Finite State Automata

COMP3220 – Principle of Programming Languages

Zhitao Gong

2016 Spring

Outline

Introduction

Man Wolf Goat Cabbage

Deterministic Finite Automata

Non-deterministic Finite Automata

Summary

Goal – Define A Language

One way To define a language

1. Construct an automaton (i.e., a kind of abstract computer) that takes a string as input and produces a *yes* or *no* answer.
2. The language it defines is the set of all strings for which it says yes.

The simplest kind of automaton is the *finite automaton*, which has a *finite* memory or *states*.

Automata Hierarchy

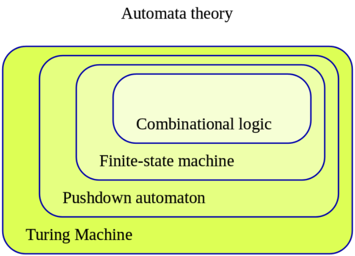


Figure: All the automata

- Combinational Logic used in computer circuits to perform Boolean algebra on input signals and on stored data. E.g., ALU.
- FSA used in communication protocol design, language parsing, etc.

Concepts and Representation

State is a description of the status of a system that is waiting to execute a transition.

Transition is a set of actions to be executed when a condition is fulfilled or when an event is received.

Representation include state/event table, UML state machine, SDL state machine, etc.

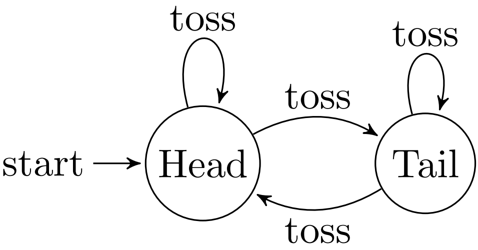


Figure: Toss A Coin

Outline

Introduction

Man Wolf Goat Cabbage

Deterministic Finite Automata

Non-deterministic Finite Automata

Summary

The Classic Riddle

- Problem
  - A man travels with wolf, goat and cabbage.
  - He wants to cross a river from east to west.
- Conditions
  - A rowboat is available, but only large enough for the man plus one possession.
  - Wolf eats goat if left alone together.
  - Goat eats cabbage if left alone together
- Goal how can the man cross *without loss*?

Solution String

Four moves may be encoded as four symbols

- Man crosses with wolf w
- Man crosses with goat g
- Man crosses with cabbage c
- Man crosses with *nothing* n

Then a sequence of moves constitute a string. E.g., $gnwgcng$: cross with goat, cross back with nothing, cross with wolf, cross back with goat, etc.

State and Transition

- State – The current items on east E and west W bank constitute the current state
- Transition – The moves are transitions, i.e., they change the current state.
- Man is represented with m which is *not* in the language.

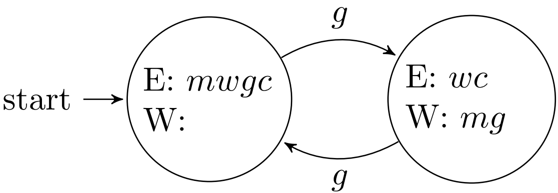
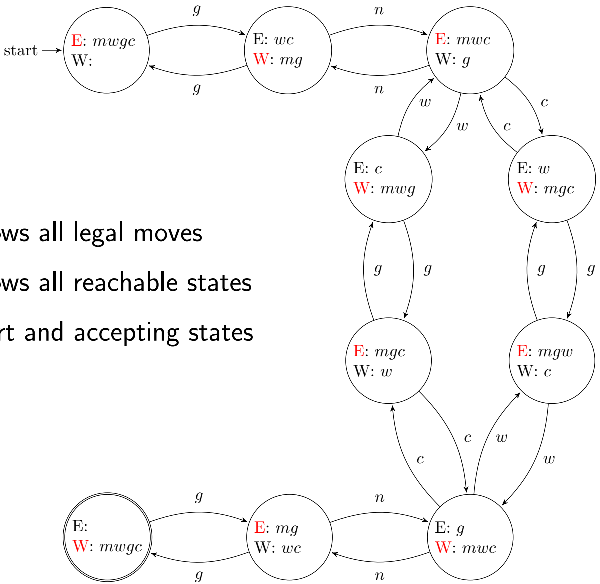


Figure: MWGC Example

Transition Diagram Version 0



- Shows all legal moves
- Shows all reachable states
- Start and accepting states

The Language of Solution

Given an alphabet $\Sigma = \{w, g, c, n\}$, every path yields a string $x \in \Sigma^*$. And all the strings constitute the language of solutions.

$$\{x \in \{w, g, c, n\}^* \mid \text{starts in the } \textit{starting state}, \text{ follows the transitions of } x \text{ and ends in the } \textit{accepting state} \}$$

The starting state is (E: $mwgc$ W:), and the accepting state is (E: W: $mwgc$)

- The language is infinite.
- Two shortest strings in the language are *gnwgcng* and *gncgwng*.

