CSE 311 Foundations of Computing I

Lecture 26
NFAs, Regular Expressions, and Equivalence with DFAs
Autumn 2012

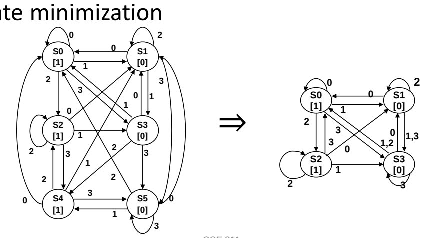
Announcements

- Reading assignments
  - 7th Edition, Sections 13.3 and 13.4
  - 6th Edition, Section 12.3 and 12.4
  - 5th Edition, Section 11.3 and 11.4
- Problem 6 dropped from Homework 9
- Topic list and sample final exam problems have been posted
- Comprehensive final, roughly 67% of material post midterm
- Review session TBA (Saturday, December 8)
- Final exam, Monday, December 10
  - 2:30-4:20 pm or 4:30-6:20 pm, Kane 220.
  - If you have a conflict, contact instructors ASAP

Last lecture highlights

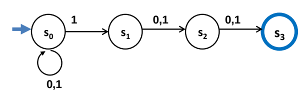
Finite State Machines with output at states

State minimization



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 λ
- Definition: The language recognized by an NFA is the set of strings x that label some path from its start state to one of its final states



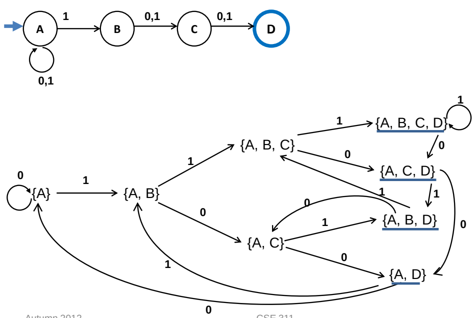
Three ways of thinking about NFAs

- Outside observer: Is there a path labeled by x from the start state to some final state?
- Perfect guesser: The NFA has input x and whenever there is a choice of what to do it magically guesses a good one (if one exists)
- Parallel exploration: The NFA computation runs all possible computations on x step-by-step at the same time in parallel

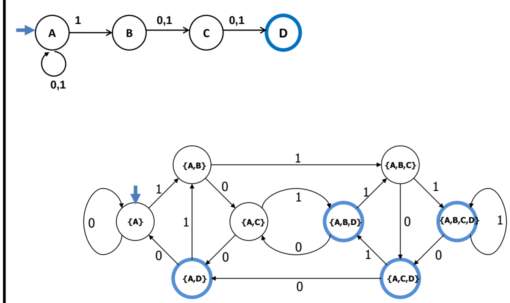
Conversion of NFAs to a DFAs

- 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

1 in third position from end

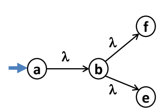


Redrawing

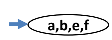


Conversion of NFAs to a DFAs

- New start state for DFA
  - The set of all states reachable from the start state of the NFA using only edges labeled λ



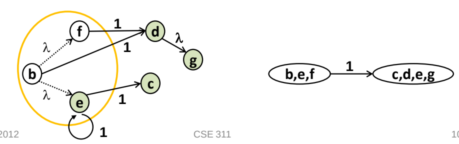
NFA



DFA

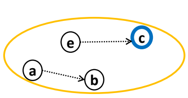
Conversion of NFAs to a DFAs

- For each state of the DFA corresponding to a set S of states of the NFA and each symbol **s**
  - Add an edge labeled **s** 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 **s**, and
    - then following some number of edges labeled by λ
  - T will be $\varnothing$ if no edges from S labeled **s** exist



Conversion of NFAs to a DFAs

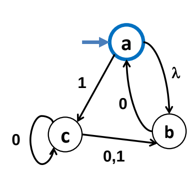
- Final states for the DFA
  - All states whose set contain some final state of the NFA



