

Examples of Regular Expressions

1. 0^*10^* , $L(0^*10^*) = \{w \mid w \text{ contains exactly a single } 1\}$
2. $\Sigma^*1\Sigma^*$, $L(\Sigma^*1\Sigma^*) = \{w \mid w \text{ contains at least one } 1\}$
3. $\Sigma^*001\Sigma^*$, $L(\Sigma^*001\Sigma^*) = \{w \mid w \text{ contains the string } 001 \text{ as a substring}\}$
4. $(\Sigma\Sigma)^*$, $L(\Sigma\Sigma)^* = \{w \mid w \text{ is a string of even length}\}$
5. $(\Sigma\Sigma\Sigma)^*$, $L(\Sigma\Sigma\Sigma)^* = \{w \mid \text{the length of } w \text{ is a multiple of three}\}$
6. $01 \cup 01$, $L(01 \cup 01) = \{01, 01\}$
7. $0\Sigma^*0 \cup 1\Sigma^*1 \cup 0 \cup 1$, $L(0\Sigma^*0 \cup 1\Sigma^*1 \cup 0 \cup 1) = \{w \mid w \text{ starts and ends with the same symbol}\}$
8. $(0 \cup \epsilon)1^*$, $L((0 \cup \epsilon)1^*) = \{01^* \cup 1^*\}$
9. $(0 \cup \epsilon)(1 \cup \epsilon)$, $L((0 \cup \epsilon)(1 \cup \epsilon)) = \{\epsilon, 0, 1, 01\}$

Computation Theory – p.2/??

Finite Automata vs. Regular Expressions

Hantao Zhang

hzhang@cs.uiowa.edu

The University of Iowa, Department of Computer Science

Based on the slides prepared by Professor Rus

Computation Theory – p.1/??

Example of Using flex

```
DIGIT      [0-9]
LETTER     [A-Za-z_]
%%
"|"|"|"|"|" { return(*yytext); }
{DIGIT}({DIGIT})*
  { get_integer(*yytext); return(INTEGER); }
{LETTER}({LETTER}|{DIGIT})*
  { get_symbol(*yytext); return(SYMBOL); }
[ \t]+     { /* jump over blanks */ }
\n        { /* ignore newlines */ }
. { report_error("Unknown symbol", *yytext); }
%%
```

Save the above text in a file called `try.1` and type

```
flex -i try.1 > try.c
```

Computation Theory – p.4/??

Application

- Regular expressions are useful tools for the design of compilers
- Language lexicon is described by regular expressions.
Example: numerical constants can be described by:
 $\{+, -, \epsilon\}(DD^* \cup DD^*.D^* \cup D.DD^*)$ where
 $D = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$
- From the lexicon description by regular expressions one can generate automatically lexical analyzers, such as `lex` and `flex` on unix/linux machines.

Computation Theory – p.3/??

Theorem 1.28

A language can be recognized by a finite automaton iff it can be represented by a regular expression.

Proof Idea: This proof has two parts:

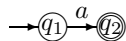
- **First part:** (Lemma 1.29) We show that if a language is represented by a regular expression, then there is a finite automaton that recognizes it.
- **Second part:** (Lemma 1.32) We show that if a language is recognized by a finite automaton, then there is a regular expression that represents it.

Equivalence with Finite Automata

- Regular expressions and finite automata are equivalent in their descriptive power.
- Any regular language L can be recognized by a finite automaton M_L .
- Any finite automaton recognizing a language A can be represented by a regular expression r_A specifying the language A .

Step 1:

If $r = a \in \Sigma$, then $L(r) = \{a\}$ and the NFA N recognizing $L(r)$ is:



Note: this is an NFA but not a DFA because it has states with no exiting arrow for each possible input symbol

Formal construction: $N = (\{q_1, q_2\}, \Sigma, \delta, q_1, \{q_2\})$ where $\delta(q_1, a) = \{q_2\}$, $\delta(r, b) = \emptyset$, for $r \neq q_1$ and $b \neq a$

Lemma 1.29

If a language is represented by a regular expression, then there is a finite automaton that recognizes it.

Proof idea: Assume that we have a regular expression r that represents the language A .

