

CMSC 330: Organization of Programming Languages

Finite Automata 2

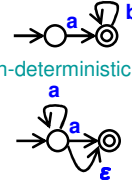
Last Lecture

► Finite automata

- Alphabet, states...
- $(\Sigma, Q, q_0, F, \delta)$

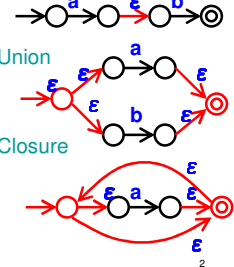
► Types

- Deterministic (DFA)
- Non-deterministic (NFA)



► Reducing RE to NFA

- Concatenation
- Union
- Closure



CMSC 330

2

This Lecture

► Reducing NFA to DFA

- ϵ -closure
- Subset algorithm

► Minimizing DFA

- Hopcroft reduction

► Complementing DFA

► Implementing DFA

CMSC 330

3

How NFA Works

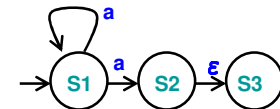
► When NFA processes a string

- NFA may be in several possible states
 - > Multiple transitions with same label
 - > ϵ -transitions

► Example

- After processing "a"
 - > NFA may be in states

- S1
- S2
- S3



CMSC 330

4

Reducing NFA to DFA

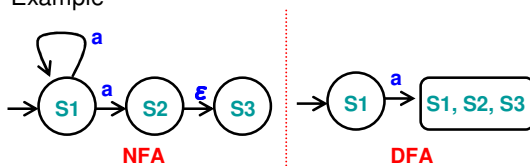
► NFA may be reduced to DFA

- By explicitly tracking the set of NFA states

► Intuition

- Build DFA where
 - > Each DFA state represents a set of NFA states

► Example



CMSC 330

5

Reducing NFA to DFA (cont.)

► 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
 - > ϵ -closure(p)
 - > move(p, a)

CMSC 330

6

ϵ -transitions and ϵ -closure

- ▶ We say $p \xrightarrow{\epsilon} q$
 - If it is possible to go from state p to state q by taking only ϵ -transitions
 - If $\exists p, p_1, p_2, \dots, p_n, q \in Q$ such that
 - > $\{p, \epsilon, p_1\} \in \delta, \{p_1, \epsilon, p_2\} \in \delta, \dots, \{p_n, \epsilon, q\} \in \delta$
- ▶ ϵ -closure(p)
 - Set of states reachable from p using ϵ -transitions alone
 - > Set of states q such that $p \xrightarrow{\epsilon} q$
 - > ϵ -closure(p) = $\{q \mid p \xrightarrow{\epsilon} q\}$
 - Note
 - > ϵ -closure(p) always includes p
 - > ϵ -closure(\cdot) may be applied to set of states (take union)

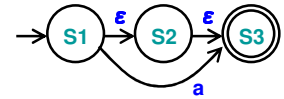
CMSC 330

7

ϵ -closure: Example 1

- ▶ Following NFA contains

- $S1 \xrightarrow{\epsilon} S2$
- $S2 \xrightarrow{\epsilon} S3$
- $S1 \xrightarrow{\epsilon} S3$



- ▶ ϵ -closures

- ϵ -closure($S1$) = $\{S1, S2, S3\}$
- ϵ -closure($S2$) = $\{S2, S3\}$
- ϵ -closure($S3$) = $\{S3\}$
- ϵ -closure($\{S1, S2\}$) = $\{S1, S2, S3\} \cup \{S2, S3\}$

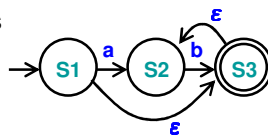
CMSC 330

8

ϵ -closure: Example 2

- ▶ Following NFA contains

- $S1 \xrightarrow{\epsilon} S3$
- $S3 \xrightarrow{\epsilon} S2$
- $S1 \xrightarrow{\epsilon} S2$



- ▶ ϵ -closures

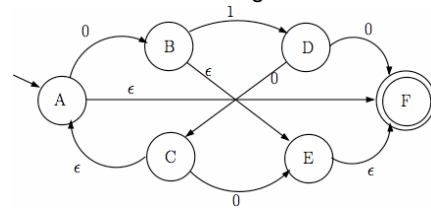
- ϵ -closure($S1$) = $\{S1, S2, S3\}$
- ϵ -closure($S2$) = $\{S2\}$
- ϵ -closure($S3$) = $\{S2, S3\}$
- ϵ -closure($\{S2, S3\}$) = $\{S2\} \cup \{S2, S3\}$

CMSC 330

9

ϵ -closure: Practice

- ▶ Find ϵ -closures for following NFA



- ▶ Find ϵ -closures for the NFA you construct for
 - The regular expression $(01^*1)111(0^*1)$

CMSC 330

10

Calculating $move(p, a)$

- ▶ $move(p, a)$

- Set of states reachable from p using exactly one transition on a
 - > Set of states q such that $\{p, a, q\} \in \delta$
 - > $move(p, a) = \{q \mid \{p, a, q\} \in \delta\}$
- Note $move(p, a)$ may be empty \emptyset
 - > If no transition from p with label a

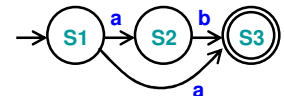
CMSC 330

11

$move(a, p)$: Example 1

- ▶ Following NFA

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



- ▶ Move

- $move(S1, a) = \{S2, S3\}$
- $move(S1, b) = \emptyset$
- $move(S2, a) = \emptyset$
- $move(S2, b) = \{S3\}$
- $move(S3, a) = \emptyset$
- $move(S3, b) = \emptyset$

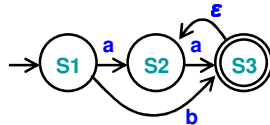
CMSC 330

12

move(a,p) : Example 2

► Following NFA

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



► Move

- $\text{move}(S1, a) = \{ S2 \}$
- $\text{move}(S1, b) = \{ S3 \}$
- $\text{move}(S2, a) = \{ S3 \}$
- $\text{move}(S2, b) = \emptyset$
- $\text{move}(S3, a) = \emptyset$
- $\text{move}(S3, b) = \emptyset$

CMSC 330

13

NFA → DFA Reduction Algorithm

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

► Algorithm

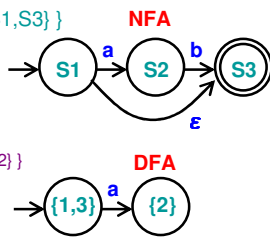
- Let $r_0 = \epsilon\text{-closure}(q_0)$, add it to R // DFA start state
- While \exists an unmarked state $r \in R$ // process DFA state r
 - > Mark r // each state visited once
 - > For each letter $a \in \Sigma$ // for each letter a
 - Let $S = \{ s \mid q \in r \ \& \ \text{move}(q,a) = s \}$ // states reached via a
 - Let $e = \epsilon\text{-closure}(S)$ // states reached via ϵ
 - If $e \notin R$ // if state e is new
 - Let $R = e \cup R$ // add e to R (unmarked)
 - Let $\delta = \delta \cup \{ r, a, e \}$ // add transition $r \rightarrow e$
- Let $F_d = \{ r \mid \exists s \in r \ \text{with} \ s \in F_n \}$ // final if include state in F_n

CMSC 330

14

NFA → DFA Example 1

- $\text{Start} = \epsilon\text{-closure}(S1) = \{ \{S1, S3\} \}$
- $R = \{ \{S1, S3\} \}$
- $r \in R = \{S1, S3\}$
- $\text{Move}(\{S1, S3\}, a) = \{S2\}$
 - > $e = \epsilon\text{-closure}(\{S2\}) = \{S2\}$
 - > $R = R \cup \{S2\} = \{ \{S1, S3\}, \{S2\} \}$
 - > $\delta = \delta \cup \{ \{S1, S3\}, a, \{S2\} \}$
- $\text{Move}(\{S1, S3\}, b) = \emptyset$

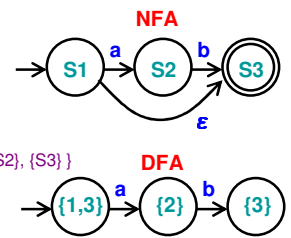


CMSC 330

15

NFA → DFA Example 1 (cont.)

- $R = \{ \{S1, S3\}, \{S2\} \}$
- $r \in R = \{S2\}$
- $\text{Move}(\{S2\}, a) = \emptyset$
- $\text{Move}(\{S2\}, b) = \{S3\}$
 - > $e = \epsilon\text{-closure}(\{S3\}) = \{S3\}$
 - > $R = R \cup \{S3\} = \{ \{S1, S3\}, \{S2\}, \{S3\} \}$
 - > $\delta = \delta \cup \{ \{S2\}, b, \{S3\} \}$

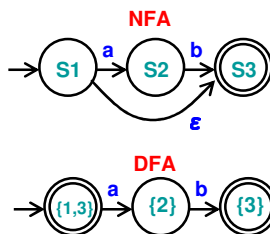


CMSC 330

16

NFA → DFA Example 1 (cont.)

- $R = \{ \{S1, S3\}, \{S2\}, \{S3\} \}$
- $r \in R = \{S3\}$
- $\text{Move}(\{S3\}, a) = \emptyset$
- $\text{Move}(\{S3\}, b) = \emptyset$
- $F_d = \{ \{S1, S3\}, \{S3\} \}$
 - > Since $S3 \in F_n$
- Done!

