L05: Subset Construction (Pre Lecture)

Dr. Neil T. Dantam

CSCI-561, Colorado School of Mines

Fall 2025

Introduction

Introduction

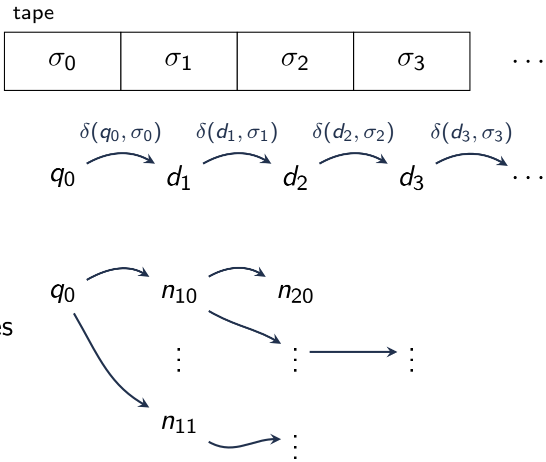
- DFA and NFA both represent Regular Languages
- Equivalent DFA using subsets of NFA states

Outcomes

- Understand ε-closures
- Understand NFA simulation
- Understand subset construction algorithm
- Understand equivalence proof of NFA and DFA

Determinism vs. Nondeterminism

Review



Deterministic

Transition to single state

$$\delta_{\mathrm{dfa}}(q_i, \sigma) = q_j$$

Nondeterministic

Transition to a multiple states

$$\delta_{\mathrm{nfa}}(q_i, \sigma) = \{q_j, q_k, q_\ell, \ldots\}$$

DFA vs. NFA

Review

Definition (DFA)

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

- Q is a finite set called the **states**
- Σ is a finite set called the **alphabet**
- $\delta : Q \times \Sigma \mapsto Q$ is the transition function
- $q_0 \in Q$ is the start state
- $F \subseteq Q$ is the set of accept states

Definition (NFA)

A nondeterministic finite automaton is a 5-tuple, $N = (Q, \Sigma, \delta, q_0, F)$, where,

- Q is a finite set called the **states**
- Σ is a finite set called the **alphabet**
- $\delta : Q \times \Sigma \mapsto \mathcal{P}(Q)$ is the transition function
- $q_0 \in Q$ is the start state
- $F \subseteq Q$ is the set of accept states

Equivalence of NFA and DFA

Proof Outline

Theorem

For any NFA N, there exists an **equivalent** *DFA M that recognizes the same language:* $\mathcal{L}(N) = \mathcal{L}(M)$.

Proof Outline.

1. Each NFA transition results in set of states that could hold.
2. Thus, at each step we may be in a subset of the NFA states Q.
3. If the NFA has k states, then there are 2^k subsets.
4. There is an equivalent DFA whose states of those 2^k subsets.

$\square$

NFA to DFA Overview

Input: NFA $N = (Q_N, \Sigma, \delta_N, q_{0,N}, F_N)$

Output: DFA $M = (Q_M, \Sigma, \delta_M, q_{0,N}, F_M)$, such that $\mathcal{L}(M) = \mathcal{L}(N)$

Solution: DFA states are subsets of NFA states

	NFA	**DFA**
states	Q_N	$Q_M = \mathcal{P}(Q_N)$
alphabet	Σ	Σ
transition	$\delta_N : Q_N \times \Sigma \mapsto \mathcal{P}(Q_N)$	$\delta_M(Q, \sigma) \triangleq \{ q \in Q_N \mid \sigma\text{-reachable from } q_i \in Q \}$
start	$q_{0,N} \in Q_N$	$q_{0,M} = \{ q \in Q_N \mid \varepsilon\text{-reachable from } q_{0,N} \}$
accept	$F_N \subseteq Q_N$	$F_M = \{ q_m \in Q_M \mid q_m \cap F_N \neq \emptyset \}$

Hey! That's exponential!

and thus wildly impractical

- Powerset is exponential
  - $|Q| = k$
  - $|\mathcal{P}(Q)| = 2^k$
- But typically, don't actually need every subset in $\mathcal{P}(Q)$

In practice, we can usually construct DFA with fewer states.