Diagram Gets Stuck

What happens to strings not in the language, e.g., *c*, *gncn*.
The previous diagram is not complete, it gets stuck when the strings are not solutions.

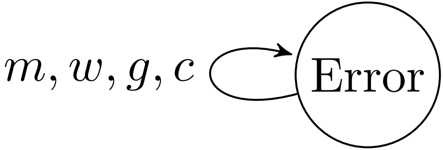
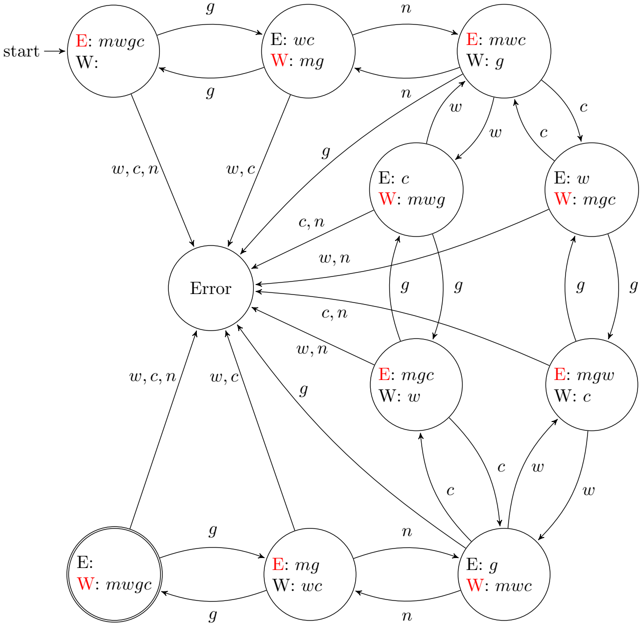


Figure: Error State

Transition Diagram Version 1



Complete Specification

- The above diagram shows exactly *one* transition from every state on every symbol in $\Sigma = \{w, g, c, n\}$.
- It gives a computational procedure for deciding whether a given string is a solution:
  1. Start in the start state, and
  2. make one transition for each symbol in the string.
  3. *Accept* if end in the accepting state, *reject* if error.

Outline

Introduction

Man Wolf Goat Cabbage

Deterministic Finite Automata

Non-deterministic Finite Automata

Summary

DFA Informal

- A diagram with *finite* number of states, represented by circles.
- Two special states, the starting state (usually pointed to by an arrow from start) and accepting state (usually double circled).
- For every state, for every symbol in Σ, there is exactly one arrow labeled with that symbol going to another state (or loop back to self)

Concepts

- Given a string over Σ, the DFA can read the string and follow its transition as denoted by the string.
- At the end of the string, if DFA reaches an accepting state, we say *it accepts the string*, otherwise *it rejects the string*.
- The language defined by a DFA is *the set of strings in* Σ^* *that it accepts*.

Example

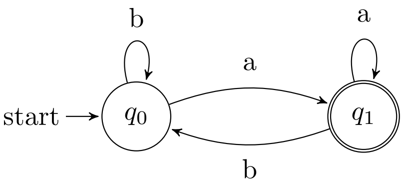


Figure: DFA Toy Example

- This DFA defines (a set-builder notation)

Example

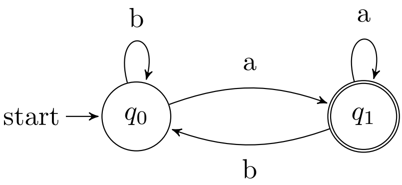
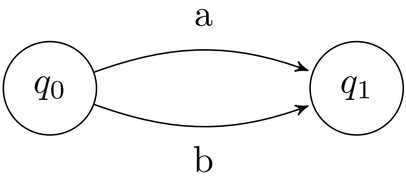


Figure: DFA Toy Example

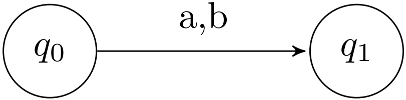
- This DFA defines (a set-builder notation) $\{xa \mid x \in \{a, b\}^*\}$.
- Meaningless states, may be omitted.

