CMSC 330: Organization of Programming Languages

DFAs, and NFAs, and Regexps

CMSC 330 Spring 2024

1

The story so far, and what's next

- Goal: Develop an algorithm that determines whether a string *s* is matched by regex *R*
  - I.e., whether *s* is a member of *R*'s *language*
- Approach to come: Convert *R* to a finite automaton *FA* and see whether *s* is accepted by *FA*
  - Details: Convert *R* to a *nondeterministic FA* (NFA), which we then convert to a *deterministic FA* (DFA),
    - which enjoys a fast acceptance algorithm

CMSC 330 Spring 2024

2

Two Types of Finite Automata

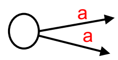
- Deterministic Finite Automata (DFA)
  - Exactly one sequence of steps for each string
    - Easy to implement acceptance check
  - (Almost) all examples so far
- Nondeterministic Finite Automata (NFA)
  - May have many sequences of steps for each string
  - Accepts if any path ends in final state at end of string
  - More compact than DFA
    - But more expensive to test whether a string matches

CMSC 330 Spring 2024

3

Comparing DFAs and NFAs

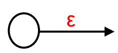
- NFAs can have more than one transition leaving a state on the same symbol



- DFAs allow only one transition per symbol
  - I.e., transition function must be a valid function
  - DFA is a special case of NFA

Comparing DFAs and NFAs (cont.)

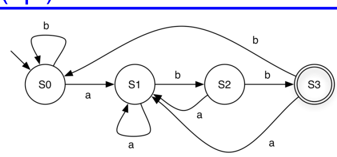
- NFAs may have transitions with empty string label
  - May move to new state without consuming character



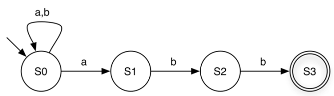
ε-transition

- DFA transition must be labeled with symbol
  - A DFA is a specific kind of NFA

DFA for (a|b)*abb

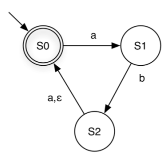


NFA for (a|b)*abb



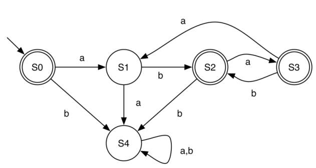
- ba
  - Has paths to either S0 or S1
  - Neither is final, so rejected
- babaabb
  - Has paths to different states
  - One path leads to S3, so accepts string

NFA for (ab|aba)*

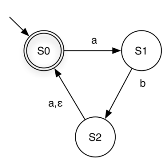


- aba
- ababa
  - Has paths to states S0, S1
  - Need to use ε-transition

NFA and DFA for (ab|aba)*



DFA



NFA

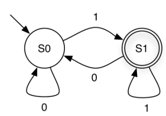
Formal Definition

- A deterministic finite automaton *(DFA)* is a 5-tuple (Σ, Q, q_0, F, δ) where
  - Σ is an alphabet
  - Q is a nonempty set of states
  - $q_0 \in Q$ is the start state
  - $F \subseteq Q$ is the set of final states
  - $\delta : Q \times \Sigma \rightarrow Q$ specifies the DFA's transitions
    - What's this definition saying that δ is?
- A DFA accepts s if it stops at a final state on s

10

Formal Definition: Example

- $\Sigma = \{0, 1\}$
- Q = {S0, S1}
- q_0 = S0
- F = {S1}
- δ =



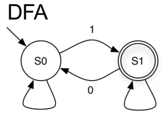
input state \ symbol	0	1
S0	S0	S1
S1	S0	S1

or as { (S0,0,S0),
(S0,1,S1),
(S1,0,S0),
(S1,1,S1) }

11

Implementing DFAs (one-off)

It's easy to build a program which mimics a DFA




```
cur_state = 0;
while (1) {

  symbol = getchar();

  switch (cur_state) {

    case 0: switch (symbol) {
              case '0':  cur_state = 0; break;
              case '1':  cur_state = 1; break;
              case '\n': printf("rejected\n"); return 0;
              default:   printf("rejected\n"); return 0;
            }
            break;

    case 1: switch (symbol) {
              case '0':  cur_state = 0; break;
              case '1':  cur_state = 1; break;
              case '\n': printf("accepted\n"); return 1;
              default:   printf("rejected\n"); return 0;
            }
            break;

    default: printf("unknown state; I'm confused\n");
             break;
  }
}
```


12

Implementing DFAs (generic)

More generally, use generic table-driven DFA

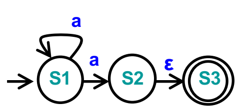
```
given components (Σ, Q, q0, F, δ) of a DFA:
let q = q0
while (there exists another symbol σ of the input string)
   q := δ(q, σ);
if q ∈ F then
  accept
else reject
```

- q is just an integer
- Represent δ using arrays or hash tables
- Represent F as a set

13

Nondeterministic Finite Automata (NFA)

- An *NFA* is a 5-tuple $(\Sigma, Q, q_0, F, \delta)$ where
  - Σ, Q, q0, F as with DFAs
  - $\delta \subseteq Q \times (\Sigma \cup \{\varepsilon\}) \times Q$ specifies the NFA's transitions



Example

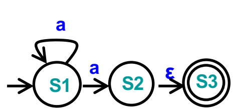
- $\Sigma = \{a\}$
- $Q = \{S1, S2, S3\}$
- $q_0 = S1$
- $F = \{S3\}$
- $\delta = \{ (S1,a,S1), (S1,a,S2), (S2,\varepsilon,S3) \}$

- An NFA accepts s if there is at least one path via s from the NFA's start state to a final state

14

NFA Acceptance Algorithm (Sketch)

- When NFA processes a string s
  - NFA must keep track of several "current states"
    - Due to multiple transitions with same label, and ε-transitions
  - If any current state is final when done then accept s
- Example
  - After processing "a"
    - NFA may be in states
      S1
      S2
      S3
    - Since S3 is final, s is accepted

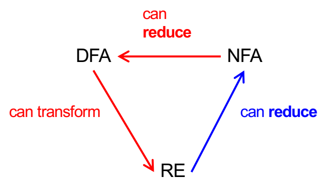


- Algorithm is slow, space-inefficient; prefer DFAs!

