Week 8 - DFAs, NFAs & Regular Languages

Billy Price

LANGUAGES

Alphabets $\Sigma = \ldots$ $\{0,1\}$ $\{a,b,c\}$ $\{a,b,c,\ldots,x,y,z\}$

Strings A finite sequence of symbols concatenated together 0100 abaa word
also called words

This includes the empty string, denoted ε (epsilon). Think ""

Kleene Star Given an alphabet, Σ, the Kleene star of Σ, written Σ^* is the set of all strings (remember, finite), using symbols in Σ only.

union over all $n \in \mathbb{N}$

$$\Sigma^* := \bigcup_{n \in \mathbb{N}} \{a_1 a_2 \ldots a_n \mid \forall 1 \leq i \leq n \; (a_i \in \Sigma)\}$$

the set is infinite, but not any of its strings

Example $\{0,1\}^* := \{\varepsilon, 0, 1, 00, 01, 10, 11, \ldots, 10110001, \ldots\}$

Languages Given an alphabet Σ, a language, L, is a subset of Σ^*

Examples $\emptyset$, Σ^*, $\{\varepsilon\}$, $\{010\}$, ab^*a, $\{ww \mid w \in \Sigma^*\}$

$\{F \mid F \text{ is a valid Haskell expression}\}$

$\{F \mid F \text{ is a valid Haskell expression which executes in finite time}\}$

$\{\langle P \rangle \mid P \text{ is a travelling salesman problem whose solution length is a prime number}\}$

Note these vary in their computability, which is not about how large the language is, but difficult it is to decide whether or not a string belongs to the language.

This isn't about algorithmic complexity, rather, how sophisticated must our computation model be, to be able to answer correctly in finite time?

Class of Languages We will distinguish between primitive and sophisticated languages by dividing them into classes (sets)

Regular ⊂ Context Free ⊂ Context Sensitive ⊂ Turing Recognisable ⊂ Decidable

Closure Given any operation on languages a class of languages is closed under <that operation> if, applying it to languages in that class can never produce a language outside the class.

Recognition We will define several models of computation, which are able to accept and sometimes reject, strings over some alphabet. We say that such a machine recognises a language if it accepts all strings in the language, and does not accept others.

REGULAR LANGUAGES

Billy Price

Definition A language, L, is regular, iff it can be recognised by some DFA

Examples $\emptyset$, Σ^*, $\{\varepsilon\}$, $\{010\}$, $\{a\}^* \cup \{b\}$ + any finite language

Regular Operations Let A and B, be languages. The regular operations are

Union: $A \cup B$ Concatenation: $A \circ B := \{xy \mid x \in A, y \in B\}$ Kleene Star: $A^* := \bigcup_{n \in \mathbb{N}} \{w_1 \dots w_n \mid \forall i \leq n,\ w_i \in A\}$ (concatenate the strings)

Question Is the class of regular languages closed under the regular operations?

DFAs (Deterministic Finite-state Automata)

Formal Definition

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

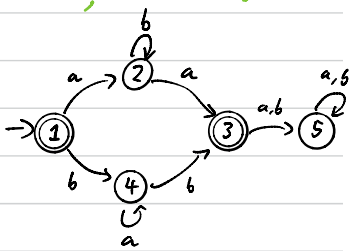
- Q is a finite set of states,
- Σ is a finite alphabet,
- $\delta : Q \times \Sigma \to Q$ is the transition function,
- $q_0 \in Q$ is the start state, and
- $F \subseteq Q$ are the accept states.

Example

$(Q, \Sigma, \delta, q_0, F)$ $\quad \delta((1,a)) = 2$

$Q = \{1,2,3,4,5\}$, $\Sigma = \{a,b\}$, $q_0 = 1$, $F = \{1,3\}$

$$\delta = \{((1,a),2), ((1,b),4), ((2,a),3), ((2,b),2), ((3,a),5), ((3,b),5), ((4,a),4), ((4,b),3), ((5,a),5), ((5,b),5)\}$$



DFA acceptance definition

Let $M = (Q, \Sigma, \delta, q_0, F)$ and let $w = v_1 v_2 \cdots v_n$ be a string from Σ^*.

M accepts w iff there is a sequence of states $r_0, r_1, \dots, r_n$, with each $r_i \in Q$, such that

1. $r_0 = q_0$
2. $\delta(r_i, v_{i+1}) = r_{i+1}$ for $i = 0, \dots, n-1$
3. $r_n \in F$

M recognises language A iff $A = \{w \mid M \text{ accepts } w\}$.

Informally

1. Begin in the start state
2. Read the next symbol in the string, and follow the corresponding transition to the next state
3. Repeat 2 until no symbols remain
4. If the current state is an accept state, accept.