Outline

ε Closures

- ε-closure
- move-ε-closure

NFA Simulation

Subset Construction Algorithm

Outline

ε Closures
- ε-closure
- move-ε-closure

NFA Simulation

Subset Construction Algorithm

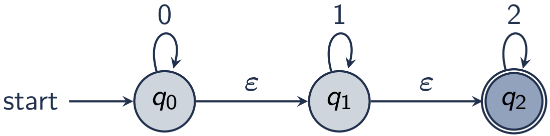
ε Closure Subroutines

ε–closure(q) States reachable from q on ε transitions

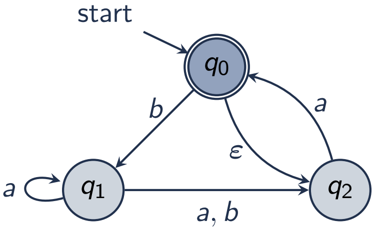
move-ε–closure(q, σ) States reachable from state q after reading symbol σ

Construct the reachable subsets of NFA states

Example: ε-closure



- $\varepsilon\text{-closure}(q_0) = \{q_0, q_1, q_2\}$
- $\varepsilon\text{-closure}(q_1) = \{q_1, q_2\}$
- $\varepsilon\text{-closure}(q_2) = \{q_2\}$



- $\varepsilon\text{-closure}(q_0) = \{q_0, q_2\}$
- $\varepsilon\text{-closure}(q_1) = \{q_1\}$
- $\varepsilon\text{-closure}(q_2) = \{q_2\}$

States reachable from initial state on only ε transitions

Idea: ε–closure

- **Input:** Initial set of states
- **Output:** Set of states (the closure) reachable only on ε transitions
- **Approach:** Recursively visit states and "accumulate" the ε–closure

  Base case: Visiting a state already contained in the ε–closure (set of states)

  Recursive case: Add the current state and recurse on all its ε–neighbors

Algorithm: ε–closure

Algorithm 1: ε-closure(NFA,S,C)

Input: NFA, S, C ; // NFA, unvisited states, initial closure

Output: C' ; // final closure

function *visit(c,q)* **is** // $\mathcal{P}(Q) \times Q \mapsto \mathcal{P}(Q)$

  if $q \in c$ **then** // base case, state q already in closure c

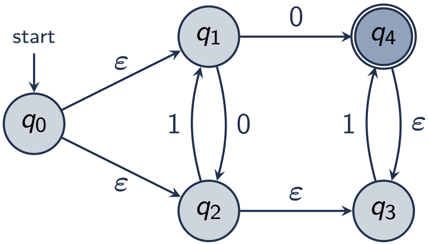
    return c;

  else // Recursive case, visit all ε-successors of state q

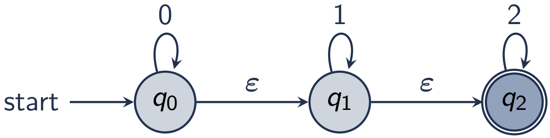
    return $\varepsilon\text{–closure}\left(\text{NFA}, \left(\underbrace{\delta(q,\varepsilon)}_{\varepsilon\text{-neighbors of } q}\right), \left(\underbrace{\{q\} \cup c}_{\text{add } q \text{ to closure}}\right)\right)$;

$C' \leftarrow$ **fold-left**(visit, C, S) ; // Visit all states in S

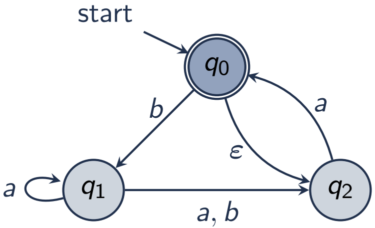
Exercise 1: ε–closure



Example: move-ε-closure



- move-ε-closure$(q_0, 0) = \{q_0, q_1, q_2\}$
- move-ε-closure$(q_0, 1) = \{q_1, q_2\}$
- move-ε-closure$(q_0, 2) = \{q_2\}$



- move-ε-closure$(q_0, a) = \{q_0, q_2\}$
- move-ε-closure$(q_1, a) = \{q_1, q_2\}$
- move-ε-closure$(q_2, a) = \{q_0, q_2\}$

States reachable from initial state after reading one symbol

Idea: move-ε–closure

- **Input:** Initial state set Q and input symbol σ
- **Output:** Set of states reachable from Q after reading σ (any number of ε–transitions are allowed)
- **Approach:**
  1. Find ε–closure of initial set Q
  2. Find set of states after reading the symbol σ
  3. Find ε–closure of resulting set

Algorithm: move-ε–closure

Algorithm 2: move-ε-closure(NFA,Q,σ)

