CPS 220 – Theory of Computation

REGULAR LANGUAGES

Regular expressions

Like mathematical expression (5+3) * 4. Regular expression are built using regular operations.
(By the way, regular expressions show up in various languages: Perl, Java, Python, etc - great for pattern matching operations).

Examples:

a^*b
$(0 \cup 1)^* = (0+1)^*$

Definition. R is a regular expression, if R is one of the following:

1. ε
2. a, for some $a \in \Sigma$
3. $\varnothing$
4. $(R_1 \cup R_2)$, where R_1 and R_2 are regular expressions
5. R_1R_2, where R_1 and R_2 are regular expressions
6. $(R_1)^*$, where R_1 is a regular expression

Note: it is a inductive definition - it is defined based on itself.
Note: the + symbol will at times be used for union (0+1)*

$R^+ = RR$

$R^+ \cup \varepsilon = R^*$

L((0+1)*1(0+1)*) = {w|w contains a 1 in the middle}
(Note: other regular expression examples on p.65)

Equivalence of Regular Expressions and Finite Automata

Theorem. A language is regular if and only if some regular expression describes it.

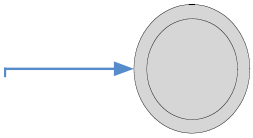
We need to prove two directions:
1. If a language is described by a regular expression, then it is regular.
2. If a language is regular, then there is a regular expression that describes it.

Part 1. This is the easy part - "If a language is described by a regular expression, then it is regular."

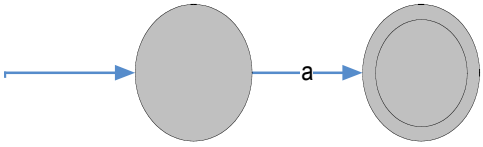
Say that a regular expression R describes some language A.
We will convert R to an NFA N that recognizes A. Then, A has to be regular.

R has one of six possible forms:

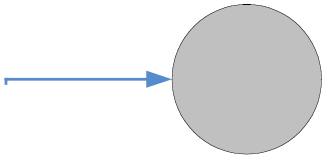
1. $R = \varepsilon$. Then,



2. $R = a$, for some $a \in \Sigma$. Then,



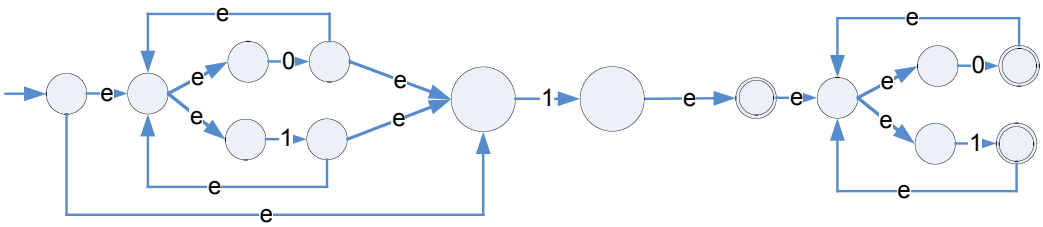
3. $R = \varnothing$. Then,



4. $R = R_1 \cup R_2$.
5. $R = R_1R_2$.
6. $R = (R_1)^*$

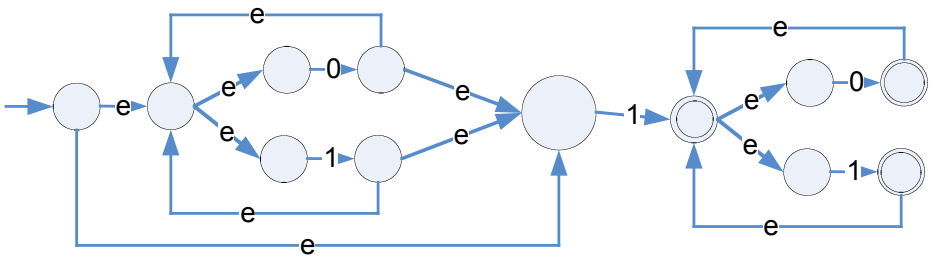
In the last three cases, the constructions given in the proofs that the class of regular languages are closed under the regular operations can be used here as well.
That is, we assume R_1 and R_2 are recognized by NFAs N_1 and N_2, and use the same constructions to create N from N_1 and N_2.

Convert the following regular expression to a NFA:
(0+1)*1(0+1)*



Note: using the letter "e" to represent ε

Note: For the above resulting NFA - major elimination of states and transistion can take place.
Double e's can be ripped out.



What else can be eliminated?

Part 2: We need to show that if a language is regular, then it is described by a regular expression.
"If a language is regular, then there is a regular expression that describes it."

If a language A is regular, then it is recognized by a DFA M.
We will show how to convert an arbitrary DFA M into an equivalent regular expression.