1. We will show how to construct from r an NFA that recognizes A , using a six-step procedure.
2. Then by Corollary 1.20, if an NFA recognizes A then A is recognizable by a DFA.

Step 3:

If $r = \emptyset$ then $L(r) = \emptyset$, and the NFA N that recognizes $L(r)$ is:



Formal construction: $N = (\{q\}, \Sigma, \delta, q, \emptyset)$ where $\delta(r, b) = \emptyset$ for any r and b

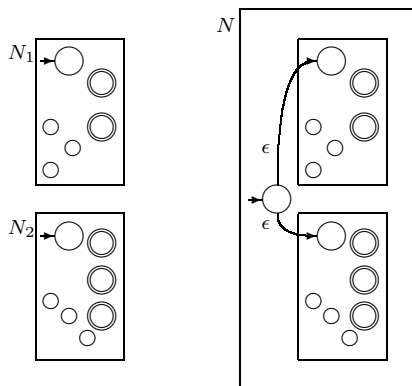
Step 2:

If $r = \epsilon$ then $L(r) = \{\epsilon\}$ and the NFA N that recognizes $L(r)$ is:



Formal construction: $N = (\{q_1\}, \Sigma, \delta, q_1, \{q_1\})$ where $\delta(r, b) = \emptyset$ for any r and $b \in \Sigma$

NFA N recognizing $L(r_1) \cup L(r_2)$



Step 4:

If $r = r_1 \cup r_2$ then $L(r) = L(r_1) \cup L(r_2)$.

Note: in view with the inductive nature of r we may assume that:

1. $N_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ is an NFA recognizing $L(r_1)$
2. $N_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ is an NFA recognizing $L(r_2)$

The NFA N recognizing $L(r_1 \cup r_2)$ is given in the next slide.

Step 5:

If $r = r_1 \circ r_2$ then $L(r) = L(r_1) \circ L(r_2)$.

Note: in view with the inductive nature of r we may assume that:

1. $N_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ is an NFA recognizing $L(r_1)$
2. $N_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ is an NFA recognizing $L(r_2)$.

The NFA N recognizing $L(r_1 \circ r_2)$ is given in the next slide.

Construction procedure

1. $Q = \{q_0\} \cup Q_1 \cup Q_2$: That is, the states of N are all states on N_1 and N_2 with the addition of a new state q_0
2. The start state of N is q_0
3. The accept states of N are $F = F_1 \cup F_2$: That is, the accept states of N are all the accept states of N_1 and N_2
4. Define δ so that for any $q \in Q$ and any $a \in \Sigma_\epsilon$:

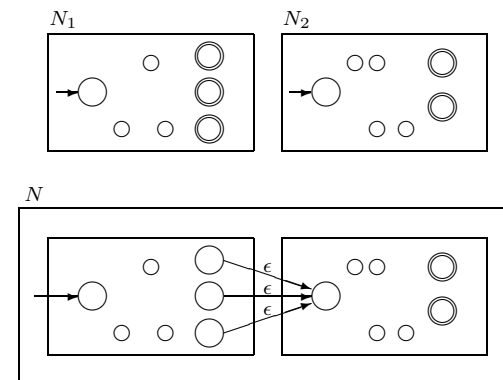
$$\delta(q, a) = \begin{cases} \delta_1(q, a), & \text{if } q \in Q_1 \\ \delta_2(q, a), & \text{if } q \in Q_2 \\ \{q_1, q_2\}, & \text{if } q = q_0 \text{ and } a = \epsilon \\ \emptyset, & \text{if } q = q_0 \text{ and } a \neq \epsilon. \end{cases}$$

Construction procedure

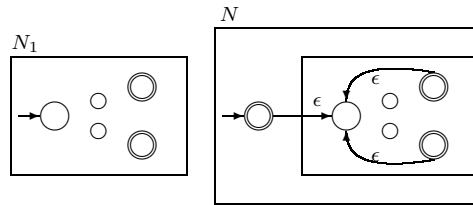
1. $Q = Q_1 \cup Q_2$. The states of N are all states of N_1 and N_2
2. The start state is the state q_1 of N_1
3. The accept states is the set F_2 of the accept states of N_2
4. Define δ so that for any $q \in Q$ and any $a \in \Sigma_\epsilon$:

$$\delta(q, a) = \begin{cases} \delta_1(q, a), & \text{if } q \in Q_1 \text{ and } q \notin F_1 \\ \delta_1(q, a), & \text{if } q \in F_1 \text{ and } a \neq \epsilon \\ \delta_1(q, a) \cup \{q_2\}, & \text{if } q \in F_1 \text{ and } a = \epsilon \\ \delta_2(q, a), & \text{if } q \in Q_2. \end{cases}$$

Construction of NFA N



Procedure for the construction of N



Step 6:

If $r = r_1^*$ then $L(r) = \cup_{i \geq 0} L(r_1)^i$.