15

Relating REs to DFAs and NFAs

- Regular expressions, NFAs, and DFAs accept the same languages! *Can convert between them*



NB. Both *transform* and *reduce* are historical terms; they mean "convert"

CMSC 330 Spring 2024

16

Reducing Regular Expressions to NFAs

- Goal: Given regular expression A, construct NFA: $\langle A \rangle = (\Sigma, Q, q_0, F, \delta)$
  - Remember regular expressions are defined recursively from primitive RE languages
  - Invariant: $|F| = 1$ in our NFAs
    - Recall F = set of final states
- Will define $\langle A \rangle$ for base cases: σ , ε , $\emptyset$
  - Where σ is a symbol in Σ
- And for inductive cases: AB, $A|B$, A^*

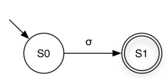
CMSC 330 Spring 2024

17

Reducing Regular Expressions to NFAs

- Base case: σ

Recall: NFA is $(\Sigma, Q, q_0, F, \delta)$
where
Σ is the alphabet
Q is set of states
q_0 is starting state
F is set of final states
δ is transition relation



$\langle\sigma\rangle = (\{\sigma\}, \{S0, S1\}, S0, \{S1\}, \{(S0, \sigma, S1)\})$

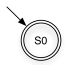
$(\Sigma, \quad Q, \quad q_0, \quad F, \quad \delta \quad)$

CMSC 330 Spring 2024

18

Reduction

- Base case: ε



$\langle\varepsilon\rangle = (\emptyset, \{S0\}, S0, \{S0\}, \emptyset)$

Recall: NFA is $(\Sigma, Q, q_0, F, \delta)$
where
- Σ is the alphabet
- Q is set of states
- q_0 is starting state
- F is set of final states
- δ is transition relation

- Base case: ∅

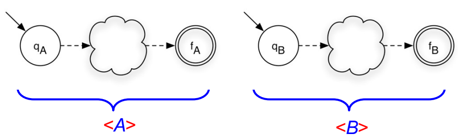
$\langle\emptyset\rangle = (\emptyset, \{S0, S1\}, S0, \{S1\}, \emptyset)$

CMSC 330 Spring 2024

19

Reduction: Concatenation

- Induction: *AB*



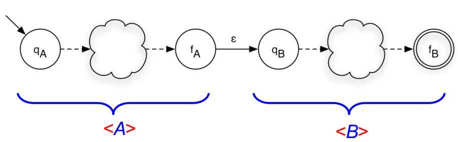
- $\langle A\rangle = (\Sigma_A, Q_A, q_A, \{f_A\}, \delta_A)$
- $\langle B\rangle = (\Sigma_B, Q_B, q_B, \{f_B\}, \delta_B)$

CMSC 330 Spring 2024

20

Reduction: Concatenation

- Induction: *AB*



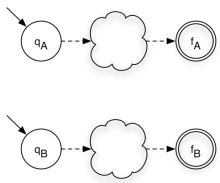
- $\langle A\rangle = (\Sigma_A, Q_A, q_A, \{f_A\}, \delta_A)$
- $\langle B\rangle = (\Sigma_B, Q_B, q_B, \{f_B\}, \delta_B)$
- $\langle AB\rangle = (\Sigma_A \cup \Sigma_B, Q_A \cup Q_B, q_A, \{f_B\}, \delta_A \cup \delta_B \cup \{(f_A,\varepsilon,q_B)\})$

CMSC 330 Spring 2024

21

Reduction: Union

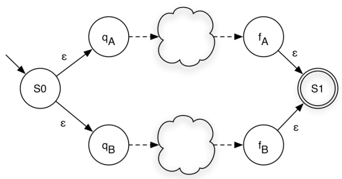
- Induction: *A*|*B*



  - <A> = (Σ_A, Q_A, q_A, {f_A}, δ_A)
  - <B> = (Σ_B, Q_B, q_B, {f_B}, δ_B)

Reduction: Union

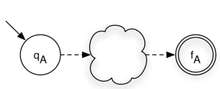
- Induction: *A*|*B*



  - <A> = (Σ_A, Q_A, q_A, {f_A}, δ_A)
  - <B> = (Σ_B, Q_B, q_B, {f_B}, δ_B)
  - <A|B> = ($\Sigma_A \cup \Sigma_B$, $Q_A \cup Q_B \cup$ {S0,S1}, S0, {S1},
    $\delta_A \cup \delta_B \cup$ {(S0,ε,q_A), (S0,ε,q_B), (f_A,ε,S1), (f_B,ε,S1)})

Reduction: Closure

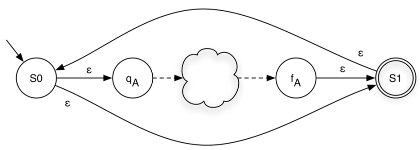
- Induction: *A**



  - <A> = (Σ_A, Q_A, q_A, {f_A}, δ_A)

Reduction: Closure

- Induction: *A**



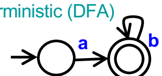
- <*A*> = (Σ_A, Q_A, q_A, {f_A}, δ_A)
- <*A**> = (Σ_A, $Q_A \cup$ {S0,S1}, S0, {S1},
  $\delta_A \cup$ {(f_A,ε,S1), (S0,ε,q_A), (S0,ε,S1), (S1,ε,S0)})

CMSC 330 Spring 2024

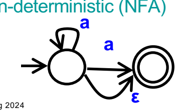
25

Recap

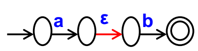
- Finite automata
  - Alphabet, states…
  - (Σ, Q, q_0, F, δ)
- Types
  - Deterministic (DFA)



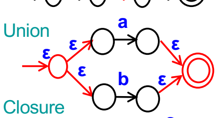
  - Non-deterministic (NFA)



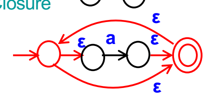
- Reducing RE to NFA
  - Concatenation



  - Union



  - Closure



CMSC 330 Spring 2024 27

27

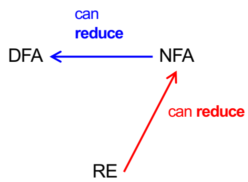
Reduction Complexity

- Given a regular expression *A* of size n...
  Size = # of symbols + # of operations
- How many states does <*A*> have?
  - Two added for each **|**, two added for each *****
  - O(n)
  - That's pretty good!

