## cse 311: foundations of computing

Fall 2015
Lecture 24: DFAs, NFAs, and regular expressions


- FSMs with output at states
- State minimization



## highlights

Lemma: The language recognized by a DFA is the set of strings $x$ that label some path from its start state to one of its final states


## nondeterministic finite automaton (NFA)

- Graph with start state, final states, edges labeled by symbols (like DFA) but
- Not required to have exactly 1 edge out of each state labeled by each symbol--- can have 0 or >1
- Also can have edges labeled by empty string $\varepsilon$
- Definition: $x$ is in the language recognized by an NFA if and only if $x$ labels a path from the start state to some final state

binary strings that have
- an even \# of 1's
- or contain the substring 111 or 1000


## NFAs and regular expressions

Theorem: For any set of strings (language) $A$ described by a regular expression, there is an NFA that recognizes $A$.

Proof idea: Structural induction based on the recursive definition of regular expressions...


## (01 $\cup 1$ )*0



Every DFA is an NFA

- DFAs have requirements that NFAs don't have

Can NFAs recognize more languages?

Every DFA is an NFA

- DFAs have requirements that NFAs don't have

Can NFAs recognize more languages? No!

Theorem: For every NFA there is a DFA that recognizes exactly the same language.

## conversion of NFAs to DFAs

## Proof Idea:

- The DFA keeps track of ALL the states that the part of the input string read so far can reach in the NFA
- There will be one state in the DFA for each subset of states of the NFA that can be reached by some string


## conversion of NFAs to a DFAs

## New start state for DFA

- The set of all states reachable from the start state of the NFA using only edges labeled $\varepsilon$



NFA
DFA

## conversion of NFAs to a DFAs

## For each state of the DFA corresponding to a set S of states of

 the NFA and each letter a- Add an edge labeled a to state corresponding to $T$, the set of states of the NFA reached by
starting from some state in S , then
following one edge labeled by a, and
then following some number of edges labeled by $\varepsilon$
- T will be $\varnothing$ if no edges from $S$ labeled a exist



## conversion of NFAs to a DFAs

## Final states for the DFA

- All states whose set contain some final state of the NFA


NFA
DFA

## example: NFA to DFA



NFA
DFA


## example: NFA to DFA



NFA
DFA

## example: NFA to DFA



NFA
DFA

## example: NFA to DFA



NFA
DFA

## example: NFA to DFA



NFA
DFA

## example: NFA to DFA



NFA
DFA

## example: NFA to DFA



NFA
DFA
exponential blow-up in simulating mondeterminism

- In general the DFA might need a state for every subset of states of the NFA
- Power set of the set of states of the NFA
- n-state NFA yields DFA with at most $2^{n}$ states
- We saw an example where roughly $2^{n}$ is necessary Is the $\mathrm{n}^{\text {th }}$ char from the end a 1 ?
- The famous " $\mathrm{P}=\mathrm{NP}$ ?" question asks whether a similar blowup is always necessary to get rid of nondeterminism for polynomial-time algorithms


## 1 in third position from end



## 1 in third position from end



## 1 in third position from end



## DFAs $\equiv$ regular expressions

We have shown how to build an optimal DFA for every regular expression

- Build NFA
- Convert NFA to DFA using subset construction
- Minimize resulting DFA

Theorem: A language is recognized by a DFA if and only if it has a regular expression.

We show the other direction of the proof at the end of these lecture slides.

## languages and machines!



## languages and machines!



## DFAs recognize any finite language

Exercise: Hard code it into the NFA.

## languages and machines!



## languages and machines!



## DFAs $\equiv$ regular expressions

Theorem: A language is recognized by a DFA if and only if it has a regular expression

Proof: We already saw: RegExp $\rightarrow$ NFA $\rightarrow$ DFA
Now: NFA $\rightarrow$ RegExp
(Enough to show this since every DFA is also an NFA.)

## generalized NFAs

- Like NFAs but allow
- Parallel edges
- Regular Expressions as edge labels

NFAs already have edges labeled $\varepsilon$ or a

- An edge labeled by A can be followed by reading a string of input chars that is in the language represented by A
- A string x is accepted iff there is a path from start to final state labeled by a regular expression whose language contains $x$

Add new start state and final state


Then eliminate original states one by one, keeping the same language, until it looks like:


Final regular expression will be A

## only two simplification rules

- Rule 1: For any two states $\mathrm{q}_{1}$ and $\mathrm{q}_{2}$ with parallel edges (possibly $\mathrm{q}_{1}=\mathrm{q}_{2}$ ), replace

- Rule 2: Eliminate non-start/final state $\mathrm{q}_{3}$ by replacing all

for every pair of states $q_{1}, q_{2}$ (even if $q_{1}=q_{2}$ )


## converting an NFA to a regular expression

## Consider the DFA for the mod 3 sum

- Accept strings from $\{0,1,2\}^{*}$ where the digits mod 3 sum of the digits is 0



## splicing out a node

## Label edges with regular expressions

$$
\begin{array}{ll}
\mathrm{t}_{0} \rightarrow \mathrm{t}_{1} \rightarrow \mathrm{t}_{0}: & 10 \star 2 \\
\mathrm{t}_{0} \rightarrow \mathrm{t}_{1} \rightarrow \mathrm{t}_{2}: & 10^{\star} 1 \\
\mathrm{t}_{2} \rightarrow \mathrm{t}_{1} \rightarrow \mathrm{t}_{0}: & 20^{\star 2} \\
\mathrm{t}_{2} \rightarrow \mathrm{t}_{1} \rightarrow \mathrm{t}_{2}: & 20^{\star} 1
\end{array}
$$



## finite automaton without $\mathrm{t}_{1}$

$\mathrm{R}_{1}: 0 \cup 10 \star 2$
$\mathrm{R}_{2}: 2 \cup 10 * 1$
$\mathrm{R}_{3}: 1 \cup 20 \star 2$
$\mathrm{R}_{4}: 0 \cup 20^{*} 1$
$\mathrm{R}_{5}: \mathrm{R}_{1} \cup \mathrm{R}_{2} \mathrm{R}_{4}{ }^{*} \mathrm{R}_{3}$


Final regular expression:
$(0 \cup 10 \star 2 \cup(2 \cup 10 * 1)(0 \cup 20 * 1) *(1 \cup 20 * 2))^{*}$