DFA Convention

- No two arrows have the same source and destination



- Instead, one arrow with list of transitions (symbols)



DFA Formal

A DFA $\mathcal{M}$ is a 5-tuple

$$\mathcal{M} = (Q, \Sigma, \delta, q_0, F)$$

- Q *finite* set of states
- Σ alphabet, i.e., *finite* set of symbols
- $\delta \in (Q \times \Sigma \to Q)$ transition function
- $q_0 \in Q$ the start state
- $F \subseteq Q$ the set of accepting states

Example

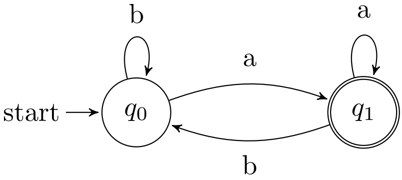


Figure: This DFA defines $\{xa \mid x \in \{a, b\}^*\}$

Formally, $\mathcal{M} = (Q, \Sigma, \delta, q_0, F)$ where

- $Q = \{q_0, q_1\}$
- $\Sigma = \{a, b\}$
- $F = q_1$
- $\delta(q_0, a) = q_1$, $\delta(q_0, b) = q_0$, $\delta(q_1, a) = q_1$, $\delta(q_1, b) = q_0$.

The δ^* Function

- δ defines 1-symbole moves
- δ^* gives whole-string results by applying zero or more δ moves

$$\delta^*(q, \epsilon) = q$$
$$\delta^*(q, xa) = \delta(\delta^*(q, x), a)$$

$\mathcal{M}$ Accepts x

$\delta^*(q, x)$ is the state $\mathcal{M}$ ends up in, starting from state q and reading all of string x

Theorem

Given a DFA $\mathcal{M} = (Q, \Sigma, \delta, q_0, F)$, a string $x \in \Sigma^*$ is accepted by $\mathcal{M}$ if and only if $\delta^*(q_0, x) \in F$.

Language Defined by DFA

For any DFA $\mathcal{M} = (Q, \Sigma, \delta, q_0, F)$, $\mathcal{L}(\mathcal{M})$ denotes the language accepted by $\mathcal{M}$

