CMSC 330: Organization of Programming Languages

DFA, and NFAs, and Regexps

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

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
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

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

DFA for \((a|b)^*abb\)
NFA for \((ab|aba)^*\)

- **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 \((ab|aba)^*\)
Formal Definition

- A deterministic finite automaton (DFA) is a 5-tuple $(\Sigma, Q, q_0, F, \delta)$ where
  - $\Sigma$ 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 $\delta$ is?

A DFA accepts $s$ if it stops at a final state on $s$

Formal Definition: Example

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

Or as $(S0,0,S0), (S0,1,S1), (S1,0,S0), (S1,1,S1)$

Implementing DFAs (one-off)

It’s easy to build a program which mimics a DFA
Implementing DFAs (generic)

More generally, use generic table-driven DFA

given components \((\Sigma, Q, q_0, F, d)\) of a DFA:

\[
\begin{align*}
\text{let } q &= q_0; \\
\text{while (there exists another symbol } &\sigma \text{ of the input string):} \\
q &= d(q, \sigma); \\
\text{if } q &\in F \text{ then accept} \\
\text{else reject}
\end{align*}
\]

- \(q\) is just an integer
- Represent \(\delta\) using arrays or hash tables
- Represent \(F\) as a set

---

Nondeterministic Finite Automata (NFA)

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

\[
\begin{align*}
\text{Example:} \\
\Sigma &= \{a\} \\
Q &= \{S1, S2, S3\} \\
q_0 &= S1 \\
F &= \{S3\} \\
\delta &= \{(S1, a, S1), (S1, a, S2), (S2, \epsilon, S3)\}
\end{align*}
\]

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

---

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 \(\epsilon\)-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!
Relating REs to DFAs and NFAs

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

Reduction to NFAs

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

Reducing Regular Expressions to NFAs

- Base case: $\sigma$

\[<\sigma> = (\emptyset, \{S_0, S_1\}, S_0, \{S_1\}, \{(S_0, \sigma, S_1)\})\]
Reduction

- Base case: ε
  \[ \langle \varepsilon \rangle = (\emptyset, \{S0\}, S0, \{S0\}, \emptyset)\]
- Base case: \(\emptyset\)
  \[\langle \emptyset \rangle = (\emptyset, \{S0, S1\}, S0, \{S1\}, \emptyset)\]

Reduction: Concatenation

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

Reduction: Concatenation

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

Induction: \( A | B \)

- \(<A> = (\Sigma_A, Q_A, q_0, \delta_A)\)
- \(<B> = (\Sigma_B, Q_B, q_0, \delta_B)\)

Reduction: Closure

Induction: \( A^* \)

- \(<A> = (\Sigma_A, Q_A, q_0, \delta_A)\)
Reduction: Closure

- Induction: $A^*$

  - $<A> = (\Sigma, Q_A, q_0, \delta_A)$
  - $<A^*> = (\Sigma, Q_A \cup \{S0, S1\}, S0, \{S1\})$
  - $\delta_A \cup \{(f_{\epsilon, r, S1}), (S0, r, q_A), (S0, r, S1)), (S1, r, S0))\}$

Recap

- Finite automata
  - Alphabet, states...
  - $(\Sigma, Q, q_0, F, \delta)$
- Types
  - Deterministic (DFA)
  - Non-deterministic (NFA)

Reducing RE to NFA

- Concatenation
- Union
- Closure

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!
Reducing NFA to DFA

DFA can reduce NFA

RE can reduce

Why NFA → DFA

• DFA is generally more efficient than NFA

Language: (a|b)*ab

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
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

```
S1 a S1, S2, S3
S2 a S1
S3 a S1, S2, S3
```

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 (Σ, Q, q₀, Fₙ, δ)
- Output
  - DFA (Σ, R, r₀, Fₙ, d)
- 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)

ε-transitions and ε-closure

- We say p ε q
  - If it is possible to go from state p to state q by taking only ε-transitions in δ
  - If ∃ p₁, p₂, ..., q ∈ Q such that
    - (p₁, p₂, ..., q) ∈ δ

ε-closure(δ, p)

- Set of states reachable from p using ε-transitions alone
  - Set of states q such that p ε q according to δ
  - ε-closure(δ, p) = { q | p ε q in δ }

- Notes
  - ε-closure(δ, p) always includes p
  - We write ε-closure(p) or ε-closure(Q) when δ is clear from context
**ε-closure: Example 1**