Note: in view with the inductive nature of r we may assume that:

1. $N_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ is an NFA recognizing $L(r_1)$

The NFA N recognizing $L(r_1^*)$ is given in the next slide.

Examples conversion

Convert the following regular expressions into NFA following the procedure presented above

1. $(ab \cup a)^*$ to an NFA
2. $(a \cup b)^* aba$

Construction procedure

1. $Q = \{q_0\} \cup Q_1$; that is, states of N are the states of N_1 plus a new state q_0
2. Start state if N is q_0
3. $F = \{q_0\} \cup F_1$; that is, the accept states of N are the accept states of N_1 plus the new start state
4. Define δ so that for any $q \in Q$ and $a \in \Sigma_\epsilon$:

$$\delta(q, a) = \begin{cases} \delta_1(q, a), & \text{if } q \in Q_1 \text{ and } q \notin F_1 \\ \delta_1(q, a), & \text{if } q \in F_1 \text{ and } a \neq \epsilon \\ \delta_1(q, a) \cup \{q_1\}, & \text{if } q \in F_1 \text{ and } a = \epsilon \\ \{q_1\}, & \text{if } q = q_0 \text{ and } a = \epsilon \\ \emptyset, & \text{if } q = q_0 \text{ and } a \neq \epsilon. \end{cases}$$

Procedure

- Because A is regular, there is a DFA D_A that recognizes A
- D_A will be converted into a regular expression r_A that specifies A

Note: This procedure is broken in two parts:

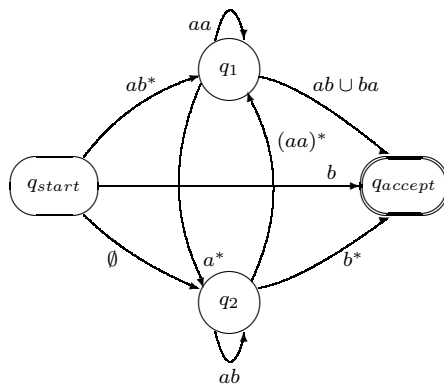
1. Convert a DFA into a *generalized nondeterministic finite automaton* GNFA
2. Convert GNFA into a regular expression

Lemma 1.32

If a language can be recognized by a finite automaton, then it is specified by a regular expression

Proof idea: For a given regular language A we will construct a regular expression that specifies A .

Example GNFA



What is an GNFA?

- A GNFA is an NFA wherein the transition arrows may have any regular expressions as labels, instead only members of the alphabet or ϵ
- Hence, GNFA reads strings specified by regular expressions (block of symbols) from the input (not necessarily just one symbol)
- GNFA moves along a transition arrow connecting two states representing regular expression.

GNFA of special form

- The start state has transition arrows to every other state but no arrow coming from any other state
- There is only one accept state and it has arrows coming in from every other state, but has no arrows going to any other state; in addition, the accept state is not the same with the start state
- Except for start and accept states, one arrow go from every state to every other state and from each state to itself

Note

- A GNFA is nondeterministic and so, it may have many different ways to process the same input string
- A GNFA accepts its input if its processing can cause the GNFA to be in an accept state at the end of the input

Note

Adding \emptyset transitions don't change the language recognized by DFA because a transition labeled by \emptyset can never be used

Assumption: from here on we assume that all GNFA's are in the special form

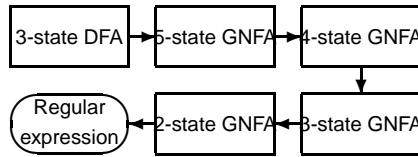
Converting DFA to GNFA

A DFA is converted to a GNFA of special form as follows:

- Add a new start state with an ϵ arrow to the old start state and a new accept state with an ϵ arrow from all old accept states
- If any arrows have multiple labels or if there are multiple arrows going between the same two states in the same direction replace each with a single arrow whose label is the union of the previous labels
- Add arrows labeled \emptyset between states that had now arrows

Example DFA conversion

Assuming that the original DFA has 3 states the process of its conversion is:



Computation Theory – p.30/??

