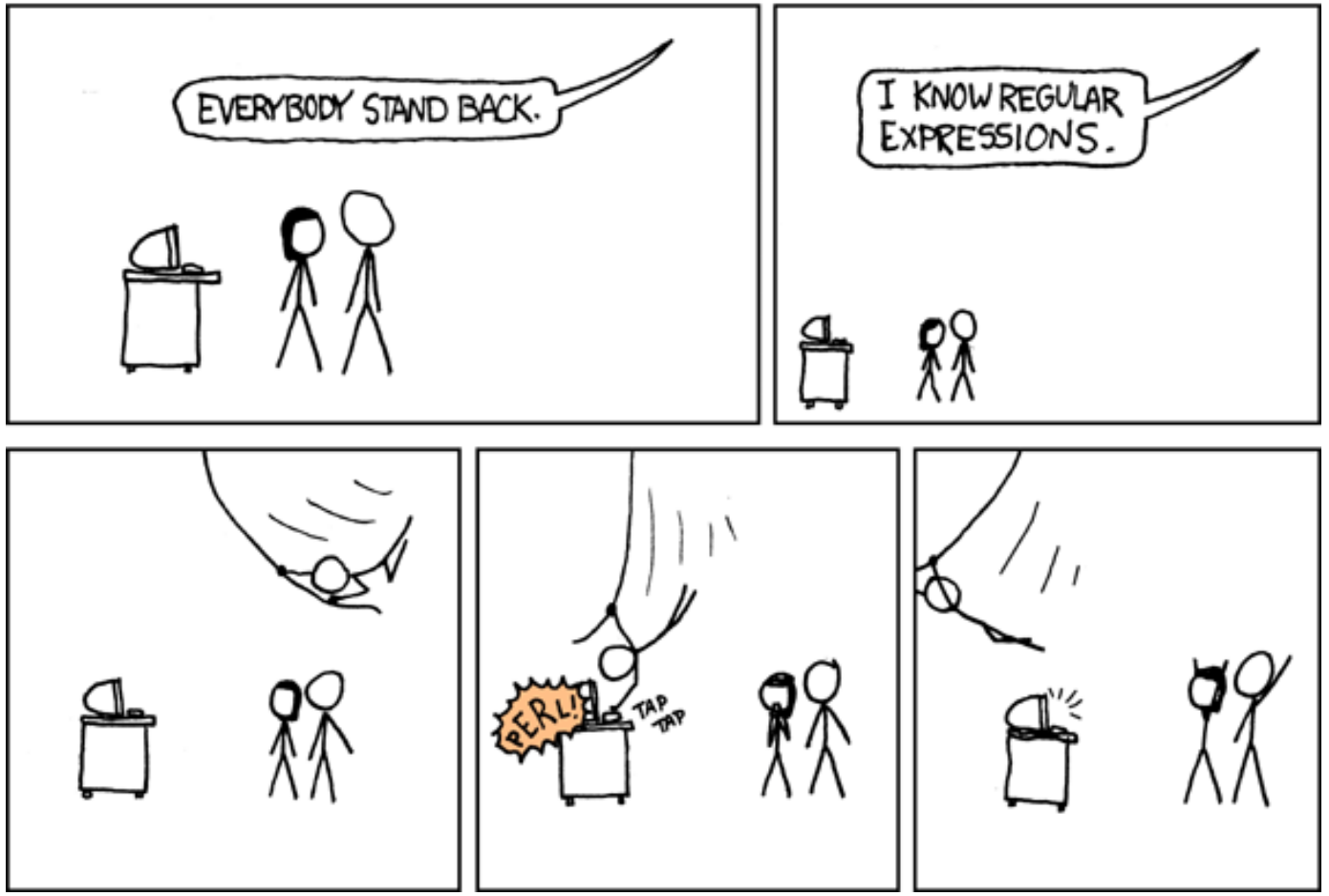
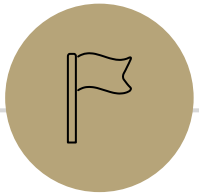


Regular Expressions, Context Free Grammars

CSE 311: Foundations of Computing I
Lecture 19

Announcements

- HW6 due Wednesday at 11:59pm on Gradescope.
- Midterm Corrections due Wednesday at 11:59pm on Gradescope.
  - $\text{midterm\_grade} = 0.75 \cdot \text{original\_grade} + 0.25 \cdot \max(\text{corrected\_grade}, \text{original\_grade})$
  - No Late Days permitted
  - More details posted on Assignments page