$$\mathcal{L}(\mathcal{M}) = \{x \in \Sigma^* \mid \delta^*(q_0, x) \in F\}$$

Outline

Introduction

Man Wolf Goat Cabbage

Deterministic Finite Automata

Non-deterministic Finite Automata

Summary

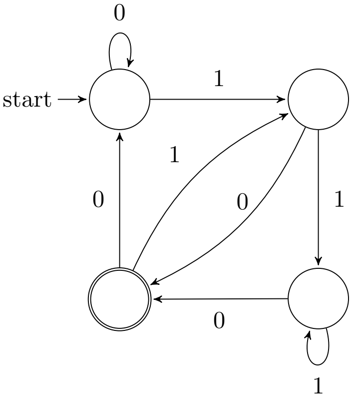
NFA Advantage

$$\{x \in \{0,1\}^* \mid x\text{'s next-to-last symbol is } 1\}$$

NFA Advantage

$$\{x \in \{0,1\}^* \mid x\text{'s next-to-last symbol is } 1\}$$

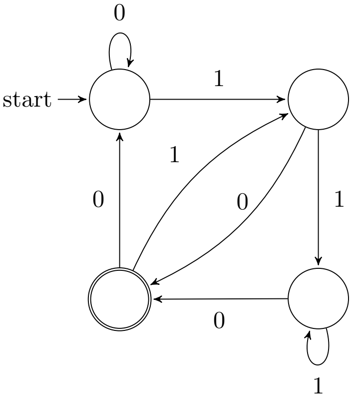
DFA



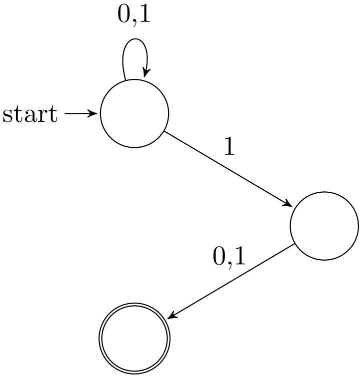
NFA Advantage

$$\{x \in \{0,1\}^* \mid x\text{'s next-to-last symbol is } 1\}$$

DFA



NFA

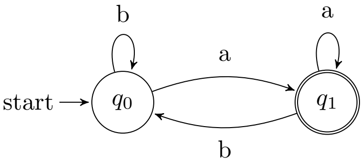


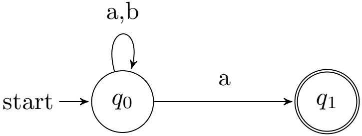
NFA Advantage Cont'd

$\{xa \mid x \in \{a, b\}^*\}$

NFA Advantage Cont'd

$\{xa \mid x \in \{a, b\}^*\}$





DFA v.s. NFA

- A DFA has exactly one transition from every state on every symbol in the alphabet.
- By relaxing this requirement we get a related but more flexible automaton: the non-deterministic finite automaton (NFA).
- DFA is a special case of NFA.

Simple NFA

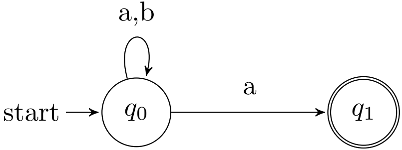


Figure: Simple NFA

Does not have exactly one transition from every state on every symbol:

- Two transitions from q_0 on symbol a
- No transitions from q_1

Possible Move Sequences

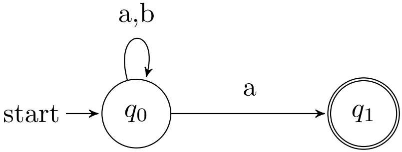


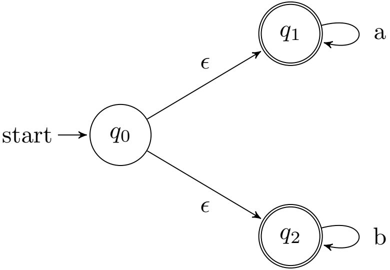
Figure: Simple NFA

Consider all the possible move sequences for the string *aa*.

- $q_0 \rightarrow q_0$ rejecting
- $q_0 \rightarrow q_1$ accepting
- $q_0 \rightarrow q_1$ stuck on the last a

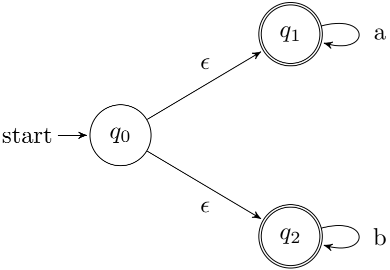
$\mathcal{M}$ accepts x (i.e., $x \in \mathcal{L}(\mathcal{M})$) if there is *at least one accepting sequence*.

ϵ-Transition



- NFA may make a state transition (spontaneously) without consuming an input symbol, or consuming an ϵ. (NFA-ϵ)
- *DFA cannot.*

Accept ϵ



Consider all the possible move sequences for ϵ, i.e., (empty string)

- q_0 no moves, ending in q_0, rejecting
- $q_0 \to q_1$ accepting
- $q_0 \to q_2$ accepting

Any state with an ϵ-transition to an accepting state ends up working like an accepting state too.

NFA(-ϵ) Formal

An NFA-ϵ $\mathcal{M}$ is a quintuple

$$\mathcal{M} = (Q, \Sigma, \delta, q_0, F)$$

- Q *finite* set of states
- Σ alphabet, i.e., *finite* set of symbols
- $\delta \in (Q \times \Sigma_\epsilon \to \mathcal{P}(Q))$ transition function
- $q_0 \in Q$ the start state
- $F \subseteq Q$ the set of accepting states

And note that

- $\epsilon \notin \Sigma$
- $\Sigma_\epsilon = \Sigma \bigcup \{\epsilon\}$

Powerset

If S is a set, the *powerset* of S is the set of all subsets of S

$$\mathcal{P}(S) = \{R \mid R \subseteq S\}$$

E.g., if $S = 1, 2, 3$, then
$\mathcal{P}(S) = \{\{\}, \{1\}, \{2\}, \{3\}, \{1, 2\}, \{1, 3\}, \{2, 3\}, \{1, 2, 3\}\}$.

NFA Example

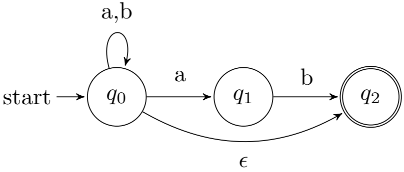


Figure: $\mathcal{L}(\mathcal{M}) = \{a, b\}^*$

$\mathcal{M} = (Q, \Sigma, \delta, q_0, F)$

- $Q = \{q_0, q_1, q_2\}$
- $\Sigma = \{a, b\}$
- $F = q_2$
- δ?