Input: NFA, Q, σ ; // NFA, initial states, token

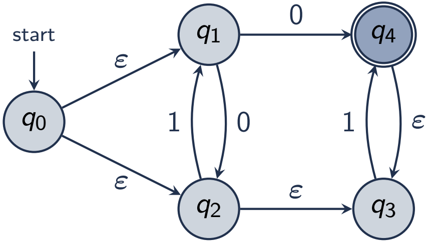
Output: C' ; // reachable states

function *visit(c,q)* **is** // c: closure (post-move), q: initial state (pre-move)

$$\textbf{return } \underbrace{\varepsilon\text{—closure}\left(\text{NFA}, \left(\overbrace{\delta(q,\sigma)}^{\sigma \text{ reachable from } q}\right), c\right)}_{\varepsilon\text{-reachable after move}};$$

$$C' \leftarrow \textbf{fold-left}\left(\text{visit}, \emptyset, \underbrace{\varepsilon\text{—closure}(\text{NFA}, Q, \emptyset)}_{\varepsilon\text{-reachable from Q}}\right);$$

Exercise 2: move–ε–closure



Outline

ε Closures
- ε-closure
- move-ε-closure

NFA Simulation

Subset Construction Algorithm

DFA Simulation

Review

- **Input:** DFA M and Input String ω
- **Output:** Does M accept ω?
- **Algorithm Outline:**
  1. Evaluate the extended transition function on input string ω.
  2. At the end of the input string:
     - If the resulting state is an accept state, return `accept`
     - Otherwise, return `reject`

Algorithm 3: DFA-Simulate

Input: $M = (Q, \Sigma, \delta, q_0, F)$; // DFA

Input: ω ; // input string

Output: $\{\texttt{accept}, \texttt{reject}\}$

```
function δ̂(q, ω) is
    if ω = ε then return q;
    else
        let
            f ← first (ω);
            r ← rest (ω);
        in return δ̂(δ(q, f), r) ;
if δ̂(q0, ω) ∈ F then return accept;
else return reject;
```

Idea: NFA Simulation

- **Input:** NFA N and Input String ω
- **Outline:** Does N accept the input string ω?
- **Algorithm Outline:**
  1. Compute ε–closure of start state as the current state (subset)
  2. While the input string contains more symbols:
     - 2.1 Read the next input symbol
     - 2.2 Compute the move–ε–closure from the current state (subset)
     - 2.3 Update the current state (subset)
  3. At the end of the input string:
     - If the current state (subset) contains an accept state, return `accept`
     - Otherwise, return `reject`

Algorithm: NFA Simulation

Algorithm 4: NFA-Simulate

Input: $N = (Q, \Sigma, \delta, q_0, F)$; // states,alphabet,transition,start,accept

Input: ω ; // input string

Output: $\{\texttt{accept}, \texttt{reject}\}$

function $\hat{\delta}(u, \omega)$ **is**

  if $\omega = \varepsilon$ **then return** u; // Base case at empty string

  else // Recurse on rest of string

    let $u' \leftarrow \texttt{move-}\varepsilon\texttt{-closure}(N, u, \texttt{first}(\omega))$ **in**

      return $\hat{\delta}(u', \texttt{rest}(\omega))$

let $u \leftarrow \varepsilon\texttt{-closure}(N, q_0, \emptyset)$ **in**

  if $\hat{\delta}(u, \omega) \cap F \neq \emptyset$ **then return** $\texttt{accept}$;

  else return $\texttt{reject}$;