Theoretical Computer Science

Recall: Course Goals

1. Learn to make & clearly communicate rigorous formal arguments
   - Mathematical Proofs
2. Understand mathematical objects that are widely used in CS
   - Number Theory, Set Theory, Recursively-Defined Functions
3. Explore and analyze models of computation
   - Regular Expressions, Context-Free Grammars, Finite Automata

Languages

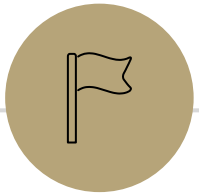
Definition:
A language is a set of all strings.

For Example:

- The set of all valid English sentences.
- The set of all binary strings of even length.
- The set of all syntactically correct Java programs.

Languages in Theoretical Computer Science

- We want to study different models of computation, and the strengths & limitations of each.
- A computer is said to recognize a language L if it can distinguish which strings are $\in L$ vs. which $\notin L$.
- One way to evaluate how powerful a model of computation is is to determine which languages it can recognize.



Regular Languages

One class of languages

Regular Expressions

Basis Step:

- ε is a regular expression
- a is a regular expression for any $a \in \Sigma$.

Recursive Step: If A, B are regular expressions, then:

- $A \cup B$ is a regular expression
- AB is a regular expression
- A^*

Regular Expressions

Each regular expression **matches** a set of strings (a language).

ε matches only the empty string

a matches only the single-character string a

Regular Expressions

Each regular expression **matches** a set of strings (a language).

$A \cup B$ matches all strings that A matches or B matches

E.g. $0 \cup 1$ matches $\{0, 1\}$

AB matches all strings $x = yz$ where A matches y & B matches z

E.g. $0(0 \cup 1)1$ matches $\{001, 011\}$

A^* matches all strings with any number of strings that A matches, i.e. $\varepsilon \cup A \cup AA \cup AAA \cup \ldots$

E.g. 0^* matches $\{\varepsilon, 0, 00, 000, \ldots\}$

Examples

a^*b^*

Matches strings w/ any # of a's followed by any # of b's.

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

Matches strings in the set {0000, 0001, 0100, 0101}.

$(00 \cup 11)^*$

Matches all binary strings where 0s & 1s come in pairs.

Examples

Construct a regular expression that matches the given set of strings.

All binary strings.

$(0 \cup 1)^*$

All binary strings that contain $\mathbf{0110}$.

$(0 \cup 1)^* 0110 (0 \cup 1)^*$

Examples

Construct a regular expression that matches the given set of strings.

All binary strings that have an even number of 1s.

✗ $(0 \cup 11)^*$

✓ $((10^*1) \cup 0)^*$

All binary strings that don't contain 00.

✗ $(01 \cup 1)^*$

✗ $(01 \cup 1)^* 0$

✓ $(01 \cup 1)^* (0 \cup \varepsilon)$

Practical Advice

- Check ε and single character strings. Those are often edge cases.
- List 5 strings that should be matched, and 5 strings that shouldn't be.

  Test your RegEx against those strings.
- Remember $*$ allows for 0 copies! To say "at least one copy", use AA^*.

Exercises

Construct a regular expression that matches the given set of strings.

The set of all binary strings of odd length.

✓ $(0 \cup 1)(00 \cup 01 \cup 10 \cup 11)^*$

✓ $(0 \cup 1)((0 \cup 1)(0 \cup 1))^*$

✓ $(00 \cup 01 \cup 10 \cup 11)^*(0 \cup 1)$

The set of all binary strings with at most two ones.

$0^*(1 \cup \varepsilon)0^*(1 \cup \varepsilon)0^*$

The set of all binary strings with equal number of 0s and 1s.

NOT POSSIBLE

Applications of Regular Expressions

An aside: Regular Expressions are used everywhere, outside of CS Theory too.