Example Acceptance The example DFA above accepts aba because the sequence of states $1, 2, 2, 3$ satisfies $1 = q_0$, $3 \in F$ and $2 = \delta(1,a)$ $2 = \delta(2,b)$ $3 = \delta(2,a)$. We could package this information like this: $q_0 = 1 \xrightarrow{a} 2 \xrightarrow{b} 2 \xrightarrow{a} 3 \in F$

Rejection (for DFAs)

Of course, there is only one possible sequence of states that could satisfy this, and if the final state in this sequence is not in F, we can say the DFA rejects that string.

Warning In general (for other automata) rejection is not the same as non-acceptance.

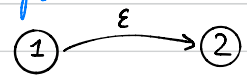
NFAs

Billy Price

Nondeterminism Being able to choose a path when faced with multiple options in executing an algorithm

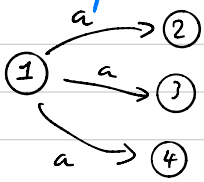
In a Non-deterministic Finite-State Automata (NFA), the non-determinism appears via

Epsilon Transitions



Can transition without consuming any input

Multiple Choice Transitions



Can read "a" and transition to either 2, 3, or 4.
Represented like this
$\delta(1,a) = \{2,3,4\}$

Formal definition

For any alphabet Σ let Σ_ϵ denote $\Sigma \cup \{\epsilon\}$.

An NFA is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where

- Q is a finite set of states,
- Σ is a finite alphabet,
- $\delta : Q \times \Sigma_\epsilon \to \mathcal{P}(Q)$ is the transition function,
- $q_0 \in Q$ is the start state, and
- $F \subseteq Q$ are the accept states.

Difference between DFAs & NFAs

DFA $\quad \delta : Q \times \Sigma \longrightarrow Q$
$(q, a) \longmapsto q'$

NFA $\quad \delta : Q \times \Sigma_\varepsilon \longrightarrow P(Q)$
$(q, a) \longmapsto$ <some subset of Q>
↑ either $a \in \Sigma$ or $a = \varepsilon$

NFA acceptance

The definition of what it means for an NFA N to accept a string says that it has to be possible to make the necessary transitions.

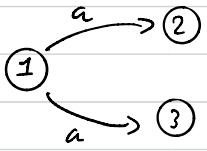
Let $N = (Q, \Sigma, \delta, q_0, F)$ be an NFA and let $w = v_1 v_2 \cdots v_n$ where each v_i is a member of Σ_ϵ.

N accepts w iff there is a sequence of states $r_0, r_1, \ldots, r_n$, with each $r_i \in Q$, such that

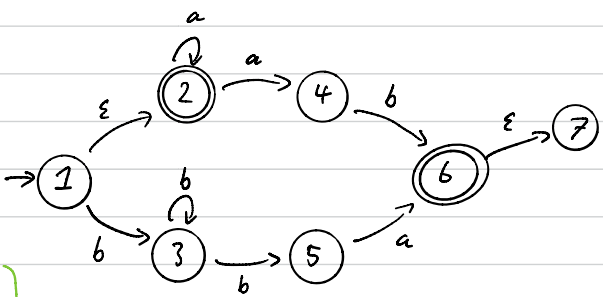
1. $r_0 = q_0$
2. $r_{i+1} \in \delta(r_i, v_{i+1})$ for $i = 0, \ldots, n-1$
3. $r_n \in F$

N recognises language A iff $A = \{w \mid N \text{ accepts } w\}$.

Note that in this situation, we cannot say $2 = \delta(1,a)$
we must say $2 \in \delta(1,a)$
It's also wrong to say $\{2\} = \delta(1,a)$



Example $\Sigma = \{a, b\}$, $q_0 = 1$, $F = \{2, 6\}$



$$\delta = \begin{cases} ((1,\varepsilon),\{2\}), & ((1,a),\phi), & ((1,b),\{3\}), \\ ((2,\varepsilon),\phi), & ((2,a),\{2,4\}), & ((2,b),\phi), \\ ((3,\varepsilon),\phi), & ((3,a),\phi), & ((3,b),\{3,5\}), \\ ((4,\varepsilon),\phi), & ((4,a),\phi), & ((4,b),\{6\}), \\ ((5,\varepsilon),\phi), & ((5,a),\{6\}), & ((5,b),\phi), \\ ((6,\varepsilon),\{7\}), & ((6,a),\phi), & ((6,b),\phi), \\ ((7,\varepsilon),\phi), & ((7,a),\phi), & ((7,b),\phi) \end{cases}$$