Converting GNFA to Regular Expressions

Assume that GNFA has k states

- Because start and accept states are different from each other, it results that $k \geq 2$
- If $k > 2$ we construct an equivalent GNFA with $k - 1$ states. This can be repeated for each new GNFA until we obtain a GNFA with $k = 2$ states.
- If $k = 2$, GNFA has a single arrow that goes from start to accept and is labeled by a regular expression that specifies the language recognized by the original DFA

Computation Theory – p.29/??

Repairing after ripping a state

Assume that state of GNFA selected for ripping is q_{rip}

- After removing q_{rip} we repair the machine by altering the regular expressions that label each of the remaining transitions
- The new labels compensate for the absence of q_{rip} by adding back the lost computation
- The new label of the arrow going from state q_i to q_j is a regular expression that specifies all strings that would take the machine from q_i to q_j either directly or via q_{rip}

Computation Theory – p.32/??

Note

- The crucial step is in constructing an equivalent GNFA with one fewer states when $k > 2$
- This is done by selecting a state, ripping it out of the machine, and repairing the remainder so that the same language is still recognized
- Any state can be selected for ripping, providing that it is not start or accept state. Such a state exist because $k > 2$

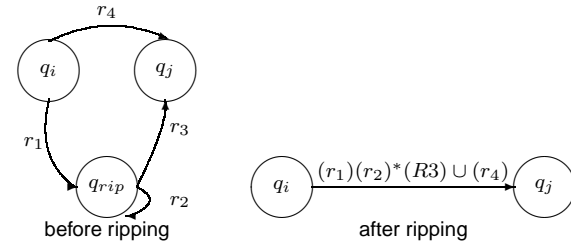
Computation Theory – p.31/??

Note

- New labels are obtained by concatenating regular expressions of the arrows that go through q_{rip} and union them with the labels of the arrows that travel directly between q_i and q_j
- This construct is carried out for each arrow that goes from state q_i to any state q_j including $q_i = q_j$

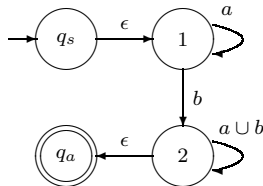
Illustration

We illustrate the approach of ripping and repairing below.



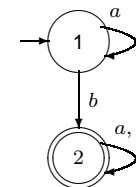
GNFA G_1 obtained from D

The following figure shows the four-state GNFA obtained from D by adding new start state and accept state and replacing a, b by $a \cup b$



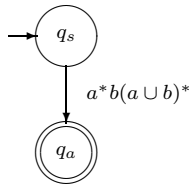
Example 1.35

Convert the DFA D into the regular expression that specifies the language accepted by D



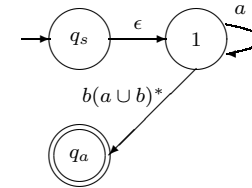
Eliminating node 1 from G_2

The following figure shows the GNFA G_3 obtained from G_2 by ripping off node 1



Eliminating node 2 from G_1

The following figure shows the GNFA G_2 obtained from G_1 by ripping off node 2



Transition function of a GNFA

- Because an arrow connects every state to every other state, except that no arrows are coming from q_a or going to q_s , the domain of the transition function of a GNFA is $\delta : (Q - \{q_a\}) \times (Q - \{q_s\}) \rightarrow \mathcal{R}$
- If $\delta(q_i, q_j) = R$ the arrow from q_i to q_j has the label R

Formal Proof

- First we need to define formally the GNFA
- Since new labels are regular expressions we use the symbol \mathcal{R} to denote the collection of regular expressions over an alphabet Σ
- To simplify, denote by q_s and q_a the start and accept states of the GNFA

Computation performed by a GNFA

A GNFA accepts a string $w \in \Sigma^*$ if $w = w_1 w_2 \dots w_k$ where $w_i \in \Sigma^*$, $1 \leq i \leq k$, and a sequence of states q_0, q_1, \dots, q_k exists such that:

1. $q_0 = q_s$ is the start state
2. $q_k = q_a$ is the accept state
3. For each i , $\delta(q_{i-1}, q_i) = r_i$ and $w_i \in L(r_i)$, i.e., r_i is the regular expression labeling the arrow from q_{i-1} to q_i and w_i is an element of the language specified by this expression

Definition 1.33

A generalized nondeterministic finite automaton is a 5-tuple $(Q, \Sigma, \delta, q_s, q_a)$ where:

1. Q is the finite set of state
2. Σ is the input alphabet
3. $\delta : (Q - \{q_a\}) \times (Q - \{q_s\}) \rightarrow \mathcal{R}$ is the transition function
4. q_s is the start state
5. q_a is the unique accept state

Convert(G)

1. Let k be the number of state of G
2. If $k = 2$ then G must consists of a start state and an accept state and a single arrow connecting them, labeled by a regular expression r .
Return r
3. While $k > 2$, select any state $q_{rip} \in Q$, different from q_s and q_a and let G' be the GNFA $(Q', \Sigma, \delta', q_s, q_a)$ where:
 - $Q' = Q - \{q_{rip}\}$
 - for any $q_i \in Q' - \{q_a\}$ and any $q_j \in Q' - \{q_s\}$ let $\delta'(q_i, q_j) = (r_1)(r_2)^*(r_3) \cup (r_4)$ where:
 $r_1 = \delta(q_i, q_{rip})$, $r_2 = \delta(q_{rip}, q_{rip})$, $r_3 = \delta(q_{rip}, q_j)$, $r_4 = \delta(q_i, q_j)$
 - $Convert(G')$;

More proof ideas

Returning to the proof of Lemma 1.32, we assume that M is a DFA recognizing the language A and proceed as follows:

- Convert M into a GNFA G by adding a new start state and a new accept state and the additional arrows
- Use the procedure $Convert(G)$ that maps G into a regular expression, as explained before, while preserving the language A

Note: $Convert()$ is recursive; however that case when GNFA has only two states is handled without recursion

Induction Basis:

$k = 2$

- If G has only two states, by definition, it can have only a single arrow which goes from q_s to q_a
- The regular expression labeling this arrow specifies the language accepted by G
- Since this expression is returned by $Convert(G)$, it means that G and $Convert(G)$ are equivalent

Claim 1.34

For any GNFA G , $Convert(G)$ is equivalent to G

Proof: by induction on k , the number of states of G

Induction Step (2)

1. If none of the states $q_s, q_1, q_2, \dots, q_a$ is q_{rip} , clearly G' also accepts w because each of the new regular expressions labeling arrows of G' contain the old regular expressions as part of a union
2. If q_{rip} does appear in the computation $q_s, q_1, q_2, \dots, q_a$ by removing each run of consecutive q_{rip} states we obtain an accepting computation for G' . This is because states q_i and q_j bracketing a run of consecutive q_{rip} states have a new regular expression on the arrow between them that specifies all strings taking q_i to q_j via q_{rip} on G . So, G' accepts w in this case too.

Induction Step

Assume that the claim is true for G having $k - 1$ states and use this assumption to show that the claim is true for an GNFA with k states

- Observe from construction that G and G' recognize the same language
- Suppose G accepts the input w . Then in an accepting branch of computation, G enters the sequence of states $q_s, q_1, q_2, q_3, \dots, q_a$
- Show that G' has an accepting computation for w , too.

Conclusion

- The induction hypothesis states that when the algorithm calls itself recursively on input G' , the result is a regular expression that is equivalent to G' because G' has $k - 1$ states
- Hence, that regular expression is also equivalent to G because G' is equivalent to G
- Consequently $Convert(G)$ and G are equivalent

Induction Step (3)

For the other direction, suppose that G' accepts w .

1. Each arrow between any two states q_i and q_j in G' is labeled by a regular expression that specifies strings specified by arrows in G from q_i directly to q_j or via q_{rip}
2. Hence, by the definition of GNFA it follows that G must also accept w .

That is, G and G' accept the same language