CMSC 330 Spring 2024

28

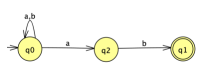
Reducing NFA to DFA



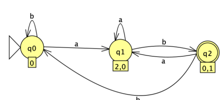
29

Why NFA → DFA

- DFA is generally more efficient than NFA



NFA



DFA

Language: (a|b)*ab

30

Why NFA → DFA

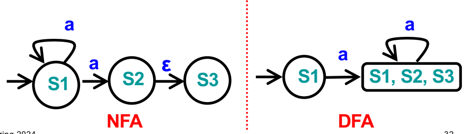
- DFA has the same expressive power as NFAs.
  - Let language $L \subseteq \Sigma^*$, and suppose L is accepted by NFA $N = (\Sigma, Q, q_0, F, \delta)$. There exists a DFA $D= (\Sigma, Q', q'_0, F', \delta')$ that also accepts L. $(L(N) = L(D))$
- NFAs are more flexible and easier to build. But DFAs have no less power than NFAs.

NFA ↔ DFA

31

Reducing NFA to DFA

- NFA may be reduced to DFA
  - By explicitly tracking the set of NFA states
- Intuition
  - Build DFA where
    - Each DFA state represents a set of NFA "current states"
- Example



32

Algorithm for Reducing NFA to DFA

- Reduction applied using the subset algorithm
  - DFA state is a subset of set of all NFA states
- Algorithm
  - Input
    - NFA $(\Sigma, Q, q_0, F_n, \delta)$
  - Output
    - DFA $(\Sigma, R, r_0, F_d, \delta)$
  - Using two subroutines
    - ε-closure(δ, p) (and ε-closure(δ, Q))
    - move(δ, p, σ) (and move(δ, Q, σ))
      - (where p is an NFA state)

33

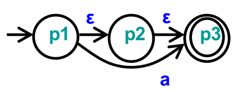
ε-transitions and ε-closure

- We say $p \xrightarrow{\varepsilon} q$
  - If it is possible to go from state p to state q by taking only ε-transitions in δ
  - If $\exists\, p, p_1, p_2, \ldots p_n, q \in Q$ such that
    - $\{p,\varepsilon,p_1\} \in \delta$, $\{p_1,\varepsilon,p_2\} \in \delta$, … , $\{p_n,\varepsilon,q\} \in \delta$
- ε-closure(δ, p)
  - Set of states reachable from p using ε-transitions alone
    - Set of states q such that $p \xrightarrow{\varepsilon} q$ according to δ
    - ε-closure(δ, p) = $\{q \mid p \xrightarrow{\varepsilon} q \text{ in } \delta\}$
    - ε-closure(δ, Q) = $\{ q \mid p \in Q,\ p \xrightarrow{\varepsilon} q \text{ in } \delta \}$
  - Notes
    - ε-closure(δ, p) always includes p
    - We write ε-closure(p) or ε-closure(Q) when δ is clear from context

34

ε-closure: Example 1

- Following NFA contains
  - p1 $\xrightarrow{\varepsilon}$ p2
  - p2 $\xrightarrow{\varepsilon}$ p3
  - p1 $\xrightarrow{\varepsilon}$ p3
    - Since p1 $\xrightarrow{\varepsilon}$ p2 and p2 $\xrightarrow{\varepsilon}$ p3



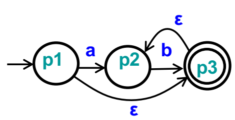
- ε-closures
  - ε-closure(p1) = { p1, p2, p3 }
  - ε-closure(p2) = { p2, p3 }
  - ε-closure(p3) = { p3 }
  - ε-closure({ p1, p2 }) = { p1, p2, p3 } ∪ { p2, p3 }

CMSC 330 Spring 2024 35

35

ε-closure: Example 2

- Following NFA contains
  - p1 $\xrightarrow{\varepsilon}$ p3
  - p3 $\xrightarrow{\varepsilon}$ p2
  - p1 $\xrightarrow{\varepsilon}$ p2
    - Since p1 $\xrightarrow{\varepsilon}$ p3 and p3 $\xrightarrow{\varepsilon}$ p2



- ε-closures
  - ε-closure(p1) = { p1, p2, p3 }
  - ε-closure(p2) = { p2 }
  - ε-closure(p3) = { p2, p3 }
  - ε-closure({ p2,p3 }) = { p2 } ∪ { p2, p3 }

CMSC 330 Spring 2024 36

36

ε-closure Algorithm: Approach

- Input: NFA (Σ, Q, q_0, F_n, δ), State Set R
- Output: State Set R'
- Algorithm

```
Let R' = R                                   // start states
Repeat
    Let R = R'                               // continue from previous
    Let R' = R ∪ {q | p ∈ R, (p, ε, q) ∈ δ}   // new ε-reachable states
Until R = R'                                 // stop when no new states
```

This algorithm computes a fixed point

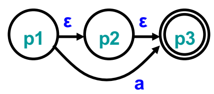
CMSC 330 Spring 2024 37

37

ε-closure Algorithm Example

- Calculate ε-closure(δ,{p1})

R	R'
{p1}	{p1}
{p1}	{p1, p2}
{p1, p2}	{p1, p2, p3}
{p1, p2, p3}	{p1, p2, p3}



Let R' = R
Repeat
  Let R= R'
  Let R' = R ∪ {q | p ∈ R, (p, ε, q) ∈ δ}
Until R = R'

38

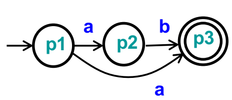
Calculating move(p,σ)