- Some search tools allow you to search for regex's instead of strings
  E.g. search for "$(a \cup \cdots \cup z \cup 1 \cup \cdots \cup 9)^*\mathbf{@uw.edu}$".
- Used in $\mathbf{grep}$, a program that does pattern matching searches in LINUX.
- Used to define the "tokens": e.g. legal variable names, keywords, in programming languages and compilers.
- Many implementations of RegExs are more powerful than our theoretical Regular Expression


WHENEVER I LEARN A NEW SKILL I CONCOCT ELABORATE FANTASY SCENARIOS WHERE IT LETS ME SAVE THE DAY.
OH NO! THE KILLER MUST HAVE FOLLOWED HER ON VACATION!
BUT TO FIND THEM WE'D HAVE TO SEARCH THROUGH 200 MB OF EMAILS LOOKING FOR SOMETHING FORMATTED LIKE AN ADDRESS!
IT'S HOPELESS!
EVERYBODY STAND BACK.
I KNOW REGULAR EXPRESSIONS.
PERL!
TAP TAP

Regular Languages

Definitions:

Regular Languages are languages that can be specified by a regular expression.

Irregular Languages are languages that are not regular

Irregular Languages

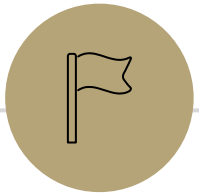
It turns out a lot of useful languages are irregular.

- Binary strings with an equal number of 0s and 1s
- Palindromes (strings that read the same forwards and backwards)
- Matched parentheses, e.g. ((())())
- Properly formed arithmetic expressions

racecar

$(1+5\cdot3)\cdot2 \in L$

$(1(5\cdot3)\cdot2 \notin L$

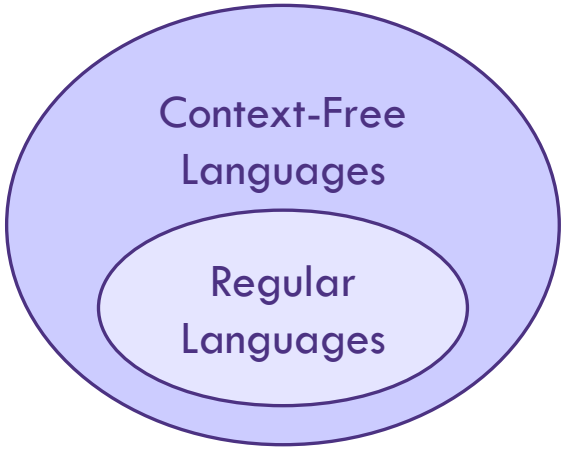


Context Free Languages

Another class of languages

Context-Free Languages

- We just saw some limitations of Regular Languages
- Context-Free Languages are a strictly larger class of languages



- Context-Free Languages are generated by Context-Free Grammars

  (just like Regular Languages are specified by Regular Expressions)

Context-Free Grammars (CFGs)

- A Context-Free Grammar is a finite set of production rules, involving:
  - Alphabet of terminal symbols (e.g. 0, 1, a, b, ε)
  - A finite set of nonterminal symbols (e.g. A, B, S, T, R)
  - One special nonterminal called the start symbol, usually S

- A production rule for a nonterminal A takes the form:

$$A \rightarrow w_1 \mid w_2 \mid \dots \mid w_k$$

where each w_i is a string of terminals & nonterminals

Context-Free Grammars (CFGs)

For example:

$S \rightarrow Ab \mid c$

$A \rightarrow Aa \mid \varepsilon$

- 2 production rules:
  $S \rightarrow Ab \mid c$
  $A \rightarrow Aa \mid \varepsilon$
- 2 nonterminals: S, A
- 4 terminals: a, b, c, ε

Context-Free Grammars

{ c, b, ab, aab, aaab, ... }

For Example:
S → Ab | c
A → Aa | ε

We think of Context-Free Grammars as **generating** strings.

1. Start from the start symbol S.
2. Choose a nonterminal, e.g. S, in the string, and replace it by one of the w's in the rules for S

$$S \rightarrow w_1 \mid w_2 \mid \ldots \mid w_k$$

S ⇒ Ab ⇒ Aab
⇒ Aaab ⇒ aab

3. Repeat step 2 until there are no nonterminals left.

The language that the CFG describes is the set of all strings that it generates.

Example

$S \rightarrow 0S \mid S1 \mid \varepsilon$

Example

$S \rightarrow 0S0 \mid 1S1 \mid 0 \mid 1 \mid \varepsilon$

Example

CFG for the language $\{0^n 1^n : n \geq 0\}$

Example

CFG for the language $\{0^n 1^n 23 : n \geq 0\}$

Exercises

CFG for the set of binary strings with the same number of 0s as 1s.

CFG for the set of balanced parentheses. E.g. (((())())())