CS241 Lecture 11

Graham Cooper

June 10th, 2015

See notes for NFA

NFA Trace: caba

read	unread	states
ϵ	caba	{1}
c	aba	{2,6}
ca	ba	{3,5}
cab	a	{4,5}
caba	ϵ	{6}

Build a DFA via the subset construction, see note

Accepting: any set that includes an accepting state from the original NFA.

Obvious Fact: Every DFA is implicitly an NFA.

Also: every NFA can be converted to a DFA for the same language

So: DFA's and NFA's accept the same class of languages

ϵ-NFA's

What if we let ourselves change states without reading a character?

ϵ-transitions: state $\xrightarrow{\epsilon}$ state
- "free pass" to a new state without reading a character
- makes it easy to glue smaller automata together

See picture of ϵ-NFA on notes

Read	Unread	states
ϵ	caba	{1,2,6}
c	aba	{3,6}
ca	ba	{4,7}
cab	a	{5, 7}
caba	ϵ	{6}

$\therefore$ By the same renaming trick as before, every ϵ-NFA has an equivalent DFA.
$\therefore$ NFA's recognize the same class of languages as DFA's
- The converstion can be automated

If we can find an ϵ-NFA for every regular expression, then we have one direction of kleene's theorem. (Regular expression $->$ ϵ-NFA $->$ DFA)

Regular Expression Types

1. $\emptyset$ - ϵ-NFA: non-accepting starting state.
2. ϵ - ϵ-NFA: Single accepting starting state.
3. a - ϵ-NFA: non-accepting starting state $\xrightarrow{a}$ accepting state
4. $E_1|E_2$ - ϵ-NFA: See paper
5. E_1E_2 - ϵ-NFA: See paper, E1's accepting states go to E2 and are no longer accepting.
6. E^* - ϵ-NFA: See Paper

$\therefore$ every regular expression has an equivalent ϵ-NFA, and $\therefore$ an equivalent DFA. and the conversion can be automated ie. can write tools to convert regular expressions to DFA's.

Scanning

Is C a regular language? Well, C keywords:

- ids

- literals
- operators
- commends
- punctiation

$\therefore$ sequences of these are also regular since the above are all regular

So we can use finite automata to do scanning (tokenization). Ordinary DFA's answer yes/no to $w \in L$?
We need:
- Input string w
- break into $w_1, w_2, ...w_n$ such that each $w_i \in L$ else error
- output each w_i

Consider: L = {valid c tokens} is regular. Let M_L = some machine, be the DFA that recognizes L
Then, M_L (with ϵ transitions going from end states to start state) recognizes LL^*

Add an action to each ϵ move (See seperate paper)
- The machine is now non-deterministic because ϵ moves are always optional.

SO: does this scheme represent a unique decomposition w = $w_1, w_2...w_n$?

NO! Consider the id portion: (See paper)
input abab could be interpreted as 1,2,3 or 4 tokens

What do we do about this?
- Decide to take the ϵ move only if there is no other choice.
$\Longrightarrow$ always return the longest possible next token
- could mean that valid matches are missed
- consider L = {aa, aaa}, w = aaaa
– If you take the longest token first, aka aaa, then you will only be left with a which is not a token.
– The a left over cannot be matched :(poor a.