- move(δ,p,σ)
  - Set of states reachable from p using exactly one transition on symbol σ
    - Set of states q such that {p, σ, q} ∈ δ
    - move(δ,p,σ) = { q | {p, σ, q} ∈ δ }
    - move(δ,Q,σ) = { q | p ∈ Q, {p, σ, q} ∈ δ }
      - i.e., can "lift" move() to a *set* of states Q
  - Notes:
    - move(δ,p,σ) is Ø if no transition (p,σ,q) ∈ δ, for any q
    - We write move(p,σ) or move(R,σ) when δ clear from context

39

move(p,σ) : Example 1

- Following NFA
  - Σ = { a, b }



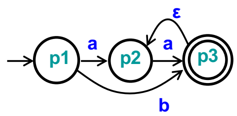
- Move
  - move(p1, a) = { p2, p3 }
  - move(p1, b) = Ø
  - move(p2, a) = Ø
  - move(p2, b) = { p3 }
  - move(p3, a) = Ø
  - move(p3, b) = Ø

move({p1,p2},b) = { p3 }

40

move(p,σ) : Example 2

- Following NFA
  - $\Sigma = \{ a, b \}$



- Move
  - move(p1, a) = { p2 }
  - move(p1, b) = { p3 }
  - move(p2, a) = { p3 }
  - move(p2, b) = Ø
  - move(p3, a) = Ø
  - move(p3, b) = Ø

move({p1,p2},a) = {p2,p3}

41

NFA → DFA Reduction Algorithm ("subset")

- Input NFA $(\Sigma, Q, q_0, F_n, \delta)$, Output DFA $(\Sigma, R, r_0, F_d, \delta')$
- Algorithm

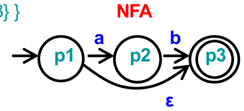
```
Let r0 = ε-closure(δ,q0), add it to R        // DFA start state
While ∃ an unmarked state r ∈ R              // process DFA state r
    Mark r                                   // each state visited once
    For each σ ∈ Σ                           // for each symbol σ
        Let E = move(δ,r,σ)                  // states reached via σ
        Let e = ε-closure(δ,E)               // states reached via ε
        If e ∉ R                             // if state e is new
            Let R = R ∪ {e}                  // add e to R (unmarked)
        Let δ' = δ' ∪ {r, σ, e}              // add transition r→e on σ
Let Fd = {r | ∃ s ∈ r with s ∈ Fn}           // final if include state in Fn
```


42

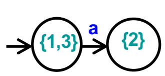
NFA → DFA Example

- Start = ε-closure(δ,p1) = { {p1,p3} }
- R = { {p1,p3} }
- r ∈ R = {p1,p3}
- move(δ,{p1,p3},a) = {p2}
  - e = ε-closure(δ,{p2}) = {p2}
  - R = R ∪ {{p2}} = { {p1,p3}, {p2} }
  - δ' = δ' ∪ {{p1,p3}, a, {p2}}
- move(δ,{p1,p3},b) = Ø

NFA



DFA

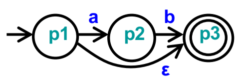


43

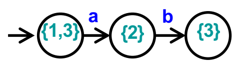
NFA → DFA Example (cont.)

- R = { {p1,p3}, {p2} }
- r ∈ R = {p2}
- move(δ,{p2},a) = Ø
- move(δ,{p2},b) = {p3}
  - e = ε-closure(δ,{p3}) = {p3}
  - R = R ∪ {{p3}} = { {p1,p3}, {p2}, {p3} }
  - δ' = δ' ∪ {{p2}, b, {p3}}

NFA



DFA

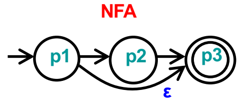


44

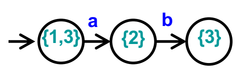
NFA → DFA Example (cont.)

- R = { {p1,p3}, {p2}, {p3} }
- r ∈ R = {p3}
- Move({p3},a) = Ø
- Move({p3},b) = Ø
- Mark {p3}, exit loop
- F_d = {{p1,p3}, {p3}}
  - Since p3 ∈ F_n
- Done!

NFA



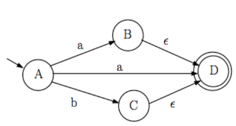
DFA



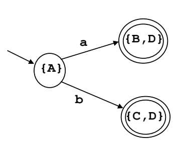
45

NFA → DFA Example 2

- NFA



- DFA

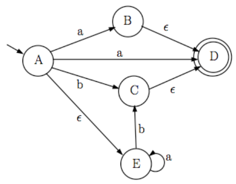


R = { {A}, {B,D}, {C,D} }

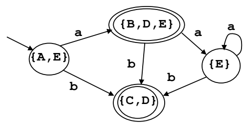
46

NFA → DFA Example 3

- NFA



- DFA

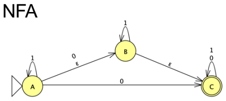


R = { {A,E}, {B,D,E}, {C,D}, {E} }

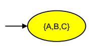
47

Detailed NFA → DFA Example

NFA



DFA

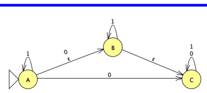


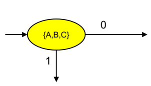
New Start State

Let r_0 = ε-closure(δ,q_0), add it to R
While ∃ an unmarked state r ∈ R
    Mark r
    For each σ ∈ Σ
    Let E = move(δ,r,σ)
        Let e = ε-closure(δ,E)
        If e ∉ R
            Let R = R ∪ {e}
        Let δ' = δ' ∪ {r, σ, e}
Let F_d = {r | ∃ s ∈ r with s ∈ F_n}

48

Detailed NFA → DFA Example



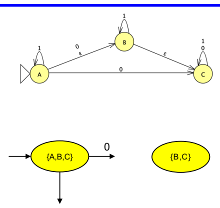


Let r_0 = ε-closure(δ,q_0), add it to R
While ∃ an unmarked state r ∈ R
    Mark r
    For each σ ∈ Σ //0
    Let E = move(δ,r,σ)
        Let e = ε-closure(δ,E)
        If e ∉ R
            Let R = R ∪ {e}
        Let δ' = δ' ∪ {r, σ, e}
Let F_d = {r | ∃ s ∈ r with s ∈ F_n}