NFA v.s. NFA-ϵ

Theorem

Any NFA-ϵ can be turned into an NFA.

- ϵ-transitions are just syntax sugar.
- Both NFA-ϵ and NFA recognize *exactly* the same languages.
- NFA and NFA-ϵ may be converted to each other.

ϵ-Transition for Correct Union

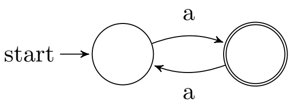


Figure: $A = \{a^n \mid n \text{ is odd}\}$

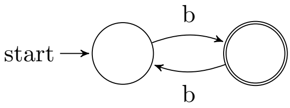


Figure: $B = \{b^n \mid n \text{ is odd}\}$

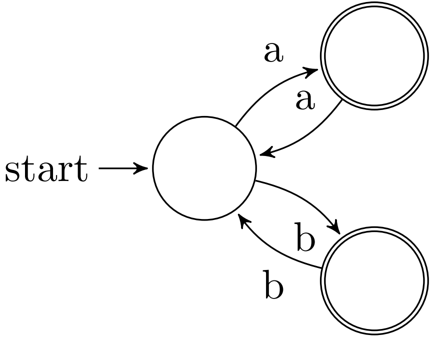


Figure: $C = A \bigcup B$???

ϵ-Transition for Correct Union

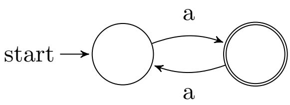


Figure: $A = \{a^n \mid n \text{ is odd}\}$

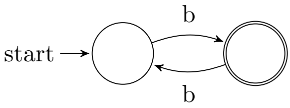


Figure: $B = \{b^n \mid n \text{ is odd}\}$

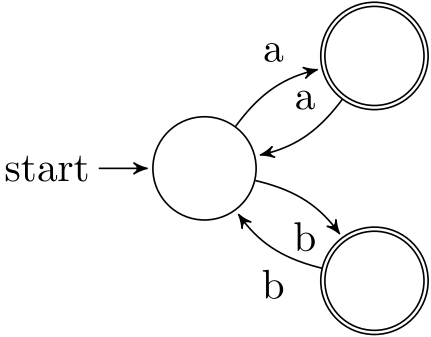


Figure: $C = A \bigcup B$???

NO, this accepts *aab*.

ϵ-Transition for Correct Union

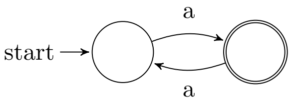


Figure: $A = \{a^n \mid n \text{ is odd}\}$

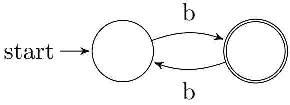


Figure: $B = \{b^n \mid n \text{ is odd}\}$

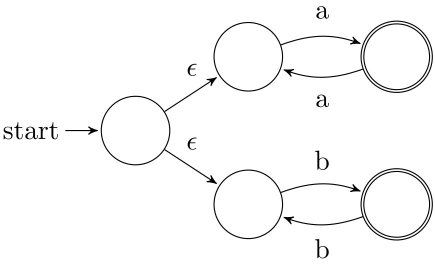


Figure: $C = A \bigcup B$

ϵ-Transition for Correct Concatenation

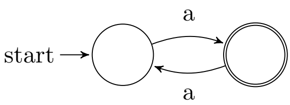


Figure: $A = \{a^n \mid n \text{ is odd}\}$

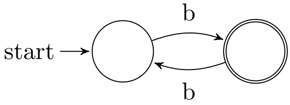


Figure: $B = \{b^n \mid n \text{ is odd}\}$

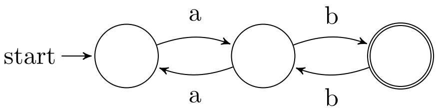


Figure: $C = \{xy | x \in A \text{ and } y \in B\}$???

ϵ-Transition for Correct Concatenation

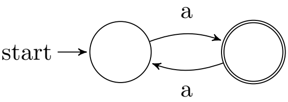


Figure: $A = \{a^n \mid n \text{ is odd}\}$

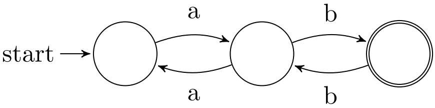


Figure: $C = \{xy | x \in A \text{ and } y \in B\}$???