Outline

ε Closures

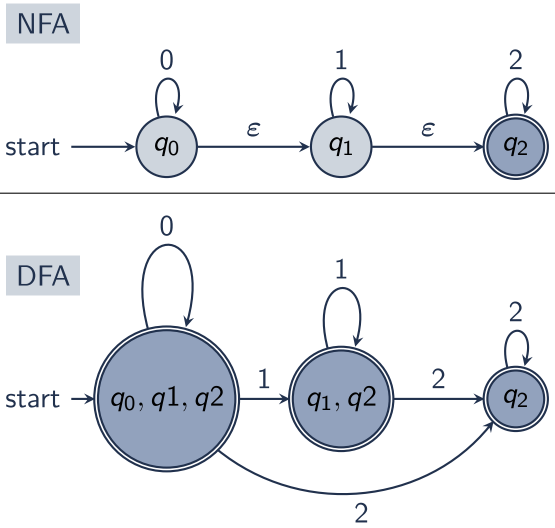
ε-closure

move-ε-closure

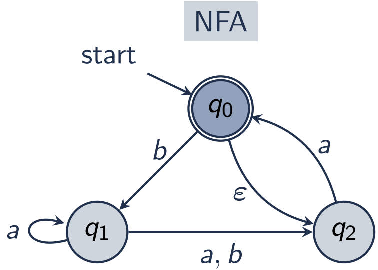
NFA Simulation

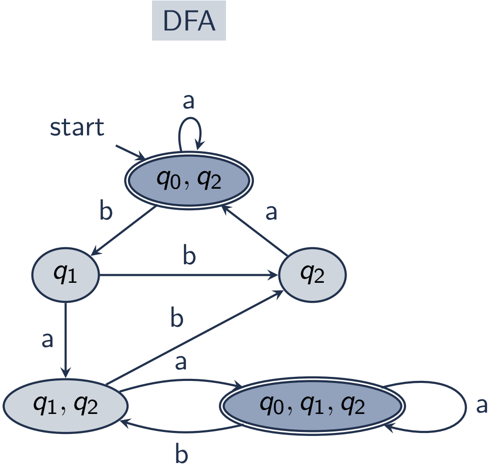
Subset Construction Algorithm

Example 0: NFA to DFA



Example 1: NFA to DFA





Idea: NFA to DFA Subset Construction

- **Input:** NFA
- **Output:** Equivalent DFA
- **Algorithm Outline:** Recursively visit subsets, beginning with ε–closure of the start state,
  1. If the current subset was already visited, return.
  2. For each symbol σ in alphabet Σ, find move–ϵ–closure from the current subset on σ.
  3. Add the resulting subset and edges, then recursively visit

Algorithm: NFA to DFA Subset Construction

Algorithm 5: NFA-to-DFA

Input: $N = (Q, \Sigma, E, q_0, F)$; `// NFA states, alphabet, edges, start, accept`

Output: $M = (Q', \Sigma, E', q_0', F')$; `// DFA states, alphabet, edges, start, accept`

function *visit-symbol*(Q', E', u, σ) **is**
- **let** $u' \leftarrow \texttt{move-}\varepsilon\texttt{-closure}(u, \sigma)$ **in** `// Find successor subset:` $u \xrightarrow{\sigma} u'$
  - **if** u' **then return** $\texttt{visit-subset}\left(Q', E' \cup \left\{u \xrightarrow{\sigma} u'\right\}, u'\right)$;
  - **else return** (Q', E');

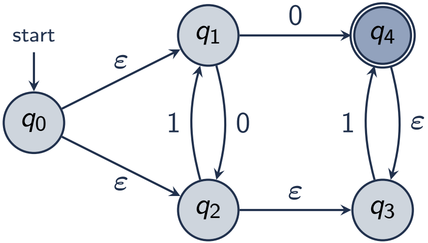
function *visit-subset*(Q', E', u) **is**
- **if** $u \in Q'$ **then return** (Q', E') ; `// Subset already constructed`
- **else**
  - **let** $Q' \leftarrow Q' \cup \{u\}$ **in**
    - **function** $h(E', \sigma)$ **is return** $\texttt{visit-symbol}(Q', E', u, \sigma)$;
    - **return fold-left**$(h, E', \Sigma \setminus \{\varepsilon\})$

$q_0' \leftarrow \varepsilon\texttt{-closure}(q_0)$;

$(Q', E') \leftarrow \texttt{visit-subset}(\emptyset, \emptyset, q_0')$;

$F' = \{q \in Q' \mid q \cap F \neq \emptyset\}$;

Exercise 3: NFA to DFA



Practical usage of DFAs vs. NFAs

- Should we use DFA or NFA?
- If we already have an NFA:
  1. Simulate NFA by tracking all possible current states (subsets)
  2. Convert NFA to DFA by constructing the subsets of NFA state set
- Why use NFA at all?
  Stay tuned!

References

Textbook: Sipser, 3rd ed.

- Ch 1.1 Finite Automata
- Ch 1.2 Nondeterminism

Alt. Textbook: Hopcroft, 3rd ed.

- Ch 2.3 Nondeterministic Finite Automata
- Ch 2.5 Finite Automata with Epsilon-Transitions

Alt. Textbook: Aho, 2nd ed.

- Ch 3.6 Finite Automata
- Ch 3.7 From Regular Expressions to Automata