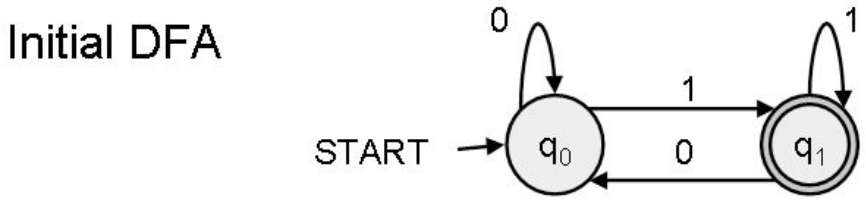
The main idea is that we will gradually eliminate the states of M.
Strategy: DFA --> GNFA --> regular expressions

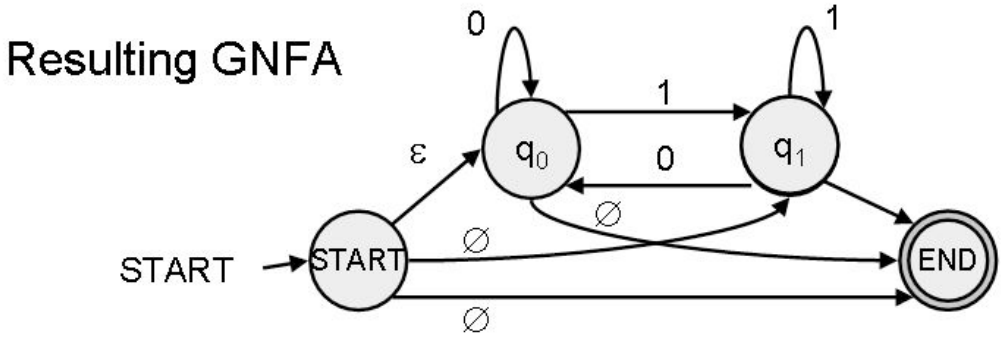
GNFA - generalized nondeterministic finite automaton

Definition (informal—details in the book). A GNFA is a Finite Automaton, except:
1. There is only one start state, one final state, and the two are distinct.
2. The start state has arrows going to every other state.
3. There are no arrows going to the start state.
4. The final state has arrows coming from every other state.
5. There are no arrows leaving the final state.
6. Every state (except start & final) has arrows going to every other state.
7. The labels of the arrows are regular expressions.

To convert a DFA into a GNFA:
1. Create a new start state, with ε-transitions to the previous start state and ∅-transitions to all other states.
2. Create a new final state, with ε-transitions from the previous final state and ∅-transitions from all other states.
3. Add ∅-transitions from every state to every state (other than start & final).

Example: (Note: a even more profound example would be a DFA with multiple accept states)



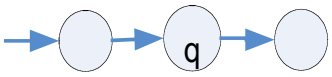


The resulting GNFA always has ≥ 2 states. (It actually always has ≥ 3 states, but now we will start removing states until it has exactly 2 states.)

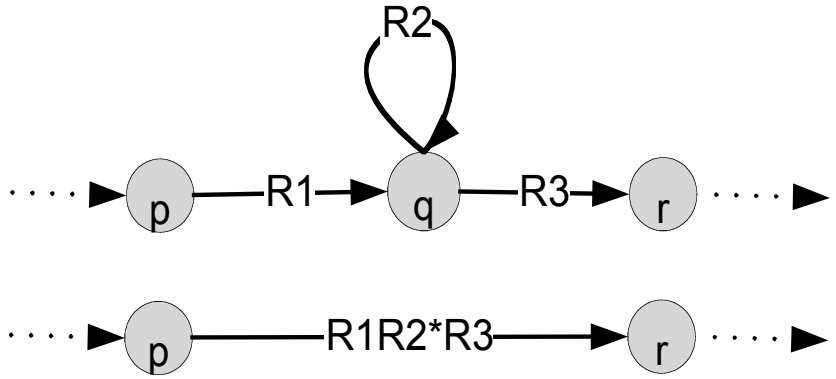
To convert from GNFA to RE:
We will remove states incrementally, and at the same time build up a regular expression.

Whenever the GNFA has ≥ 2 states, we eliminate a state as follows:

The state q below is any state other than the (unique) start and (unique) final states. That state q, has possibly a number of transitions to it, including a self-transition, and a number of transitions from it.