	0	1
{A,B,C}		

49

Detailed NFA $\rightarrow$ DFA Example

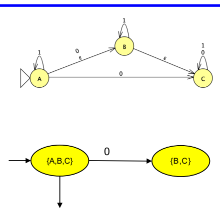


Let $r_0 = \varepsilon\text{-closure}(\delta, q_0)$, add it to R
While $\exists$ an unmarked state $r \in R$
    Mark r
    For each $\sigma \in \Sigma$
    Let E = move(δ,r,σ)
        Let e = ε-closure(δ,E)
        If $e \notin R$
            Let $R = R \cup \{e\}$
        Let $\delta' = \delta' \cup \{r, \sigma, e\}$
Let $F_d = \{r \mid \exists\, s \in r \text{ with } s \in F_n\}$

	0	1
{A,B,C}	{B,C}	

50

Detailed NFA $\rightarrow$ DFA Example

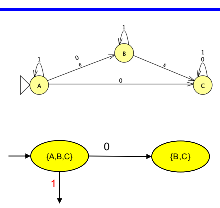


Let $r_0 = \varepsilon\text{-closure}(\delta, q_0)$, add it to R
While $\exists$ an unmarked state $r \in R$
    Mark r
    For each $\sigma \in \Sigma$
    Let E = move(δ,r,σ)
        Let e = ε-closure(δ,E)
        If $e \notin R$
            Let $R = R \cup \{e\}$
        Let $\delta' = \delta' \cup \{r, \sigma, e\}$
Let $F_d = \{r \mid \exists\, s \in r \text{ with } s \in F_n\}$

	0	1
{A,B,C}	{B,C}	
{B,C}		

51

Detailed NFA $\rightarrow$ DFA Example

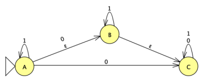


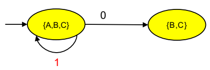
Let $r_0 = \varepsilon\text{-closure}(\delta, q_0)$, add it to R
While $\exists$ an unmarked state $r \in R$
    Mark r
    For each $\sigma \in \Sigma$ //1
    Let E = move(δ,r,σ)
        Let e = ε-closure(δ,E)
        If $e \notin R$
            Let $R = R \cup \{e\}$
        Let $\delta' = \delta' \cup \{r, \sigma, e\}$
Let $F_d = \{r \mid \exists\, s \in r \text{ with } s \in F_n\}$

	0	1
{A,B,C}	{B,C}	
{B,C}		

52

Detailed NFA → DFA Example





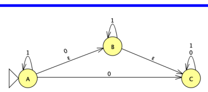
Let r_0 = ε-closure(δ,q_0), add it to R
While ∃ an unmarked state r ∈ R
    Mark r
    For each σ ∈ Σ   //1
    Let E = move(δ,r,σ)
        Let e = ε-closure(δ,E)
        If e ∉ R
            Let R = R ∪ {e}
        Let δ' = δ' ∪ {r, σ, e}
Let F_d = {r | ∃ s ∈ r with s ∈ F_n}

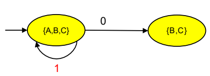
	0	1
{A,B,C}	{B,C}	{A,B,C}
{B,C}		

CMSC 330 Spring 2024

53

Detailed NFA → DFA Example





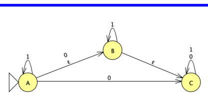
Let r_0 = ε-closure(δ,q_0), add it to R
While ∃ an unmarked state r ∈ R
    Mark r
    For each σ ∈ Σ   //1
    Let E = move(δ,r,σ)
        Let e = ε-closure(δ,E)
        If e ∉ R
            Let R = R ∪ {e}
        Let δ' = δ' ∪ {r, σ, e}
Let F_d = {r | ∃ s ∈ r with s ∈ F_n}

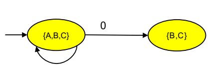
	0	1
{A,B,C}	{B,C}	{A,B,C}
{B,C}		

CMSC 330 Spring 2024

54

Detailed NFA → DFA Example





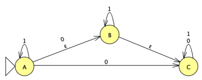
Let r_0 = ε-closure(δ,q_0), add it to R
While ∃ an unmarked state r ∈ R
    Mark r
    For each σ ∈ Σ   //1
    Let E = move(δ,r,σ)
        Let e = ε-closure(δ,E)
        If e ∉ R
            Let R = R ∪ {e}
        Let δ' = δ' ∪ {r, σ, e}
Let F_d = {r | ∃ s ∈ r with s ∈ F_n}

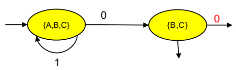
	0	1
{A,B,C}	{B,C}	{A,B,C}
{B,C}		

CMSC 330 Spring 2024

55

Detailed NFA → DFA Example





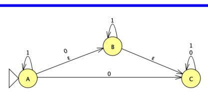
Let r_0 = ε-closure(δ,q_0), add it to R
While ∃ an unmarked state r ∈ R
Mark r
For each σ ∈ Σ //0
Let E = move(δ,r,σ)
Let e = ε-closure(δ,E)
If e ∉ R
Let R = R ∪ {e}
Let δ' = δ' ∪ {r, σ, e}
Let F_d = {r | ∃ s ∈ r with s ∈ F_n}

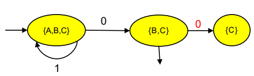
	0	1
{A,B,C}	{B,C}	{A,B,C}
{B,C}		

CMSC 330 Spring 2024

56

Detailed NFA → DFA Example





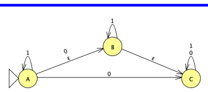
Let r_0 = ε-closure(δ,q_0), add it to R
While ∃ an unmarked state r ∈ R
Mark r
For each σ ∈ Σ //0
Let E = move(δ,r,σ)
Let e = ε-closure(δ,E)
If e ∉ R
Let R = R ∪ {e}
Let δ' = δ' ∪ {r, σ, e}
Let F_d = {r | ∃ s ∈ r with s ∈ F_n}

