CSE 2001: Introduction to Theory of Computation

Fall 2007

Suprakash Datta
datta@cs.yorku.ca

Course page: http://www.cs.yorku.ca/course/2001

Announcements

- TA's will cover office hours (see web page for locations/times)
- Please hand in Assignment 1
- Please take a copy of Assignment 2

Regular Operations

Pages 44-47 (Sipser)

The regular operations are:

1. Union
2. Concatenation
3. Star (Kleene Closure): For a language A, $A^* = \{w_1 w_2 w_3 \ldots w_k \mid k \geq 0, \text{ and each } w_i \in A\}$

Closure Properties

- Set of regular languages is closed under
  - Union
  - Concatenation
  - Star (Kleene Closure)

Union of Two Languages

Theorem 1.12: If A_1 and A_2 are regular languages, then so is $A_1 \cup A_2$.
(The regular languages are 'closed' under the union operation.)

Proof idea: A_1 and A_2 are regular, hence there are two DFA M_1 and M_2, with $A_1=L(M_1)$ and $A_2=L(M_2)$. Out of these two DFA, we will make a third automaton M_3 such that $L(M_3) = A_1 \cup A_2$.

Proof Union-Theorem (1)

$M_1=(Q_1,\Sigma,\delta_1,q_1,F_1)$ and $M_2=(Q_2,\Sigma,\delta_2,q_2,F_2)$

Define $M_3 = (Q_3,\Sigma,\delta_3,q_3,F_3)$ by:

- $Q_3 = Q_1 \times Q_2 = \{(r_1,r_2) \mid r_1 \in Q_1 \text{ and } r_2 \in Q_2\}$
- $\delta_3((r_1,r_2),a) = (\delta_1(r_1,a), \delta_2(r_2,a))$
- $q_3 = (q_1,q_2)$
- $F_3 = \{(r_1,r_2) \mid r_1 \in F_1 \text{ or } r_2 \in F_2\}$

Proof Union-Theorem (2)

The automaton $M_3 = (Q_3,\Sigma,\delta_3,q_3,F_3)$ runs M_1 and M_2 in 'parallel' on a string w.

In the end, the final state (r_1,r_2) 'knows' if $w \in L_1$ (via $r_1 \in F_1$?) and if $w \in L_2$ (via $r_2 \in F_2$?)

The accepting states F_3 of M_3 are such that $w \in L(M_3)$ if and only if $w \in L_1$ *or* $w \in L_2$, for: $F_3 = \{(r_1,r_2) \mid r_1 \in F_1 \text{ or } r_2 \in F_2\}$.

Closure under intersection

- Modify the previous proof.

Concatenation of L_1 and L_2

Definition: $L_1 \bullet L_2 = \{ xy \mid x \in L_1 \text{ and } y \in L_2 \}$

Example: $\{a,b\} \bullet \{0,11\} = \{a0,a11,b0,b11\}$

Theorem 1.13: If L_1 and L_2 are regular langues, then so is $L_1 \bullet L_2$.
(The regular languages are 'closed' under concatenation.)

Proving Concatenation Thm.

Consider the concatenation:
{1,01,11,001,011,…} • {0,000,00000,…}
(That is: the bit strings that end with a "1", followed by an odd number of 0's.)

Problem is: given a string w, how does the automaton know where the L_1 part stops and the L_2 substring starts?

We need an M with 'lucky guesses'.

Guessing machines?

- Non-deterministic Finite Automata.

Closure under regular operations

Union (new proof):

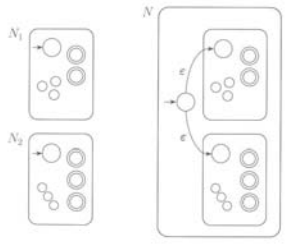


FIGURE 1.46
Construction of an NFA N to recognize $A_1 \cup A_2$

Closure under regular operations

Concatenation:

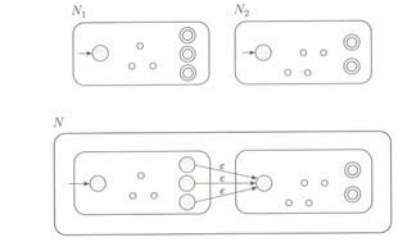


FIGURE 1.48
Construction of N to recognize $A_1 \circ A_2$

Closure under regular operations

Star:

