Lecture 9

Regular expression matching

The *regular expressions* over an alphabet Σ are the strings over the alphabet $\Sigma \cup \{(,),\emptyset,|,*\}$ such that the following hold

- $\emptyset$, ϵ, and a character in Σ are regular expressions.
- If x and y are regular expressions, so is (xy) – concatenation.
- If x and y are regular expressions, so is $(x|y)$ – OR.
- If x is a regular expression, so is x^*, where $x^* = \epsilon|x|x^2|x^3|\ldots$

For example,

```
(hot | cold)(apple | blueberry | cherry) pie
the (very)^* hot apple pie
(a|b)^* a
a (a|b)^* b
```


Definition 1 *A language (set of strings) is regular if it is described by a regular expression.*

Theorem 1 *A language is regular if and only if it is accepted by a finite automaton.*

Regular expression matching: Given a regular expression R (all patterns represented by R) and a text T, find all patterns of R in T.

- Similar to multiple keyword matching
- Difference: size of input is $|R| = m$ and $|T| = n$. It's not the sum of all pattern sizes. In fact, the number of patterns can be infinite.

Example

```
R=(a|b)^*aba,  T=abcaabaabaabc
two longest occurrences: aaba, aabaaba
|R| = 9
```

Applications: Unix commands grep family

We prepend to R the expression $(a_1|a_2|...|a_z)^*$, where $a_1, \ldots, a_z$ are all symbols of the input alphabet Σ. This prefix allows matching to begin at any position in T. So we will find end positions of occurrences. Assume that R contains this prefix.

Finite automata

- Deterministic FA: every state has (at most) one transition on any input character.
- Nondeterministic FA:
  - a state may have more than one transition on an input character.
  - ϵ transitions

  An NFA accepts a string if there is at least ONE path from the start state to an accepting state whose edge labels spell out the string.

Running an FA with an input string

- ∘ At a time, a DFA has only one current state.
- ∘ An NFA has a set of current states.

Example, p58 of LP: accepts the set of all strings containing an occurrence of bab or baab.

Two approaches to regular expression matching

1. Build an NFA, and search T with the NFA.
2. Build a DFA, and search T with the DFA.

- Building FA: 1 is easier (size of DFA may be exponential)
- Search: 2 is easier

Thompson's algorithm (approach 1)

1. Construct an NFA from R recursively. See pp22,23 of Aho.

- a character x
- $r_1|r_2$
- $r_1 r_2$

- r^*
- (r)

Example $(a|b)^*aba$

An NFA N constructed as above has the following properties.

- The number of states in N is $\leq 2|R|$, since Steps A,B,D create at most two new states.
- N has one start state and one accepting state, and the accepting state has no outgoing transitions. This property holds for each of the sub-NFAs as well.
- Each state has either one outgoing edge labeled by a character or at most two outgoing ϵ edges.

2. Search T with the NFA.

Run NFA N on input string T.

Let i, f be the initial and acceping states of N.

- epsilon(Q'): all states that can be reached from a set of states Q' by following only ϵ edges.
- goto(Q, x): all states that can be reached from a state in Q by a transition on x.

◦ Q : the set of current states of N.
◦ j : the current text position.
◦ Initially, Q = epsilon($\{i\}$), and current text position is 1.

How to compute the next set of current states.

1. Q' = goto(Q,T[j])
2. Q = epsilon(Q')

```
Search(N,T)
  Q = epsilon({i})
  if Q contains f then report "yes" fi
  for j = 1 to n do
    Q' = goto(Q,T[j])
    Q = epsilon(Q')
    if Q contains f then report "yes" fi
  od
end
```


Time Analysis

- Let $|N|$ denote the number of states in N.
- Each of Q and Q' contains at most $|N|$ states.

Since each state has at most one outgoing transition by a character, each state in Q adds at most one new state into Q'. Need to determine whether a state is already in Q'.

- Use arrays of size $|N|$ whose indices are state numbers for Q and Q'. Then, goto(Q,x) takes $O(|N|)$ time.
- For epsilon(Q'), use a reachability algorithm (DFS or BFS) (follow ϵ transitions without overlap). It takes $O(|N|)$ time.

The overall time is $O(|N|n)$. Since $|N| \leq 2m$, the algorithm takes $O(mn)$.