- Following NFA contains
  - \( p_1 \xrightarrow{\epsilon} p_2 \)
  - \( p_2 \xrightarrow{\epsilon} p_3 \)
  - \( p_1 \xrightarrow{\epsilon} p_3 \)

\( \epsilon \)-closures

- \( \epsilon \)-closure\( (p_1) = \{ p_1, p_2, p_3 \} \)
- \( \epsilon \)-closure\( (p_2) = \{ p_2, p_3 \} \)
- \( \epsilon \)-closure\( (p_3) = \{ p_3 \} \)
- \( \epsilon \)-closure\( (\{ p_1, p_2 \}) = \{ p_1, p_2, p_3 \} \cup \{ p_2, p_3 \} \)

**ε-closure: Example 2**

- Following NFA contains
  - \( p_1 \xrightarrow{\epsilon} p_3 \)
  - \( p_3 \xrightarrow{\epsilon} p_2 \)
  - \( p_1 \xrightarrow{\epsilon} p_2 \)

\( \epsilon \)-closures

- \( \epsilon \)-closure\( (p_1) = \{ p_1, p_2, p_3 \} \)
- \( \epsilon \)-closure\( (p_2) = \{ p_2 \} \)
- \( \epsilon \)-closure\( (p_3) = \{ p_2, p_3 \} \)
- \( \epsilon \)-closure\( (\{ p_2, p_3 \}) = \{ p_2 \} \cup \{ p_2, p_3 \} \)

**ε-closure Algorithm: Approach**

- **Input:** NFA \( (\Sigma, Q, q_0, F, \delta) \), State Set \( R \)
- **Output:** State Set \( R' \)
- **Algorithm**
  
  Let \( R' = R \)  
  Repeat
  
  Let \( R = R' \)  
  Let \( R' = R \cup \{ q | p \in R, (p, c, q) \in \delta \} \)  
  Until \( R = R' \)

This algorithm computes a fixed point.
ε-closure Algorithm Example

Calculate ε-closure(ε(p1))

\[ \begin{align*}
R & \quad R' \\
p1 & \quad (p1) \\
p1 & \quad (p1, p2) \\
p1, p2 & \quad (p1, p2, p3) \\
p1, p2, p3 & \quad (p1, p2, p3)
\end{align*} \]

Let \( R' = R \)
Repeat
Let \( R' = R' \cup \{ (p \in R, (p, \varepsilon, q) \in \delta) \} \)
Until \( R = R' \)

Calculating \( \text{move}(p, \sigma) \)

\( \text{move}(\delta, p, \sigma) \)
- Set of states reachable from \( p \) using exactly one transition on symbol \( \sigma \)
  - Set of states \( q \) such that \( (p, \sigma, q) \in \delta \)
  - \( \text{move}(\delta, p, \sigma) = \{ q | (p, \sigma, q) \in \delta \} \)
  - \( \text{move}(\delta, Q, \sigma) = \{ q | p \in Q, (p, \sigma, q) \in \delta \} \)
    - i.e., can "lift" \( \text{move}() \) to a set of states \( Q \)

Notes:
- \( \text{move}(\delta, p, \sigma) \) is \( \emptyset \) if no transition \( (p, \sigma, q) \in \delta \), for any \( q \)
- We write \( \text{move}(p, \sigma) \) or \( \text{move}(R, \sigma) \) when \( \delta \) clear from context

move(p, σ) : Example 1

Following NFA
- \( \Sigma = \{ a, b \} \)

Move
- \( \text{move}(p1, a) = (p2, p3) \)
- \( \text{move}(p1, b) = \emptyset \)
- \( \text{move}(p2, a) = \emptyset \)
- \( \text{move}(p2, b) = (p3) \)
- \( \text{move}(p3, a) = \emptyset \)
- \( \text{move}(p3, b) = \emptyset \)
move(p,σ): Example 2

Following NFA

- \( \Sigma = \{ a, b \} \)

Move

- \( \text{move}(p_1, a) = \{ p_2 \} \)
- \( \text{move}(p_1, b) = \{ p_3 \} \)
- \( \text{move}(p_2, a) = \{ p_3 \} \)
- \( \text{move}(p_2, b) = \emptyset \)
- \( \text{move}(p_3, a) = \emptyset \)
- \( \text{move}(p_3, b) = \emptyset \)

NFA \( \rightarrow \) DFA Reduction Algorithm ("subset")

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

Algorithm

Let \( r_0 = \varepsilon - \text{closure}(\delta, q_0) \), add it to \( R \) // DFA start state

While \( \exists \) an unmarked state \( r \subseteq R \)