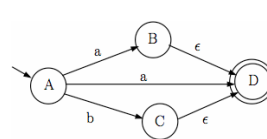


CMSC 330

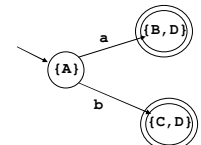
17

NFA → DFA Example 2

► NFA



► DFA



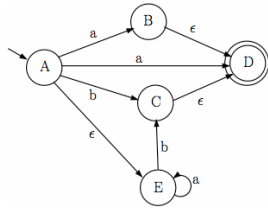
$$R = \{ \{A\}, \{B, D\}, \{C, D\} \}$$

CMSC 330

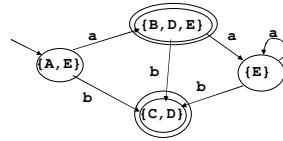
18

NFA → DFA Example 3

► NFA



► DFA



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

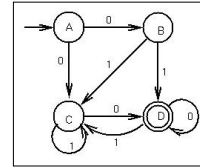
CMSC 330

19

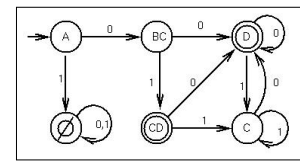
Equivalence of DFAs and NFAs

► Any string from {A} to either {D} or {CD}

- Represents a path from A to D in the original NFA



NFA



DFA

CMSC 330

20

Equivalence of DFAs and NFAs (cont.)

- 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 $O(2^n)$

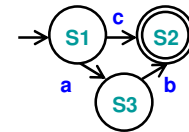
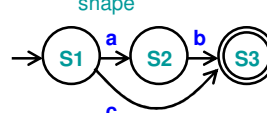
CMSC 330

21

Minimizing DFA

► Result from CS theory

- Every regular language is recognizable by a minimum-state DFA that is **unique** up to state names
- In other words
 - For every DFA, there is a unique DFA with minimum number of states that accepts the same language
 - Two minimum-state DFAs have **same** underlying shape



CMSC 330

22

Minimizing DFA: Hopcroft Reduction

► Intuition

- Look for states that can be distinguish from each other
 - > End up in different accept / non-accept state with identical input

► Algorithm

- Construct initial partition
 - > Accepting & non-accepting states
- Iteratively refine 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

CMSC 330

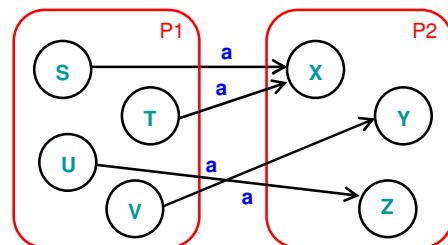
J. Hopcroft, "An $n \log n$ algorithm for minimizing states in a finite automaton," 1971

23

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

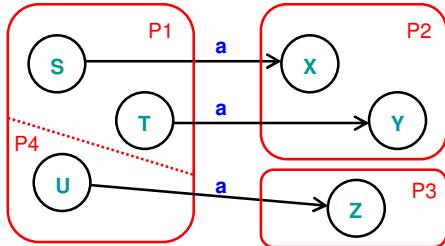


CMSC 330

24

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 R lead to partition $P3$

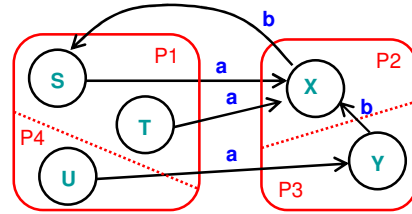


CMSC 330

25

Resplitting Partitions

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

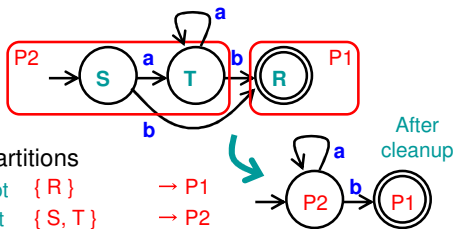


CMSC 330

26

Minimizing DFA: Example 1

- DFA



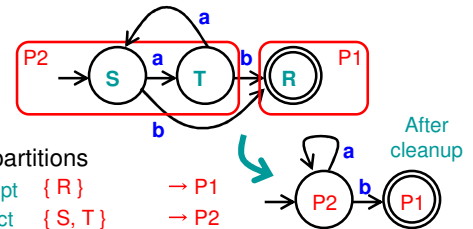
- Initial partitions
 - Accept $\{R\} \rightarrow P1$
 - Reject $\{S, T\} \rightarrow P2$
- Split partition? \rightarrow Not required, minimization done
 - $move(S, a) = T \rightarrow P2$ $move(S, b) = R \rightarrow P1$
 - $move(T, a) = T \rightarrow P2$ $move(T, b) = R \rightarrow P1$