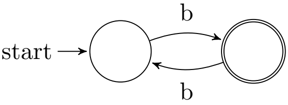


Figure: $B = \{b^n \mid n \text{ is odd}\}$

NO, this accepts *abbaab*.

ϵ-Transition for Correct Concatenation

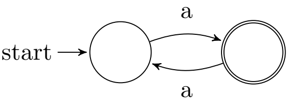


Figure: $A = \{a^n \mid n \text{ is odd}\}$

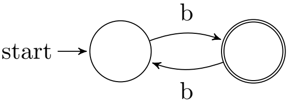


Figure: $B = \{b^n \mid n \text{ is odd}\}$

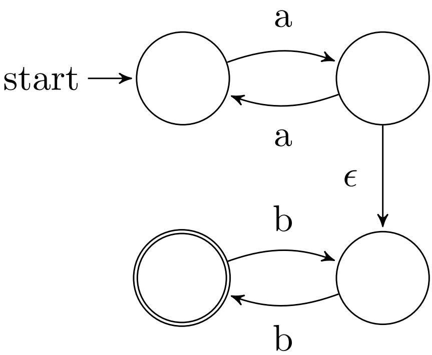


Figure: $C = \{xy | x \in A \text{ and } y \in B\}$

δ^* Function

- δ function gives 1-symbol move.
- δ^* gives a whole-string result.

$$\delta^*(q, x) = \{r \mid (q, x) \mapsto^* (r, \epsilon)\}$$

Intuitively, $\delta^*(q, x)$ is the set of all states the NFA might be in starting in state q and reading whole x.

NFA ID

Instantaneous Description (ID) is a description of a point in an NFA's execution.

Suppose I is an ID, $I = (q, x)$

- $q \in Q$ is the current state
- $x \in \Sigma^*$ is the *unread* part of the input

And given an NFA $\mathcal{M}$ and a string x,

- Initially, before $\mathcal{M}$ processes the x, we have ID (q_0, x).
- When $\mathcal{M}$ *accepts* x, it ends in an ID (f, ϵ) where $f \in F$.

Move Relation on ID

- One-Move Relation
  If $I = (q, \omega x)$ and $J = (r, x)$, $\forall x \in \Sigma^*, \forall \omega \in \Sigma$ or $\omega = \epsilon$

  $$I \mapsto J \text{ if and only if } r \in \delta(q, \omega)$$

- Zero-or-More-Move Relation
  if there is a sequence of zero or more moves that starts with I and ends with J

  $$I \mapsto^* J$$

  Note that $I \mapsto^* I$

$\mathcal{M}$ Accepts x

Theorem

A string $x \in \Sigma^*$ is accepted by an NFA $\mathcal{M} = (Q, \Sigma, \delta, q_0, F)$ if and only if $\delta^*(q_0, x) \cap F \neq \emptyset$.

Language Defined by NFA

For any NFA $\mathcal{M} = (Q, \Sigma, \delta, q_0, F)$, $\mathcal{L}(\mathcal{M})$ denotes the language accepted by $\mathcal{M}$

$$\mathcal{L}(\mathcal{M}) = \{x \in \Sigma^* \mid \delta^*(q_0, x) \cap F \neq \emptyset\}$$

Outline

Introduction

Man Wolf Goat Cabbage

Deterministic Finite Automata

Non-deterministic Finite Automata

Summary

DFA Summary

- Good Faster and simpler
- Bad
  - There are languages for which DFA-based implementation takes exponentially more space than NFA-based.
  - Harder to extend for non-regular constructs
- Example: scanner in compiler
  - Speed is critical
  - Token languages do not usually bring out the exponential-size pathology of DFAs

NFA Summary

- Good
  - Easier to extend for non-regular language constructs
  - No exponential-space pathology
- Bad slower and trigger
- Example: regular expression features in `Python`, `Perl`, etc.
  Need to handle non-regular constructs as well as regular ones.

D/N-FA Accept Regular Languages

$\mathcal{L}_{DFA}$ All languages accepted by DFAs

$\mathcal{L}_{NFA}$ All languages accepted by NFAs

$\mathcal{L}_{R}$ Regular Languages

$$\mathcal{L}_{DFA} = \mathcal{L}_{NFA} = \mathcal{L}_{R}$$

Advanced Topics

- Conversion between DFA and NFA
- ϵ-transition elimination
- Equivalence prove
- Implementation