Mark \( r \) // process DFA state \( r \)

For each \( \sigma \in \Sigma \)

Let \( E = \text{move}(\delta, r, \sigma) \) // states reached via \( \sigma \)

Let \( e = \varepsilon - \text{closure}(\delta, E) \) // states reached via \( \varepsilon \)

If \( e \notin R \) // if state \( e \) is new

Let \( R = R \cup \{ e \} \) // add \( e \) to \( R \) (unmarked)

Let \( \delta' = \delta' \cup \{ r \rightarrow e \} \) // add transition \( r \rightarrow e \) on \( \sigma \)

Let \( F_d = \{ r \mid s \in r \text{ with } s \in F_n \} \) // final if include state in \( F_n \)

NFA \( \rightarrow \) DFA Example

- Start = \( \varepsilon - \text{closure}(\delta, p_1) = \{ p_1, p_3 \} \)
- \( R = \{ p_1, p_3 \} \)
- \( r \subseteq R = \{ p_1, p_3 \} \)
- \( \text{move}(\delta, (p_1, p_3), a) = \{ p_2 \} \)
- \( e = \varepsilon - \text{closure}(\delta, p_2) = \{ p_2 \} \)
- \( R = R \cup \{ p_2 \} = \{ p_1, p_2, p_3 \} \)
- \( \delta' = \delta' \cup \{ (p_1, p_3), a, (p_2) \} \)
- \( \text{move}(\delta, (p_1, p_3), b) = \emptyset \)
NFA $\rightarrow$ DFA Example (cont.)

- $R = \{ (p_1,p_3), (p_2) \}$
- $r \in R = (p_2)$
- $\text{move}(\delta, (p_2), a) = \emptyset$
- $\text{move}(\delta, (p_2), b) = (p_3)$
  - $e = \text{closure}(\delta, (p_3)) = (p_3)$
  - $R = R \cup \{ (p_3) \} = \{ (p_1,p_3), (p_2), (p_3) \}$
  - $\delta' = \delta \cup \{ (p_2, b, (p_3)) \}$

NFA

$\begin{array}{c}
\delta
\end{array}$

DFA

$\begin{array}{c}
\delta'
\end{array}$

$\text{move}(\delta', (p_3), a) = \emptyset$

$\text{move}(\delta', (p_3), b) = (p_3)$

$\text{Mark} (p_3), \text{exit loop}$

$F_d = \{ (p_1,p_3), (p_3) \}$

Since $p_3 \in F_d$

$\text{Done!}$

NFA $\rightarrow$ DFA Example 2

$R = \{ \{A\}, \{B,D\}, \{C,D\} \}$
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 $\sigma \in S$

Let $E = \text{move}(\delta, r, \sigma)$

Let $e = \varepsilon - \text{closure}(\delta, E)$

If $e \in R$

Let $R = R - \{e\}$

Let $F = \{r \mid \exists s \in r \text{ with } s \in F_n\}$

Let $d' = d' - \{r, \sigma, e\}$
Let \( r_0 = \epsilon - \text{closure}(\delta, q_0) \), add it to \( R \)

While \( \exists \) an unmarked state \( r \in R \)

Mark \( r \)

For each \( \sigma \in S \)

Let \( E = \text{move}(\delta, r, \sigma) \)

Let \( e = \text{closure}(\delta, E) \)

If \( e \notin R \)

Let \( R = R \cup \{ e \} \)

Let \( d' = d' \cup \{ r, \sigma, e \} \)

Let \( F_d = \{ r | \exists s \in r \text{ with } s \in F_n \} \)
Let $r_0 = e - \text{closure}(\delta, q_0)$, add it to $R$