NFA

DFA

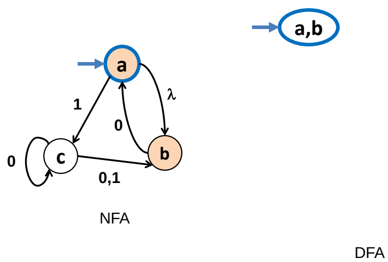
Example: NFA to DFA



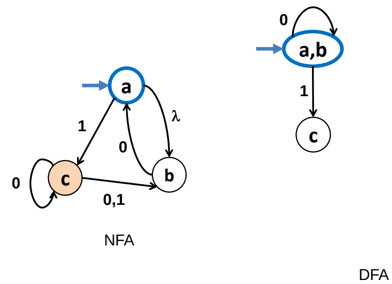
NFA

DFA

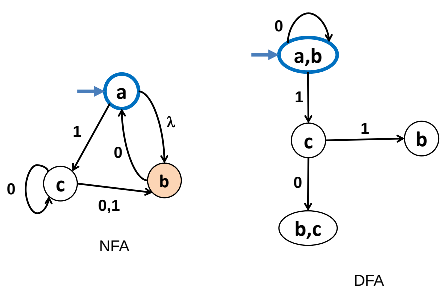
Example: NFA to DFA



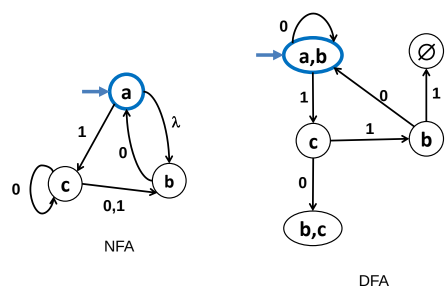
Example: NFA to DFA



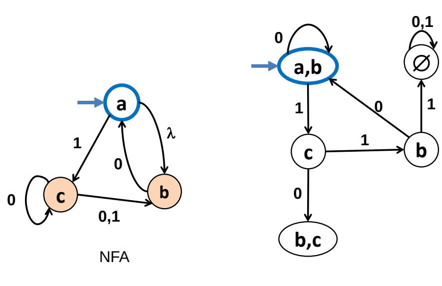
Example: NFA to DFA



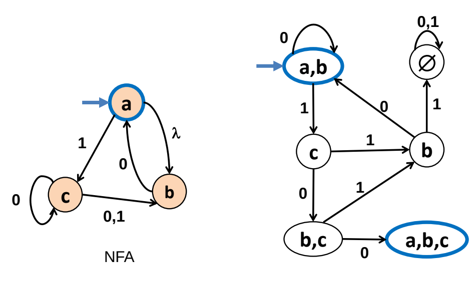
Example: NFA to DFA



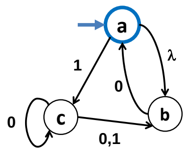
Example: NFA to DFA



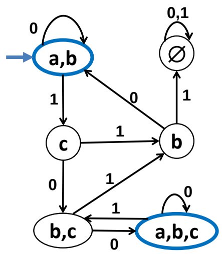
Example: NFA to DFA



Example: NFA to DFA



NFA



DFA

Exponential blow-up in simulating nondeterminism

- 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 10th char from the end a 1?
- The famous "P=NP?" question asks whether a similar blow-up is always necessary to get rid of nondeterminism for polynomial-time algorithms

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...

Note: One can also find a regular expression to describe the language recognized by any NFA but we won't prove that fact

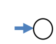
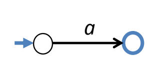
Regular expressions over Σ

- Basis:
  - $\varnothing$, λ are regular expressions
  - ***a*** is a regular expression for any $a \in \Sigma$
- Recursive step:
  - If **A** and **B** are regular expressions then so are:
    - (**A** $\cup$ **B**)
    - (**AB**)
    - **A***

Basis

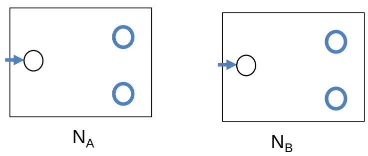
- Case $\varnothing$:
- Case λ:
- Case ***a***:

Basis

- Case $\varnothing$: 
- Case λ: 
- Case ***a***: 

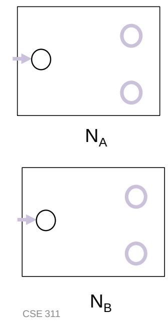
Inductive Hypothesis

- Suppose that for some regular expressions **A** and **B** there exist NFAs N_A and N_B such that N_A recognizes the language given by **A** and N_B recognizes the language given by **B**



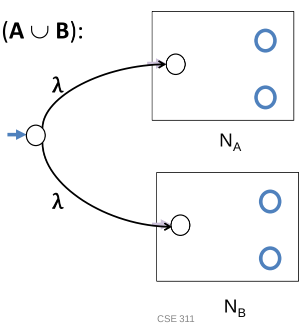
Inductive Step

- Case (**A** ∪ **B**):



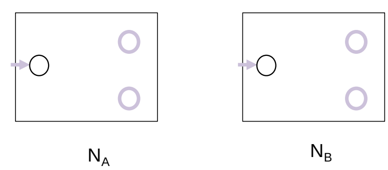
Inductive Step

- Case (**A** ∪ **B**):



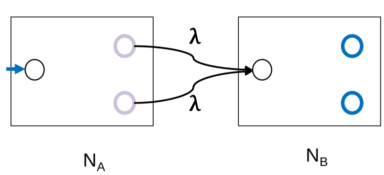
Inductive Step

- Case (**AB**):



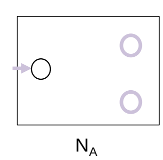
Inductive Step

- Case (**AB**):



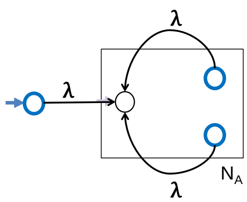
Inductive Step

- Case **A***



Inductive Step

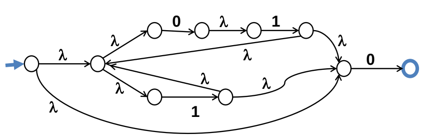
- Case **A***



Build a NFA for (01 ∪1)*0

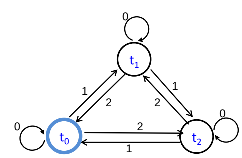
Solution

(01 ∪1)*0



Converting an NFA to a regular expression

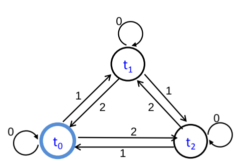
- Consider the DFA for the mod 3 sum
  - Accept strings from {0,1,2}* where the digits mod 3 sum of the digits is 0



Splicing out a node

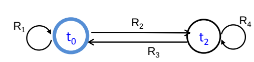
- Label edges with regular expressions

$t_0 \to t_1 \to t_0$: 10*2
$t_0 \to t_1 \to t_2$: 10*1
$t_2 \to t_1 \to t_0$: 20*2
$t_2 \to t_1 \to t_2$: 20*1

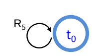


Finite Automaton without t_1

R_1: 0 | 10*2
R_2: 2 | 10*1
R_3: 1 | 20*2
R_4: 0 | 20*1



R_5: R_1 | R_2R_4*R_3



Final regular expression:
(0 | 10*2 | (2 | 10*1)(0 | 20*1)*(1 |20*2))*