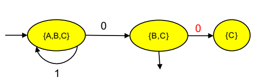
	0	1
{A,B,C}	{B,C}	{A,B,C}
{B,C}	{C}	

CMSC 330 Spring 2024

57

Detailed NFA → DFA Example





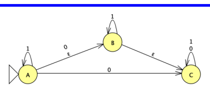
Let r_0 = ε-closure(δ,q_0), add it to R
While ∃ an unmarked state r ∈ R
Mark r
For each σ ∈ Σ //0
Let E = move(δ,r,σ)
Let e = ε-closure(δ,E)
If e ∉ R
Let R = R ∪ {e}
Let δ' = δ' ∪ {r, σ, e}
Let F_d = {r | ∃ s ∈ r with s ∈ F_n}

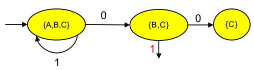
	0	1
{A,B,C}	{B,C}	{A,B,C}
{B,C}	{C}	
{C}		

CMSC 330 Spring 2024

58

Detailed NFA → DFA Example



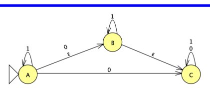


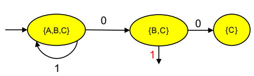

```
Let r0 = ε-closure(δ,q0), add it to R
While ∃ an unmarked state r ∈ R
    Mark r
    For each σ ∈ Σ          //1
    Let E = move(δ,r,σ)
        Let e = ε-closure(δ,E)
        If e ∉ R
            Let R = R ∪ {e}
        Let δ' = δ' ∪ {r, σ, e}
Let Fd = {r | ∃ s ∈ r with s ∈ Fn}
```

	0	1
{A,B,C}	{B,C}	{A,B,C}
{B,C}	{C}	?
{C}		

59

Detailed NFA → DFA Example



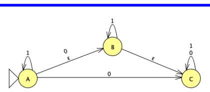


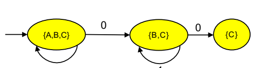

```
Let r0 = ε-closure(δ,q0), add it to R
While ∃ an unmarked state r ∈ R
    Mark r
    For each σ ∈ Σ          //1
    Let E = move(δ,r,σ)
        Let e = ε-closure(δ,E)
        If e ∉ R
            Let R = R ∪ {e}
        Let δ' = δ' ∪ {r, σ, e}
Let Fd = {r | ∃ s ∈ r with s ∈ Fn}
```

	0	1
{A,B,C}	{B,C}	{A,B,C}
{B,C}	{C}	{B,C}
{C}		

60

Detailed NFA → DFA Example



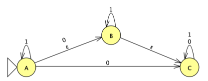


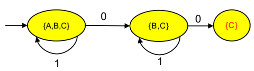

```
Let r0 = ε-closure(δ,q0), add it to R
While ∃ an unmarked state r ∈ R
    Mark r
    For each σ ∈ Σ          //1
    Let E = move(δ,r,σ)
        Let e = ε-closure(δ,E)
        If e ∉ R
            Let R = R ∪ {e}
        Let δ' = δ' ∪ {r, σ, e}
Let Fd = {r | ∃ s ∈ r with s ∈ Fn}
```

	0	1
{A,B,C}	{B,C}	{A,B,C}
{B,C}	{C}	{B,C}
{C}		

61

Detailed NFA → DFA Example

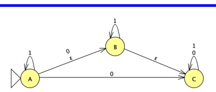


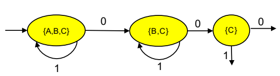


Let r_0 = ε-closure(δ,q_0), add it to R
While ∃ an unmarked state r ∈ R
Mark r
For each σ ∈ Σ //1
Let E = move(δ,r,σ)
Let e = ε-closure(δ,E)
If e ∉ R
Let R = R ∪ {e}
Let δ' = δ' ∪ {r, σ, e}
Let F_d = {r | ∃ s ∈ r with s ∈ F_n}

	0	1
{A,B,C}	{B,C}	{A,B,C}
{B,C}	{C}	{B,C}
{C}		

Detailed NFA → DFA Example

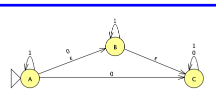


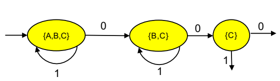


Let r_0 = ε-closure(δ,q_0), add it to R
While ∃ an unmarked state r ∈ R
Mark r
For each σ ∈ Σ
Let E = move(δ,r,σ)
Let e = ε-closure(δ,E)
If e ∉ R
Let R = R ∪ {e}
Let δ' = δ' ∪ {r, σ, e}
Let F_d = {r | ∃ s ∈ r with s ∈ F_n}

	0	1
{A,B,C}	{B,C}	{A,B,C}
{B,C}	{C}	{B,C}
{C}		

Detailed NFA → DFA Example

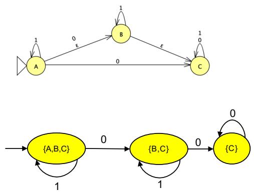




Let r_0 = ε-closure(δ,q_0), add it to R
While ∃ an unmarked state r ∈ R
Mark r
For each σ ∈ Σ //0
Let E = move(δ,r,σ)
Let e = ε-closure(δ,E)
If e ∉ R
Let R = R ∪ {e}
Let δ' = δ' ∪ {r, σ, e}
Let F_d = {r | ∃ s ∈ r with s ∈ F_n}

	0	1
{A,B,C}	{B,C}	{A,B,C}
{B,C}	{C}	{B,C}
{C}	{C}	

Detailed NFA $\rightarrow$ DFA Example



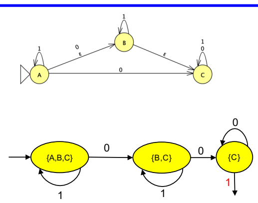
Let $r_0 = \varepsilon\text{-closure}(\delta, q_0)$, add it to R
While $\exists$ an unmarked state $r \in R$
Mark r
For each $\sigma \in \Sigma$ //0
Let E = move(δ,r,σ)
Let e = ε-closure(δ,E)
If $e \notin R$
Let $R = R \cup \{e\}$
Let $\delta' = \delta' \cup \{r, \sigma, e\}$
Let $F_d = \{r \mid \exists s \in r \text{ with } s \in F_n\}$