While $\exists$ an unmarked state $r \in R$

Mark $r$

For each $\sigma \in S$

Let $E = \text{move}(\delta, r, \sigma)$

Let $e = e - \text{closure}(\delta, E)$

If $e \notin R$

Let $R = R \cup \{e\}$

Let $d' = d' \cup \{r, \sigma, e\}$

Let $F_d = \{ r \mid s \in r \text{ with } s \in F_n \}$

{A,B,C} 0

{B,C}

{A,B,C}

{B,C}

{A,B,C}

{B,C} 1
Let $r_0 = e - \text{closure}(\delta, q_0)$, add it to $R$

While an unmarked state $r \in R$

- Mark $r$

For each $\sigma \in \Sigma$

Let $E = \text{move}(\delta, r, \sigma)$

Let $e = e - \text{closure}(\delta, E)$

If $e \notin R$

Let $R = R \cup \{e\}$

Let $d' = d' \cup \{r, \sigma, e\}$

Let $F_d = \{r \mid s \in r \text{ with } s \in F_n\}$

---

### Detailed NFA → DFA Example

Let $m = \text{closure}(\delta, q_0)$, add it to $R$

While an unmarked state $r \in R$

- Mark $r$

For each $\sigma \in \Sigma$

Let $E = \text{move}(\delta, r, \sigma)$

Let $e = e - \text{closure}(\delta, E)$

If $e \notin R$

Let $R = R \cup \{e\}$

Let $F_d = \{r \mid s \in r \text{ with } s \in F_n\}$
Let $r_0 = e - \text{closure}(\delta, q_0)$, add it to $R$.

While $\exists$ an unmarked state $r \in R$, mark $r$.

For each $\sigma \in \Sigma$, $\theta$.

Let $E = \text{move}(\delta, r, \sigma)$.

Let $e = e - \text{closure}(\delta, E)$.

If $e \notin R$

Let $R = R \cup \{e\}$.

Let $d' = d' \cup \{r, \sigma, e\}$.

Let

$$F_d = \{ r \mid s \in r \text{ with } s \in F_n \}.$$
Let $r_0 = e - \text{closure}(\delta, q_0)$, add it to $R$.
While \(\exists\) an unmarked state \(r \in R\)
Mark \(r\)
For each \(\sigma \in \Sigma\)
Let \(E = \text{move}(\delta, r, \sigma)\)
Let \(e = e - \text{closure}(\delta, E)\)
If \(e \not\in R\)
Let \(R = R \cup \{e\}\)
Let \(F_d = \{ s \mid \exists r \in R \text{ with } s \in F^n_d \}\)
Let $r_0 = e - \text{closure}(-, q_0)$, add it to $R$

While there is an unmarked state $r \in R$

Mark $r$

For each $\sigma \in S$

Let $E = \text{move}(\delta, r, \sigma)$

Let $e = e - \text{closure}(\delta, E)$

If $e \not\in R$

Let $R = R \cup \{e\}$

Let $d' = d' \cup \{r, \sigma, e\}$

Let $F_d = \{r \mid s \in r \text{ with } s \in F_n\}$
Let $r_0 = e - \text{closure}(\delta, q_0)$, add it to $R$.

While there is an unmarked state $r \in R$:
Mark $r$.
For each $\sigma \in \Sigma$:// 1
Let $E = \text{move}(\delta, r, \sigma)$
Let $e = e - \text{closure}(\delta, E)$
If $e \notin R$
Let $R = R \cup \{e\}$
Let $F_{d'} = \{s \mid \exists \sigma \in \Sigma \text{ with } s \in F_n\}$

Detailed NFA → DFA Example

Let $r_0 = e - \text{closure}(\delta, q_0)$, add it to $R$.

While there is an unmarked state $r \in R$:
Mark $r$.
For each $\sigma \in \Sigma$:// 1
Let $E = \text{move}(\delta, r, \sigma)$
Let $e = e - \text{closure}(\delta, E)$
If $e \notin R$
Let $R = R \cup \{e\}$
Let $F_{d'} = \{s \mid \exists \sigma \in \Sigma \text{ with } s \in F_n\}$

NFA

\[
\begin{array}{c|c|c|c}
\text{A} & \text{B} & \text{C} \\
\hline
0 & 0,1 & 0,1 \\
\end{array}
\]

DFA

\[
\begin{array}{c|c|c|c}
\text{A} & \text{B} & \text{C} \\
\hline
0 & 0,1 & 0,1 \\
\end{array}
\]
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 \( \mathcal{O}(2^n) \)

Recap: Matching a Regexp \( R \)

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

**DFA to RE example**
Language over $\Sigma = \{0,1\}$ such that every string is a multiple of 3 in binary

\[(0 + 1(01^*)01^*)^*\]
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.

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

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

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
Splitting Partitions (cont.)

- Need to split partition \(\{S, T, U\}\) into \(\{S, T\}, \{U\}\)
  - Transitions on \(a\) from \(S, T\) lead to partition \(P_2\)
  - Transition on \(a\) from \(U\) lead to partition \(P_3\)

```
```

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 \(\{S, T\}, \{U\}\)

```
```

Minimizing DFA: Example 1

- DFA
- Initial partitions
- Split partition
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

P1
P2

Minimizing DFA: Example 2

DFA

- 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

P1
P2
P3
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

Complement of DFA

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

[Diagram of DFA and its complement]
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?

---

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