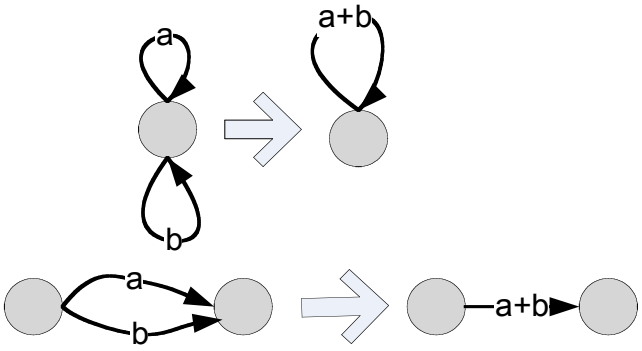
For every pair of states p and r (including start & final), q is "between" p and r: p always has a transition to q, and r always has a transition from q.
We remove q, and in its place put a regular expression describing how to get from p to r through q:



The above 2 GNFA segments are equivalent.
This idea of equivalency means: "to get from p to r, either get through the arrow from p to r, or
get through the old q-path, in which case you need to go through the old p-to-q transition, then possibly loop in q an arbitrary number of times, and then go through the old q-to-r path".

Continue this process of eliminating states, until we are left with just the start and final states.
Then, E is the regular expression we need! E will accept exactly the strings that the old DFA was accepting.

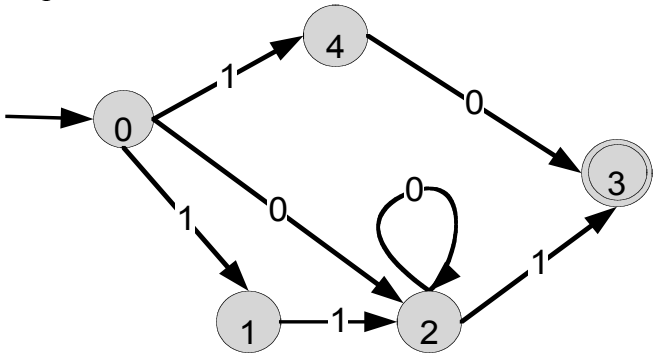
Multiple edges can be removed:



Practice.

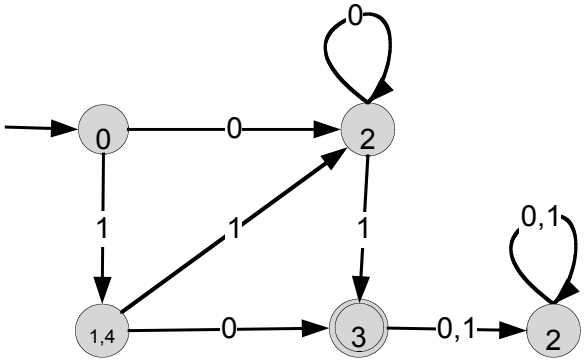
1. Find a DFA accepting the language 10 + (0 + 11)0*1

Step1. Find NFA

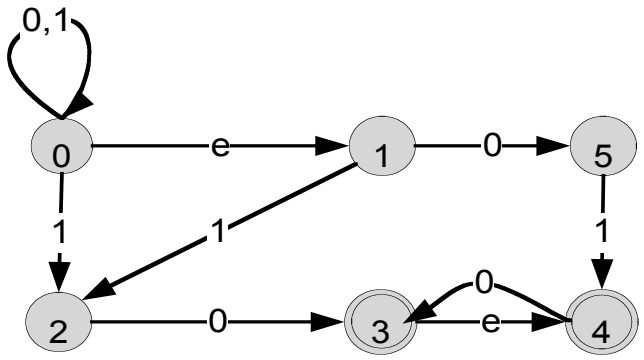


Step2. Transform NFA to DFA

δ'	0	1
$Q_\varepsilon = \{q0\}$	$\{q2\}$	$\{q1, q4\}$
$Q_0 = \{q2\}$	$\{q2\}$	$\{q3\}$
$Q_1 = \{q1, q4\}$	$\{q3\}$	$\{q2\}$
$Q_{01} = \{q3\}$	$\varnothing$	$\varnothing$



2. Find the regular expression accepted by this NFA.



Answer:
(01+10)0*(e+0)+10 = (10+10)0*

Simplifying formulas for REs (prove them as an exercise!):

Note: E represents the alphabet.

$\varnothing + E = E$

$\varepsilon E = E\varepsilon = E$

$\varnothing E = E\varnothing = \varnothing$

$(E^*)^* = E^*$

$\varnothing^* = \varepsilon$

$\varepsilon^* = \varepsilon$

$E + E = E$

$E(F + G) = EF + EG$

$(F + G)E = FE + GE$

The UNIX-style $^+$-operator, matches a string once or more times (not zero times). We could define it here as $E^+ = EE^* = E^*E$. Then, check as an exercise that $E^* = E^+ \cup \varepsilon$.

References:

Introduction to the Theory of Computation (2nd ed.) Michael Sipser
Problem Solving in Automata, Languages, and Complexity Ding-Zhu Du and Ker-I Ko