More Closure Properties, Algorithms, and the Pumping Lemma

CS154

Chris Pollett

Feb 21, 2007.

Outline

- Closure under Quotients and Homomorphisms
- Algorithms for membership, emptiness, finiteness, and equality
- Determining if a language is regular

Intro to Homomorphisms

- From time to time it is useful to be able to translate things from one language to another.
- In the simplest case this might involve transliterating character by character, as when we convert Russian spellings of proper names into English. For example, СТАЛИН --> STALIN.
- As another example, we might want to encode our alphabet using an error correcting code.
- In converting from one such language to another we have to be sensitive to the fact that sometimes an exactly equivalent string might not exist. For example, in English we have the words blue and turquoise which in some other language might both be translate to blue.
- Homomorphisms allow us to do these kind of mappings for regular languages.

Definition of Homomorphism

Given two alphabets Σ and Σ'. A function

$$h:\Sigma \dashrightarrow (\Sigma')^*$$

is called a **homomorphism**. The domain of h can be extended to all strings over Σ^* as follows: if $w=a_1a_2..a_n$ then $h(w) = h(a_1)h(a_2)\ldots h(a_n)$.

Given a language L, its **homomorphic image** is defined to be $h(L) = \{h(w) \mid w \in L\}$.

Example

- For example strings over {a,b} might be encoded to strings over {0,1} via the homomorphism: h(a) = 000, h(b) = 111.
- In this case the language L={aa, baba} has as its homomorphic image: h(L) = {000000, 111000111000}.
- A homomorphism does not have to be one to one. Could map {a, b} to the alphabet {a} via h(a)=a, h(b) =a. In which case h(ababa) = aaaaa.

Closure under Homomorphism

Theorem. Let L be a regular language over ∑ and let h:∑-->(∑´)* be a homomorphism. Then h(L) is a regular language over ∑´.

Proof. We have shown that every regular language can be represented by a regular expression. Let R be the regular expression for L. We prove by induction on the complexity of R that h(L) will be regular. In the base R is either a symbol *a* of ∑ or it is the empty string, or it is the empty set. In the latter two cases L(R) = L(h(R)), so we are done. In the first case, we note that h(a) is a string over ∑´ and so will be a regular expression over the ∑´ alphabet. for the induction step, R is either of the form $R = (R_1 \cup R_2)$, $R = (R_1R_2)$, or $R = (R_1)^*$. In each of these cases, we have by the induction hypothesis a regular expressions $R´_1$ and $R´_2$ for the homomorphic images of the languages of the subexpressions. So to make regular expressions for the homomorphic image of the language for R we can take either: $R´ = (R_1´ \cup R_2´)$, $R´ = (R_1´R_2´)$, or $R´ = (R_1´)^*$.

Quotients

- A common problem in the computer processing of natural languages is to come up with “stems” of a given sequence of words.
- For example, if we do a Google search on fished, fishing, fishes, etc. as a preprocessing step this might be stemmed to just the word “fish”.
- We will next consider a notion of the quotient of two languages which allows us to formally consider things like stemming.

Definition. If A and B are two languages, their quotient A/B is the language: {v | vw is in A and w is in B}.

- So if A={fished, fish, fishes, fishing, jumping, oranges} and B={ing, ed} then A/B = {fish, jump}.

Closure under Quotients

Theorem. If A and B are regular languages, then A/B is also a regular language.

Proof. Let $M=(Q,\Sigma, \partial, q_0, F)$ be a DFA for A and let M' be a DFA for B. So a string v is in $A/B = L(M)/L(M')$ if $\partial^*(q_0,v)=q_i$ for some i, $\partial^*(q_i,w) \in F$, and $w \in L(M')$. Let $M_i = (Q,\Sigma, \partial, q_i, F)$, then $L(M_i) \cap L(M')$ is regular from last day. Notice $L(M_i) \cap L(M')$ is nonempty iff the two conditions $\partial^*(q_i,w) \in F$, and $w \in L(M')$ hold for some w. Further, we can check if $L(M_i) \cap L(M')$ is nonempty by seeing if the some accepting state of this language is reachable from the start state. Hence, we can make a machine for A/B as $(Q,\Sigma, \partial, q_0, F')$ where F' are those state in Q such that $L(M_i) \cap L(M')$ is nonempty.

Algorithms for regular languages

- We now briefly present some algorithms for checking various properties of regular languages.
- We say a regular language is in **standard representation** if it is represented either as a DFA, as an NFA, a regular expression, or as a regular grammar.
- We know we can convert via an algorithm any of these forms to any other.

Membership, Emptiness and Finiteness Checking

Theorem Given a regular language L in standard representation and a string w, there are algorithms which can check: (a) if w is in L, (b) if L is empty, and (c) if L is finite.

Proof. The first step of each algorithm is to obtain a DFA for L.

(a)
```
cur_state = start_state;
for(i=0; i<w.length; i++)
{
  …
  //handle in ∂(q,a) =q´
  if(cur_state== q &&
    w.charAt(i) == ‘a’)
  {
     cur_state = q´;
   }
…//handle other case
}
if(cur_state in final_states) return accept;
else return reject;
```

(b)
```
foreach( state in final_states)
{
  if(isReachable(start_state,  state)
  {
     /* isReachable checks if the
       second vertex is reachable from
       the first vertex by a simple
       path in the graph */

      return not_empty;
  }
}
return empty;
```

(c)
```
foreach( state in dfa_states)
{
  if( !(isReachable(start_state,  state) &&
        isReachable(state, state)) break;

  foreach(fstate in final_states)
  {
     if(isReachable(state, fstate))
     {
       return language_is_infinite;
     }
  }
}
return language_is_finite;
```