CMSC 330

27

Minimizing DFA: Example 2

- DFA



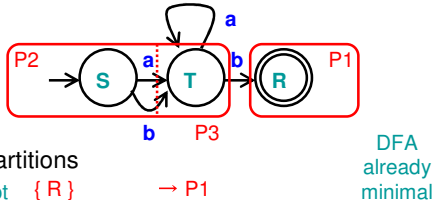
- Initial partitions
 - Accept $\{R\} \rightarrow P1$
 - Reject $\{S, T\} \rightarrow P2$
- Split partition? \rightarrow Not required, minimization done
 - $move(S, a) = T \rightarrow P2$ $move(S, b) = R \rightarrow P1$
 - $move(T, a) = S \rightarrow P2$ $move(T, b) = R \rightarrow P1$

CMSC 330

28

Minimizing DFA: Example 3

- DFA



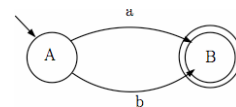
- Initial partitions
 - Accept $\{R\} \rightarrow P1$
 - Reject $\{S, T\} \rightarrow P2$
- Split partition? \rightarrow Yes, different partitions for B
 - $move(S, a) = T \rightarrow P2$ $move(S, b) = T \rightarrow P2$
 - $move(T, a) = T \rightarrow P2$ $move(T, b) = R \rightarrow P1$

CMSC 330

29

Complement of DFA

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

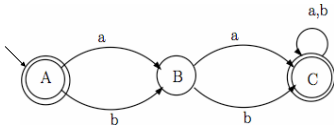


CMSC 330

30

Complement of DFA (cont.)

- ▶ 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?

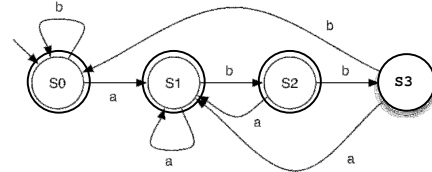


CMSC 330

31

Practice

Make the DFA which accepts the complement of the language accepted by the DFA below.

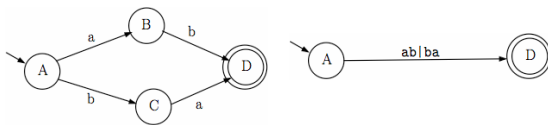


CMSC 330

32

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



CMSC 330

33

Relating REs to DFAs and NFAs

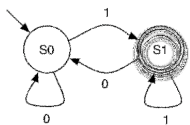
- ▶ Why do we want to convert between these?
 - Can make it easier to express ideas
 - Can be easier to implement

CMSC 330

34

Implementing DFAs

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;
    }
}
    
```

CMSC 330

35

Implementing DFAs (Alternative)

Alternatively, use generic table-driven DFA

```

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

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

CMSC 330

36

Run Time of DFA

- ▶ How long for DFA to decide to accept/reject string s ?
 - Assume we can compute $\delta(q, c)$ in constant time
 - Then time to process s is $O(|s|)$
 - Can't get much faster!
- ▶ Constructing DFA for RE A may take $O(2^{|A|})$ time
 - But usually not the case in practice
- ▶ So there's the initial overhead
 - But then processing strings is fast

CMSC 330

37

Regular Expressions in Practice

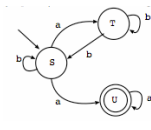
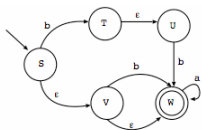
- ▶ Regular expressions are typically "compiled" into tables for the generic algorithm
 - Can think of this as a simple byte code interpreter
 - But really just a representation of $(\Sigma, Q_A, q_A, \{f_A\}, \delta_A)$, the components of the DFA produced from the RE
- ▶ Regular expression implementations often have extra constructs that are non-regular
 - I.e., can accept more than the regular languages
 - Can be useful in certain cases
 - Disadvantages
 - Nonstandard, plus can have higher complexity

CMSC 330

38

Practice

- ▶ Convert to a DFA



- ▶ Convert to an NFA and then to a DFA
 - $(0|1)^*11|0^*$
 - Strings of alternating 0 and 1
 - $aba^*|(ba|b)$

CMSC 330

39

Summary of Regular Expression Theory

- ▶ Finite automata
 - DFA, NFA
- ▶ Equivalence of RE, NFA, DFA
 - RE \rightarrow NFA
 - Concatenation, union, closure
 - NFA \rightarrow DFA
 - ϵ -closure & subset algorithm
- ▶ DFA
 - Minimization, complement
 - Implementation

CMSC 330

40