	0	1
{A,B,C}	{B,C}	{A,B,C}
{B,C}	{C}	{B,C}
{C}	{C}	

CMSC 330 Spring 2024

65

Detailed NFA $\rightarrow$ DFA Example



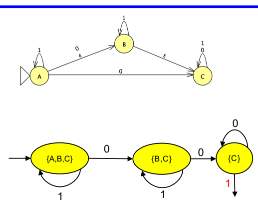
Let $r_0 = \varepsilon\text{-closure}(\delta, q_0)$, add it to R
While $\exists$ an unmarked state $r \in R$
Mark r
For each $\sigma \in \Sigma$ //1
Let E = move(δ,r,σ)
Let e = ε-closure(δ,E)
If $e \notin R$
Let $R = R \cup \{e\}$
Let $\delta' = \delta' \cup \{r, \sigma, e\}$
Let $F_d = \{r \mid \exists s \in r \text{ with } s \in F_n\}$

	0	1
{A,B,C}	{B,C}	{A,B,C}
{B,C}	{C}	{B,C}
{C}	{C}	

CMSC 330 Spring 2024

66

Detailed NFA $\rightarrow$ DFA Example



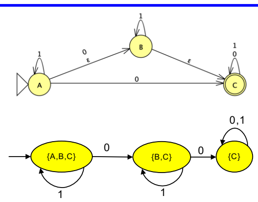
Let $r_0 = \varepsilon\text{-closure}(\delta, q_0)$, add it to R
While $\exists$ an unmarked state $r \in R$
Mark r
For each $\sigma \in \Sigma$ //1
Let E = move(δ,r,σ)
Let e = ε-closure(δ,E)
If $e \notin R$
Let $R = R \cup \{e\}$
Let $\delta' = \delta' \cup \{r, \sigma, e\}$
Let $F_d = \{r \mid \exists s \in r \text{ with } s \in F_n\}$

	0	1
{A,B,C}	{B,C}	{A,B,C}
{B,C}	{C}	{B,C}
{C}	{C}	{C}

CMSC 330 Spring 2024

67

Detailed NFA → DFA Example



Let r_0 = ε-closure(δ,q_0), add it to R

While ∃ an unmarked state r ∈ R

Mark r

For each σ ∈ Σ //1

Let E = move(δ,r,σ)

Let e = ε-closure(δ,E)

If e ∉ R

Let R = R ∪ {e}

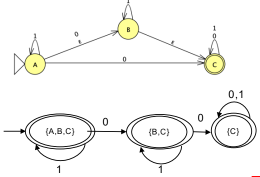
Let δ' = δ' ∪ {r, σ, e}

Let F_d = {r | ∃ s ∈ r with s ∈ F_n}

	0	1
{A,B,C}	{B,C}	{A,B,C}
{B,C}	{C}	{B,C}
{C}	{C}	{C}

68

Detailed NFA → DFA Example



Let r_0 = ε-closure(δ,q_0), add it to R

While ∃ an unmarked state r ∈ R

Mark r

For each σ ∈ Σ //1

Let E = move(δ,r,σ)

Let e = ε-closure(δ,E)

If e ∉ R

Let R = R ∪ {e}

Let δ' = δ' ∪ {r, σ, e}

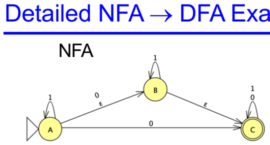
Let F_d = {r | ∃ s ∈ r with s ∈ F_n}

	0	1
{A,B,C}	{B,C}	{A,B,C}
{B,C}	{C}	{B,C}
{C}	{C}	{C}

69

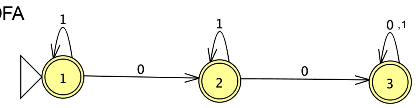
Detailed NFA → DFA Example: Completed

NFA



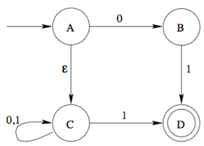
	0	1
{A,B,C}	{B,C}	{A,B,C}
{B,C}	{C}	{B,C}
{C}	{C}	{C}

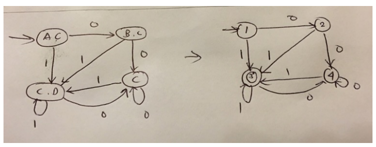
DFA



70

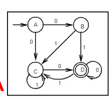
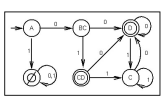
NFA → DFA Example





Analyzing the Reduction

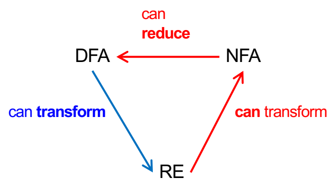
- Can reduce any NFA to a DFA using subset alg.
- How many states in the DFA?
  - Each DFA state is a subset of the set of NFA states
  - Given NFA with n states, DFA may have 2^n states
    - Since a set with n items may have 2^n subsets
  - Corollary
    - Reducing a NFA with n states may be $O(2^n)$

NFA   DFA

Recap: Matching a Regexp *R*

- Given *R*, construct NFA. Takes time O(*R*)
- Convert NFA to DFA. Takes time $O(2^{|R|})$
  - But usually not the worst case in practice
- Use DFA to accept/reject string s
  - Assume we can compute $\delta(q,\sigma)$ in constant time
  - Then time to process s is O(|s|)
    - Can't get much faster!
- Constructing the DFA is a one-time cost
  - But then processing strings is fast

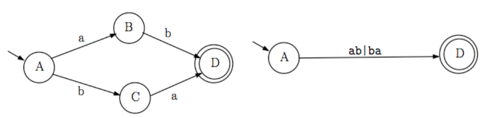
Closing the Loop: Reducing DFA to RE



81

Reducing DFAs to REs

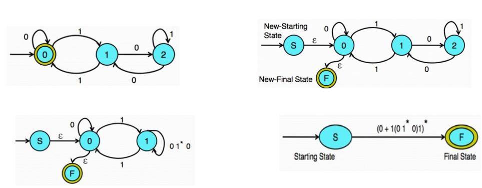
- General idea
  - Remove states one by one, labeling transitions with regular expressions
  - When two states are left (start and final), the transition label is the regular expression for the DFA



82

DFA to RE example

Language over Σ= {0,1} such that every string is a multiple of 3 in binary

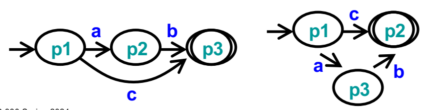


$$(0 + 1(0\ 1^*\ 0)1)^*$$

83

Minimizing DFAs

- Every regular language is recognizable by a unique minimum-state DFA
  - Ignoring the particular names of states
- In other words
  - For every DFA, there is a unique DFA with minimum number of states that accepts the same language



85

J. Hopcroft, "An n log n algorithm for minimizing states in a finite automaton," 1971

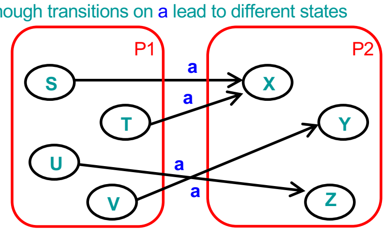
Minimizing DFA: Hopcroft Reduction

- Intuition
  - Look to distinguish states from each other
    - End up in different accept / non-accept state with identical input
- Algorithm
  - Construct initial partition
    - Accepting & non-accepting states
  - Iteratively split partitions (until partitions remain fixed)
    - Split a partition if **members in partition have transitions to different partitions for same input**
      - Two states x, y belong in same partition if and only if for all symbols in Σ they transition to the same partition
  - Update transitions & remove dead states

86

Splitting Partitions

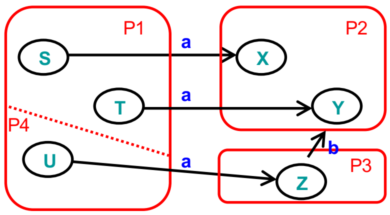
- No need to split partition {S,T,U,V}
  - All transitions on a lead to identical partition P2
  - Even though transitions on a lead to different states



87

Splitting Partitions (cont.)

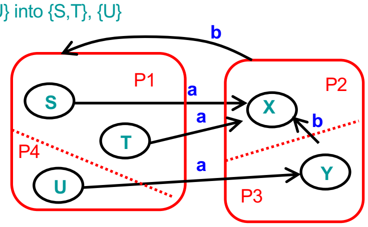
- Need to split partition {S,T,U} into {S,T}, {U}
  - Transitions on a from S,T lead to partition P2
  - Transition on a from U lead to partition P3



88

Resplitting Partitions

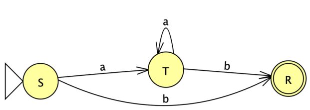
- Need to reexamine partitions after splits
  - Initially no need to split partition {S,T,U}
  - After splitting partition {X,Y} into {X}, {Y} we need to split partition {S,T,U} into {S,T}, {U}



89

Minimizing DFA: Example 1

- DFA

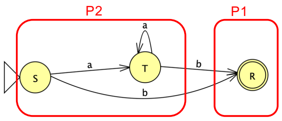


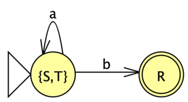
- Initial partitions
- Split partition

90

Minimizing DFA: Example 1

- DFA

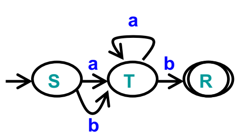




- Initial partitions
  - Accept { R } = P1
  - Reject { S, T } = P2
- Split partition? → Not required, minimization done
  - move(S,a) = T ∈ P2 – move(S,b) = R ∈ P1
  - move(T,a) = T ∈ P2 – move (T,b) = R ∈ P1

91

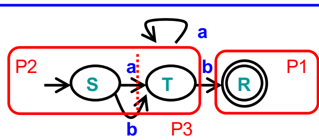
Minimizing DFA: Example 2



92

Minimizing DFA: Example 2

- DFA



DFA already minimal

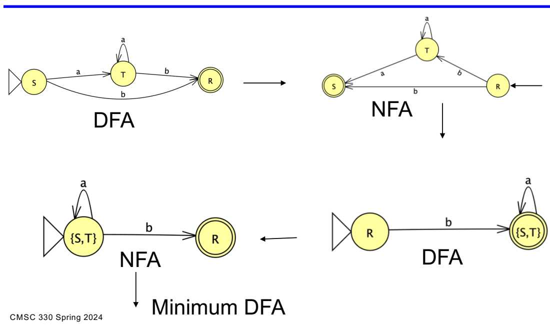
- Initial partitions
  - Accept { R }= P1
  - Reject { S, T } = P2
- Split partition? → Yes, different partitions for B
  - move(S,a) = T ∈ P2 – move(S,b) = T ∈ P2
  - move(T,a) = T ∈ P2 – move (T,b) = R ∈ P1

93

Brzozowski's Algorithm: DFA Minimization

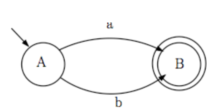
1. Given a DFA, reverse all the edges, make the initial state an accept state, and the accept states initial, to get an NFA
2. NFA-> DFA
3. For the new DFA, reverse the edges (and initial-accept swap) get an NFA
4. NFA -> DFA

Brzozowski's algorithm



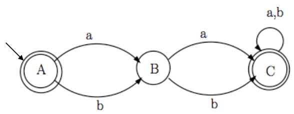
Complement of DFA

- Given a DFA accepting language L
  - How can we create a DFA accepting its complement?
  - Example DFA
    - $\Sigma = \{a,b\}$



Complement of DFA

- Algorithm
  - Add explicit transitions to a dead state
  - Change every accepting state to a non-accepting state & every non-accepting state to an accepting state
- Note this only works with DFAs
  - Why not with NFAs?



99

Summary of Regular Expression Theory

- Finite automata
  - DFA, NFA
- Equivalence of RE, NFA, DFA
  - RE → NFA
    - Concatenation, union, closure
  - NFA → DFA
    - ε-closure & subset algorithm
- DFA
  - Minimization, complementation

100