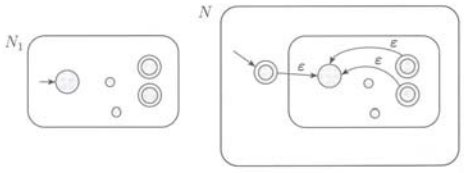


FIGURE 1.50
Construction of N to recognize A^*

More on regular languages

- Built up using regular operations.
- Recall: Set of regular languages is closed under
  - Union
  - Concatenation
  - Star (Kleene Closure)

Incorrect reasoning about RL

- Since $L_1 = \{w |\ w=a^n, n \in \mathbf{N}\}$, $L_2 = \{w |\ w = b^n, n \in \mathbf{N}\}$ are regular, therefore $L_1 \bullet L_2 = \{w |\ w=a^n b^n, n \in \mathbf{N}\}$ is regular
- If L_1 is a regular language, then $L_2 = \{w^R |\ w \in L_1\}$ is regular, and Therefore $L_1 \bullet L_2 = \{w\, w^R \mid w \in L_1\}$ is regular

Regular Expressions (Def. 1.52)

Given an alphabet Σ, R is a regular expression if:
(INDUCTIVE/RECURSIVE DEFINITION)

- $R = a$, with $a \in \Sigma$
- $R = \varepsilon$
- $R = \varnothing$
- $R = (R_1 \cup R_2)$, with R_1 and R_2 regular expressions
- $R = (R_1 \bullet R_2)$, with R_1 and R_2 regular expressions
- $R = (R_1^*)$, with R_1 a regular expression

Precedence order: *, $\bullet$, $\cup$

Regular Expressions

- Unix 'grep' command: Global Regular Expression and Print
- Lexical Analyzer Generators (part of compilers)
- Both use regular expression to DFA conversion

Languages corresponding to regular expressions

Examples

- $e_1 = a \cup b$, $L(e_1) = \{a,b\}$
- $e_2 = ab \cup ba$, $L(e_2) = \{ab,ba\}$
- $e_3 = a^*$, $L(e_3) = \{a\}^*$
- $e_4 = (a \cup b)^*$, $L(e_4) = \{a,b\}^*$
- $e_5 = (e_m . e_n)$, $L(e_5) = L(e_m) \bullet L(e_n)$
- $e_6 = a^*b \cup a^*bb$,
  $L(e_6) = \{w |$ w $\in \{a,b\}^*$ and w has 0 or more a's followed by 1 or 2 b's$\}$

Thm 1.54: RL ~ RE

We need to prove both ways:
• If a language is described by a regular expression, then it is regular (Lemma 1.55)
(We will show we can convert a regular expression R into an NFA M such that L(R)=L(M))

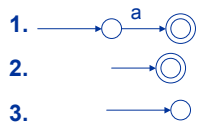
• The second part:
If a language is regular, then it can be described by a regular expression (Lemma 1.60)

Regular expression to NFA

Claim: If L = L(e) for some RE *e*, then L = L(M) for some NFA M

Construction: Use inductive definition

1. R = a, with a$\in\Sigma$
2. R = ε
3. R = $\varnothing$
4. R = $(R_1 \cup R_2)$, with R_1 and R_2 regular expressions
5. R = $(R_1 \bullet R_2)$, with R_1 and R_2 regular expressions
6. R = (R_1^*), with R_1 a regular expression



4,5,6: similar to closure of RL under regular operations.

Examples of RE to NFA conv.

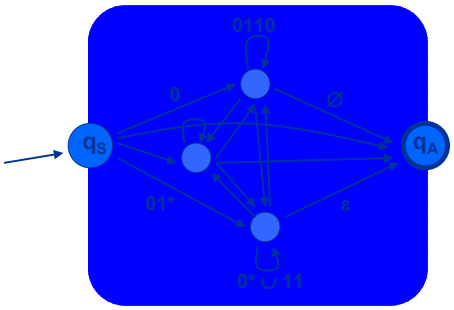
L = {ab,ba}
L = {ab,abab,ababab,……}
L = {w | w = $a^m b^n$, m<10, n>10}

Back to RL ~ RE

- The second part (Lemma 1.60):
  If a language is regular, then it can be described by a regular expression.
- Proof strategy:
  - regular implies equivalent DFA.
  - convert DFA to GNFA (generalized NFA)
  - convert GNFA to NFA.

