Regular Expressions

2020-9-11

Pattern Matching

- Pattern matching is a problem as old as computer science
- We've devised a number of mechanisms for pattern matching

Pattern Matching

- Suppoose we have a text document from which we'd like to extract the dates in the following format

```
2020-03-24
1996-04-09
1995-12-03
```

- YYYY-MM-DD

Regular Expressions

- Regexes provide convenient syntax for describing patterns to match
- A regex describes a **set of strings**
- The regex **recognizes** this set of strings

Regular Expressions

- Set of strings called a **language**
- Set of languages recognizable by regular expressions called **regular languages**
- Describable with only three operations

Regular Expression Operations

Operator	Symbol*	Description
Concatenation	$\circ$	$a \circ b$ means b must follow a; often written ab.
Union	$\cup$	Combine languages L_1 and L_2
Star	*	A* recognizes A repeated 0 or more times

- Concatenation symbol usually omitted.

Regular Expression Operations

Operator	Symbol*	Description
Concatenation	$\circ$	$a \circ b$ means b must follow a; often written ab.
Union	$\cup$	Combine languages L_1 and L_2
Star	*	A* recognizes A repeated 0 or more times

- Concatenation symbol usually omitted.
- Many more shorthand symbols used
  - `+` for one or more of a symbol
  - superscripts for multiple concatenations, e.g., $a^5 = aaaaa$

Language	Examples	Description
$L_2 = ab$	ab	ab
$L_3 = ab^*$	a, ab, abb, abbb	a, followed by 0 or more b's
$L_4 = (ab)^*$	ϵ, ab, abab, ababab	ab repeated 0 or more times
$L_5 = a \cup b$	a, b	a or b
L_6 = $a \cup b$*	ϵ, a, b, bb, bbb	a, or any number of b's
$L_7 = a \cup bb^*$	a, b, bb, bbb	a or 1 or more b's
$(L_1 \cup L_2)^*$	{ϵ, a, ab, aab, aba, ababa}	Strings from {a, ab} repeated 0 or more times

Regular Expressions

- Many progamming languages (Java, Python, Perl) have built-in regex support
- Several common Unix tools (`grep`, `sed`, `awk`) use regexes.
  - Consult documentaiton

Regular Expressions

- In `grep` and many other languages, we use `[0-9]` for number in $\{0, 1, 2, 3, \ldots, 10\}$,
- `[A-Za-z]` for alphabetical characters
- Many more

Regular Expressions

- Let's return to date example.

```
[0-9][0-9][0-9][0-9]-[0-9][0-9]-[0-9][0-9]
```

Regular Expressions

- We can use shortcuts.
- Let's return to date example.

```
[0-9][0-9][0-9][0-9]-[0-9][0-9]-[0-9][0-9]
```

In grep, for example:

```
[0-9]\{4\}-[0-9]\{2\}-[0-9]\{2\}
```

Simple Morphology

- Among other things, morphology is concerned with word inflections
- Consider *compute, computer, computational, computational*
- How might we capture these in a `grep` regex?

Simple Morphology

```
grep "comput\(e|er\|ation\|ational\)"
```

- All finite languages are regular
- Regexes are powerful but have limitations
- Infinitely many non-regular languages
  - $L = a^i b^i$

Finite State Automata

- A finite state automaton (FSA) is an abstract computational model
  - Studied in depth in Theory of Computation
- Graphically, consists of **states** (squares or rectangles) and **transitions** (arrows).
- Deterministic and non-deterministic automata
  - Deterministic FSA called **DFA**
  - Nondeterministic FSA called **NFA**

Finite State Automata

- In a deterministic finite state automaton, the machine can be in exactly one state (circle or square) at a given time.
- From the state, it can follow a **transition** (arrow) to another state.

<center> <div class="mermaid"> graph LR 1--a-->2 classDef orange fill:#FFFFF,stroke:,stroke-width:4px; class 2 orange </div>

Finite State Automata

- NFAs introduce ϵ-transitions, which consume empty string ϵ.
- In an NFA, we only need one valid path to an acceptance state for an input string.
- NFAs and DFAs recognize the same languages, so we'll use NFAs

Finite State Automata

NFA for language ab^*

<center> <div class="mermaid"> stateDiagram-v2 [*] --> 1 : ϵ 1 --> 2 : a 2 --> 3 : b 3 --> 2 : ϵ 3 --> [*] : ϵ 2 --> [*] : ϵ </div> <center>

Finite State Automata

- If we cannot reach an accepting state, the input string is rejected
  - String is outside of the language

Finite State Automata

- Bottom line:
  - The set of languages recognized by NFAs is the same as the set of languages recognized by DFAs.
  - The set of languages recognized by DFAs is the same as the set of languages recognized by regexes.
  - If there exists a regex for a set of strings, there exists an NFA and a DFA for it, as well.
  - There are many languages unrecognizable by regexes, e.g., a^ib^i.