Lec 9 CS241

Graham Cooper

June 3rd, 2015

Regular Languages

built from:

- finite languages
- union
- concatenation
- repetition

- Union of two languages:
$L_1 \cup L_2 = \{x | x \in L_1 or x \in L_2\}$
- Concatenation of two languages: $L_1 \cdot L_2 = \{xy | x \in L_1, y \in L_2\}$
eg: $L_1 = \{dog, cat\}$ $L_2 = \{fish, \epsilon\}$ $L_1 L_2 = \{dogfish, dog, catfish, cat\}$

- Repetition: $L^* = \{\epsilon\} \cup \{xy | x \in L^*, y \in L\}$
$= \{\epsilon\} \cup L \cup LL \cup LLL \cup ....$
= 0 or more occurences of a word in L

EG: L = {a,b}
$L^* = \{\epsilon, a, b, aa, ab, bb, ba, aaa, aab, abb, ...\}$
Show: $\{2^{2n} b | n \geq 0\}$ is regular
$(\{aa\})^* \cdot \{b\}$

Shorthand - regular expression.

Language	Regular Expression	name	
$\{\}$	$\emptyset$	empty language	
$\{\epsilon\}$	ϵ	language consisting of the empty word	
{aaa}	aaa	singleton language	ie:
$L_1 \cup L_2$	$L_1 \| L_2$	alternation (union)	
$L_1 L_2$	$L_1 L_2$	concatenation	
L^*	$L*$	repetition	

$\{a^{2n} b | n \geq 0\} = (aa) * b$

Is C regular?

A C program is a sequence of tokens.

A C program is a sequence of tokens, each of which comes from a regular language. C $\subseteq \{validctokens\}^*$ maybe.

How can we recognize an arbitrary regular languae automatically?

Eg. $\{a^{2n}b|n \geq 0\} = (aa)^*b$
Can we harness what we leanred about finite languages?
- Yes if we allow loops in the diagram

start $\overset{a}{\underset{a}{\rightleftarrows}}$ state
$\underset{b}{\downarrow}$
finished state

Set of MIPS labels

start $\overset{a-z|A-Z}{\rightarrow}$ state(loops with a-z|A-Z |0-9) $\overset{:}{\rightarrow}$ finished state

Deterministic Finite Automata

These "machines" (state diagrams) are called Deterministic Finite Automata (DFAs)

- always start at the start state
- for each character in the input, follow the corresponding arc to the next state
- if on an accepting state when the input is exhuasted, you accept, else you reject.

What if there is no transition? start $\underset{a}{\overset{a}{\rightleftarrows}}$ state
$\downarrow_b$
finished state

If you exit at the middle (not finished or start) state, you fall off the machine and are rejected.
How to reject:

- you fall off of the machine
- you are not in an accepting state at end of input

More formally:
There is an implicit "error state", all unlabeled transitions go to this error state.

Example: Strings over {a,b} wit han even number of a's and an odd number of b's

$\rightarrow$ start state $\xrightarrow{b}$ accept state
$\downarrow_a \uparrow_a$
mid state $\overset{b}{\rightleftarrows}$ midstate (odd a, odd b) (this goes up to the accept state back and forth with a's)

Formal Definition of a DFA

A DFA is a 5-tuple $(\Sigma, Q, q_0, A, \delta)$

- Σ is a finite, non-empty set (alphabet)
- Q is a finite non-empty set (states)
- $q_0 \in Q$ (start state)
- A $\subseteq$ Q (accepting states)

- δ takes $q \times \Sigma \to Q$ (transition function: maps state + input character to next state)

δ consumes one character, can extend δ to a function that consumes an entire word:

<u>Definition:</u>
$\delta^*(q, \epsilon) = q$
$\delta^*(q, cw) = qc$

We say a DFA$(\Sigma, Q, q_0, A, \delta)$ accepts a word w if:
$\delta^*(q_0, w) \in A$

If M is a DFA, we denote by L(M) ("The language of M"), the set of all strings accepted by M
L(M) = {w | M accepts w}

Theorem: Kleene

L is regular iff L = L(M) for some DFA M. (The regular languages are the languages accepted by DFA's.)