GNFA: NFA that have regular expressions as transition labels

Example GNFA



Generalized NFA - defn

Generalized non-deterministic finite automaton $M=(Q, \Sigma, \delta, q_{start}, q_{accept})$ with

- Q finite set of states
- Σ the input alphabet
- q_{start} the start state
- q_{accept} the (unique) accept state
- $\delta:(Q - \{q_{accept}\})\times(Q - \{q_{start}\}) \rightarrow \mathcal{R}$ is the transition function

($\mathcal{R}$ is the set of regular expressions over Σ)

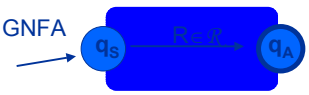
(NOTE THE NEW DEFN OF δ)

Characteristics of GNFA's δ

- $\delta:(Q\backslash\{q_{accept}\})\times(Q\backslash\{q_{start}\}) \rightarrow \mathcal{R}$

The interior $Q\backslash\{q_{accept},q_{start}\}$ is fully connected by δ
From q_{start} only 'outgoing transitions'
To q_{accept} only 'ingoing transitions'
Impossible $q_i \rightarrow q_j$ transitions are labeled "$\delta(q_i,q_j) = \emptyset$"

Observation: This GNFA recognizes the language L(R)



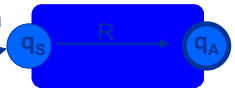
Proof Idea of Lemma 1.60

Proof idea (given a DFA M):

Construct an equivalent GNFA M' with $k \geq 2$ states

Reduce one-by-one the internal states until k=2

This GNFA will be of the form

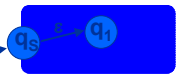


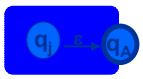
This regular expression R will be such that L(R) = L(M)

DFA M $\rightarrow$ Equivalent GNFA M'

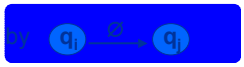
Let M have k states $Q=\{q_1,\ldots,q_k\}$

- Add two states q_{accept} and q_{start}
- Connect q_{start} to earlier q_1:

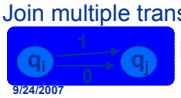




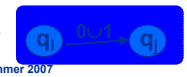
- Connect old accepting states to q_{accept}
- Complete missing transitions by



- Join multiple transitions:



becomes

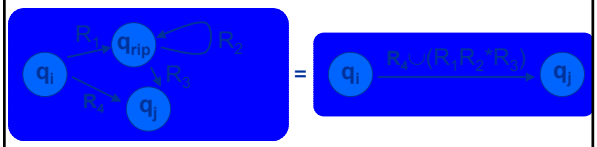


Remove Internal state of GNFA

If the GNFA M has more than 2 states, 'rip' internal q_{rip} to get equivalent GNFA M' by:

- Removing state q_{rip}: $Q'=Q\backslash\{q_{rip}\}$
- Changing the transition function δ by

$\delta'(q_i,q_j) = \delta(q_i,q_j) \cup (\delta(q_i,q_{rip})(\delta(q_i,q_j))^*\delta(q_{rip},q_j))$
for every $q_i \in Q'\backslash\{q_{accept}\}$ and $q_j \in Q'\backslash\{q_{start}\}$



Proof Lemma 1.60

Let M be DFA with k states

Create equivalent GNFA M' with k+2 states

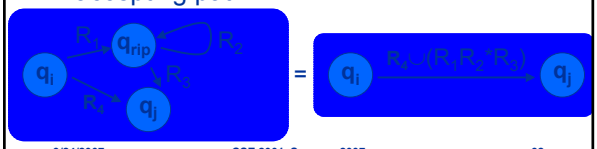
Reduce in k steps M' to M'' with 2 states

The resulting GNFA describes a single regular expressions R

The regular language L(M) equals the language L(R) of the regular expression R

Proof Lemma 1.60 - continued

- Use induction (on number of states of GNFA) to prove correctness of the conversion procedure.
- Base case: k=2.
- Inductive step: 2 cases – q_{rip} is/is not on accepting path.



Recap RL = RE

Let R be a regular expression, then there exists an NFA M such that L(R) = L(M)

The language L(M) of a DFA M is equivalent to a language L(M') of a GNFA = M', which can be converted to a two-state M''

The transition $q_{start} \xrightarrow{R} q_{accept}$ of M'' obeys L(R) = L(M'')

Hence: RE $\subseteq$ NFA = DFA $\subseteq$ GNFA $\subseteq$ RE

Example

L = {w| the sum of the bits of w is odd}

Non-regular Languages §1.4

Which languages cannot be recognized by finite automata?

Example: L={ 0^n1^n | $n \in N$ }
- 'Playing around' with FA convinces you that the 'finiteness' of FA is problematic for "all $n \in N$"
- The problem occurs between the 0^n and the 1^n
- Informal: the memory of a FA is limited by the the number of states |Q|