Example acceptance

This NFA accepts aab, because $aab = \varepsilon aab$, and the sequence of states 1, 2, 2, 4, 6 satisfies $1 = q_0$, $6 \in F$ and $2 \in \delta(1,\varepsilon)$, $2 \in \delta(2,a)$, $4 \in \delta(2,a)$, $6 \in \delta(4,b)$

This could also be written $q_0 = 1 \xrightarrow{\varepsilon} 2 \xrightarrow{a} 2 \xrightarrow{a} 4 \xrightarrow{b} 6 \in F$

CONSTRUCTIONS ON DFAs / NFAs

Billy Price

Language of an Automaton

Given an Automaton, D, we define the language recognised by D as $L(D) := \{w \mid D \text{ accepts } w\}$, and if $L(D_2) = L(D_2)$, we say D_2 & D_2 are equivalent

Theorem

For all NFAs, N, there exists a DFA, D, such that $L(D) = L(N)$

This is how we do it =)

Each DFA is a set of states we could be right now in the NFA - remember, epsilon transitions cost nothing.

The Subset Construction

Let $N = (Q, \Sigma, \delta, q_0, F)$. Let $\xrightarrow{*}_\epsilon$ be the reflexive transitive closure of $\rightarrow_\epsilon$, which in turn is defined by $s \rightarrow_\epsilon s'$ iff $s' \in \delta(s, \epsilon)$.

Let $E(S)$ be the "ϵ closure" of $S \subseteq Q$, that is, S together with all states reachable from states in S, using only ϵ steps:

$$E(S) = \bigcup_{s \in S} \{s' \in Q \mid s \xrightarrow{*}_\epsilon s'\}$$

We construct $M = (\mathcal{P}(Q), \Sigma, \delta', q_0', F')$ as follows:

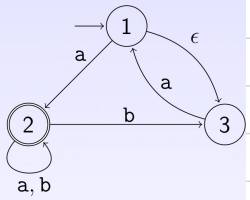
- $q_0' = E(\{q_0\})$.
- $\delta'(S, v) = \bigcup_{s \in S} E(\delta(s, v))$.
- $F' = \{S \subseteq Q \mid S \cap F \neq \emptyset\}$.

Example

Consider the NFA on the right. We can systematically construct an equivalent DFA from the NFA.

The DFA's start state is $\{1, 3\}$.
From $\{1, 3\}$ a takes us to $\{1, 2, 3\}$.
From $\{1, 2, 3\}$, a takes us back to $\{1, 2, 3\}$, b takes us to $\{2, 3\}$.

Any state S which contains an accept state from the NFA will be an accept state for the DFA. Here we mark accept states with a star.

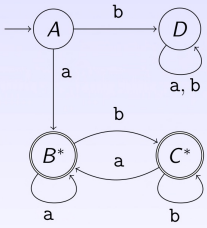


	a	b
A={1,3}	B*	–
B*={1,2,3}	B*	C*
C*={2,3}	B*	C*

Here is the equivalent DFA that we derive.

Any state S which contains an accept state from the NFA (in this case the NFA has just one, namely state 2) becomes an accept state for the DFA.

We add (dead) state D that corresponds to the empty set.



	a	b
A={1,3}	B*	D
B*={1,2,3}	B*	C*
C*={2,3}	B*	C*
D=∅	D	D

Minimizing a DFA/NFA

The following algorithm takes an NFA and produces an equivalent minimal DFA. Of course the input can also be a DFA.

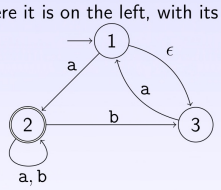
1. Reverse the NFA;
2. Determinize the result;
3. Reverse again;
4. Determinize.

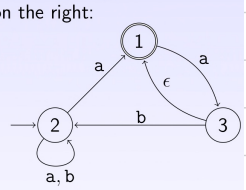
To reverse an NFA A with initial state q_0 and accept states $F \neq \emptyset$:

1. If $F = \{q\}$, let q be the accept state.
2. Otherwise add a new state q which becomes the only accept state; then, for each state in F, add an ϵ transition to q.
3. Reverse every transition in the resulting NFA, making q the initial state and q_0 the (only) accept state.

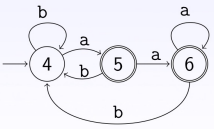
Example Minimization

Consider the NFA that we determinized two slides ago.
Here it is on the left, with its reversal on the right:

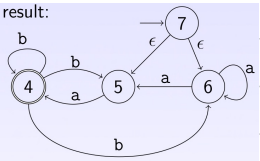




Making the reversed NFA deterministic.
We renamed the states to avoid later confusion;
4 corresponds to $\{2\}$, 5 to $\{1, 2\}$, and 6 to $\{1, 2, 3\}$.



Now reversing the result:



And finally making the result deterministic: