COMP1600, week 8: Non-Deterministic Finite Automata (NFAs) and Regular Expressions

convenors: Dirk Pattinson, Pascal Bercher

lecturer: Pascal Bercher

slides based on those by: Dirk Pattinson
(with contributions by Victor Rivera and previous colleagues)

Semester 2, 2024

Australian National University

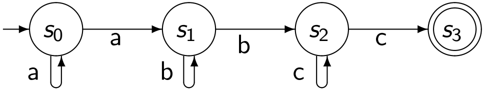
Overview of Week 8

- Introduction
- Non-Deterministic Finite Automata (NFAs) — Formally —
- Language of an NFA
- Determinisation of NFAs
- NFAs with ϵ-transitions
- Regular Expressions

Introduction

Non-Deterministic Finite State Automata — NFAs

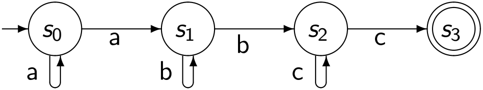
Consider this FSA:



Q. Is it a DFA in the sense of our definition?

Non-Deterministic Finite State Automata — NFAs

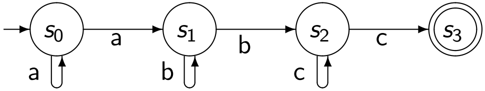
Consider this FSA:



Q. Is it a DFA in the sense of our definition?

Q. Is it intuitively clear what it does? Test it! What's it's language?

Is it legal, i.e., a "proper" DFA?

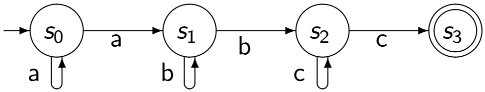


A. It makes sense, but it is *nondeterministic*: A nondeterministic finite automaton (NFA). So not a "legal" DFA, but a specimen of a different breed.

Differences to deterministic automata

- Multiple edges with the **same label** come out of states
- For some states, there is **not an edge** for every token

Is it legal, i.e., a “proper” DFA?



A. It makes sense, but it is *nondeterministic*: A nondeterministic finite automaton (NFA). So not a “legal” DFA, but a specimen of a different breed.

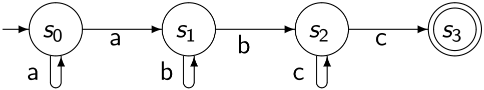
Differences to deterministic automata

- Multiple edges with the **same label** come out of states
- For some states, there is **not an edge** for every token

Formally. NFAs have a transition *relation* rather than a transition *function*.

- transition relation $R(s_1, x, s_2)$ is true if there's an x-labelled edge from s_1 to s_2
- there can be *many* states that are connected to s_1 via an x-labelled edge. (Example: s_0, s_1, s_2)
- there can be *no* x-labelled edge between s_1 and *any* state. (Example: s_3)

Is it clear what it does?



Observations.

- Some states don't have an outgoing edge with a certain letter, so the NFA can "get stuck".
- In some states, there's more than one possible successor state with a certain letter.

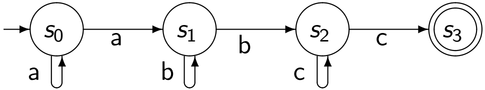
Acceptance condition for NFAs given string w:

- can get from initial to final state, making the "right" choice of successor state without getting stuck

Example. $w = aaabcc$

- need to "look ahead" to make the right choice
- (alternatively, try to backtrack if wrong choice has been made)

Is it clear what it does?



Observations.

- Some states don't have an outgoing edge with a certain letter, so the NFA can "get stuck".
- In some states, there's more than one possible successor state with a certain letter.

Acceptance condition for NFAs given string w:

- can get from initial to final state, making the "right" choice of successor state without getting stuck

Non-Example. $w = aaacc$

- Doesn't work because we are (definitely) stuck after reading the last a.

Key Differences: DFAs *vs* NFAs

DFA:

- DFAs have a *transition function*.

NFA:

- NFAs have a *transition* ***relation***.

Key Differences: DFAs *vs* NFAs

DFA:

- DFAs have a *transition function*.
- For each state in a DFA and for each input symbol, there is a **unique** successor state.

NFA:

- NFAs have a *transition* ***relation***.
- NFAs allow **zero, one, or more** transitions from a state for the same input symbol.

Key Differences: DFAs *vs* NFAs

DFA:

- DFAs have a *transition function*.
- For each state in a DFA and for each input symbol, there is a **unique** successor state.
- An input sequence $x_1, x_2, \ldots, x_n$ is *accepted* by a DFA if there *exists* some sequence of transitions that leads from the initial state to a final state.

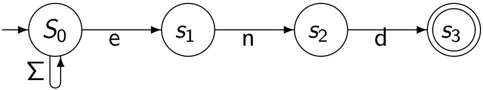
NFA:

- NFAs have a *transition* ***relation***.
- NFAs allow **zero, one, or more** transitions from a state for the same input symbol.
- An input sequence $x_1, x_2, \ldots, x_n$ is *accepted* by a NFA if there *exists* some sequence of transitions that leads from the initial state to a final state.

Q. Is there actually a difference between the solution criteria?

Example: NFA vs. DFA

$L = \{\alpha end \mid \alpha \in \Sigma^*\}$ An NFA recognising strings of letters ending in "end":
(The alphabet Σ here is the Latin alphabet.)



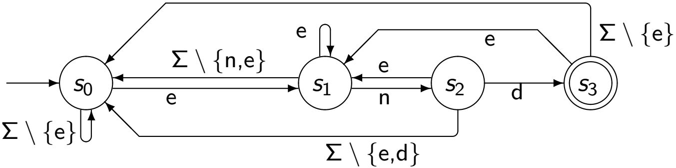
Note.

- *two* transitions from s_0 for the letter "e"
- *no* transition from s_1 for (e.g.) the letter "d"

An Equivalent DFA

Example. DFAs are (often) more complex.

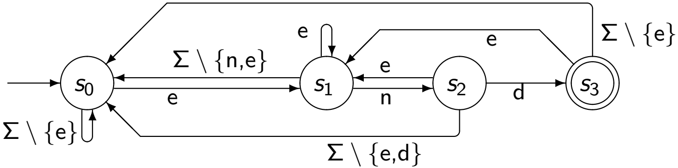
A DFA that recognises strings of letters than end in “end”.



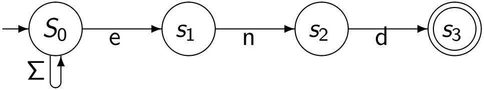
An Equivalent DFA

Example. DFAs are (often) more complex.

A DFA that recognises strings of letters than end in “end”.



Q. Which FSA is easier to write and read?



Why do we need/use/have NFAs?

So, why do we have NFAs?

- They are more compact.
- They are (sometimes!) easier to read and write.

Q1. Why only sometimes?

Why do we need/use/have NFAs?

So, why do we have NFAs?

- They are more compact.
- They are (sometimes!) easier to read and write.

Q1. Why only sometimes?

Q2. Can you think of another reason why “NFAs exist”?

Why do we need/use/have NFAs?

So, why do we have NFAs?

- They are more compact.
- They are (sometimes!) easier to read and write.
- Because we are step-wise increasing the power of our models of computation! (this week: add non-determinism.)

Q1. Why only sometimes?

Q2. Can you think of another reason why "NFAs exist"?

Non-Deterministic Finite Automata (NFAs) — Formally —

NFAs: Formal Definition

A Nondeterministic Finite State Automaton (NFA) consists of five parts:

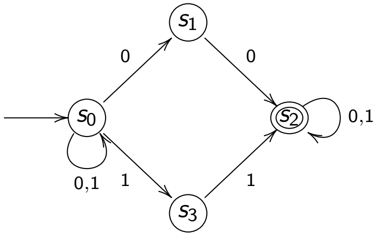
$$A = (\Sigma, S, s_0, F, R)$$

- a finite input **alphabet** Σ, the (finite) set of tokens
- a finite set of **states** S
- an **initial** state $s_0 \in S$ (we start here)
- a set of **final** states $F \subseteq S$ (we hope to finish in one of these)
- a **transition relation** $R \subseteq S \times \Sigma \times S$.

Aside. The transition *relation* is what makes the automaton nondeterministic. It can be seen as a function $\delta : S \times \Sigma \to \mathcal{P}(S)$, where $\mathcal{P}(S)$ is the set of subsets (i.e., power set) of S. (Cf. slide 18 of last week!)

Another Example

Transition Diagram



As a transition table.

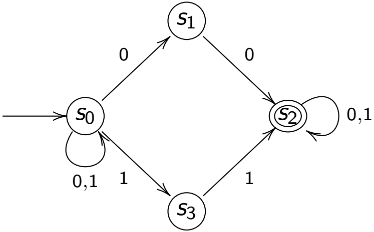
	0	1
$\to s_0$	$\{s_0, s_1\}$	$\{S_0, s_3\}$
s_1	$\{s_2\}$	$\emptyset$
$\odot s_2$	$\{s_2\}$	$\{s_2\}$
s_3	$\emptyset$	$\{s_2\}$

Both convey precisely the same information.

Q. What is the language of this automaton?

Another Example

Transition Diagram



As a transition table.

	0	1
$\to s_0$	$\{s_0, s_1\}$	$\{S_0, s_3\}$
s_1	$\{s_2\}$	$\emptyset$
$\odot s_2$	$\{s_2\}$	$\{s_2\}$
s_3	$\emptyset$	$\{s_2\}$

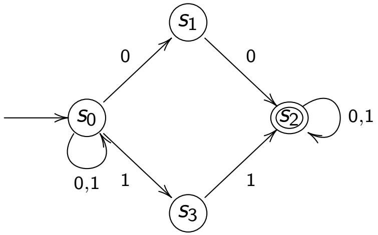
Both convey precisely the same information.

Q. What is the language of this automaton?

A. Informally: Any string that contains at least two consecutive 0s or 1s.

Another Example

Transition Diagram



As a transition table.

	0	1
$\rightarrow s_0$	$\{s_0, s_1\}$	$\{S_0, s_3\}$
s_1	$\{s_2\}$	$\emptyset$
$\odot s_2$	$\{s_2\}$	$\{s_2\}$
s_3	$\emptyset$	$\{s_2\}$

Both convey precisely the same information.

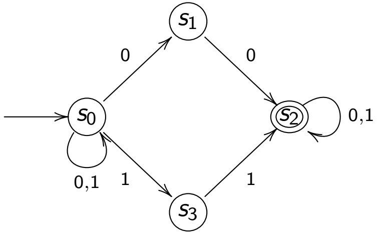
Q. What is the language of this automaton?

A. Informally: Any string that contains at least two consecutive 0s or 1s.

- Formally: $L = \{\alpha x x \beta \mid \alpha, \beta \in \Sigma^* \text{ and } x \in \Sigma\}$ (with $\Sigma = \{0, 1\}$)

Another Example

Transition Diagram



As a transition table.

	0	1
$\rightarrow s_0$	$\{s_0, s_1\}$	$\{S_0, s_3\}$
s_1	$\{s_2\}$	$\emptyset$
$\odot s_2$	$\{s_2\}$	$\{s_2\}$
s_3	$\emptyset$	$\{s_2\}$

Both convey precisely the same information.

Q. What is the language of this automaton?

A. Informally: Any string that contains at least two consecutive 0s or 1s.

- Formally: $L = \{\alpha x x \beta \mid \alpha, \beta \in \Sigma^* \text{ and } x \in \Sigma\}$ (with $\Sigma = \{0, 1\}$) or
  $L = (0 \mid 1)^*(00|11)(0 \mid 1)^*$ (this is a regular expression!)

Acceptance for NFAs

Acceptance Informally. An NFA $A = (\Sigma, S, F, s_0, R)$ *accepts* a word $w = x_1 x_2 \dots x_n$ (in symbols: $w \in L(A)$) iff there exists a sequence of states

$$s_0 \xrightarrow{x_1} s_1 \xrightarrow{x_2} \dots \xrightarrow{x_{n-1}} s_{n-1} \xrightarrow{x_n} s_n$$

where s_0 is the starting state, $s_n \in F$ is an accepting state, and $s_i \xrightarrow{x} s_j$ if $(s_i, x, s_j) \in R$.

Acceptance for NFAs

Acceptance Informally. An NFA $A = (\Sigma, S, F, s_0, R)$ *accepts* a word $w = x_1 x_2 \ldots x_n$ (in symbols: $w \in L(A)$) iff there exists a sequence of states

$$s_0 \xrightarrow{x_1} s_1 \xrightarrow{x_2} \ldots \xrightarrow{x_{n-1}} s_{n-1} \xrightarrow{x_n} s_n$$

where s_0 is the starting state, $s_n \in F$ is an accepting state, and $s_i \xrightarrow{x} s_j$ if $(s_i, x, s_j) \in R$.

Aside. This is like for deterministic automata, the only difference is that for

- *deterministic automata* we have $s_i \xrightarrow{x} s_j$ if $N(s_i, x) = s_j$
  (that is, the automaton makes the (unique) transition)
- *non-deterministic automata* we have $s_i \xrightarrow{x} s_j$ if $(s_i, x, s_j) \in R$
  (that is, the automaton can make a transition)

Eventual State Relation for NFAs

Basic Idea. The eventual state relation $R^*(s, w, s')$ is true if s' is a state that the NFA ***can*** reach, starting in state s and reading string w.

Eventual State Relation for NFAs

Basic Idea. The eventual state relation $R^*(s, w, s')$ is true if s' is a state that the NFA ***can*** reach, starting in state s and reading string w.

Formal Definition. The eventual state relation has type

$$R^* \subseteq S \times \Sigma^* \times S$$

$$(\text{equivalent to } R^* : S \times \Sigma^* \times S \to Bool)$$

Eventual State Relation for NFAs

Basic Idea. The eventual state relation $R^*(s, w, s')$ is true if s' is a state that the NFA ***can*** reach, starting in state s and reading string w.

Formal Definition. The eventual state relation has type

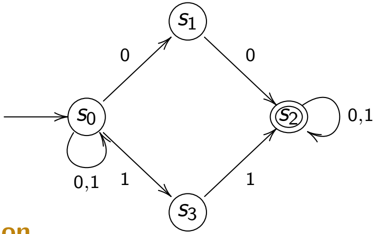
$$R^* \subseteq S \times \Sigma^* \times S$$
$$(\text{equivalent to } R^* : S \times \Sigma^* \times S \to \mathit{Bool})$$

and is defined inductively as follows:

$$R^*(s, \epsilon, s) \text{ (is true)}$$
$$R^*(s, x\alpha, s') = \exists s''. R(s, x, s'') \wedge R^*(s'', \alpha, s')$$

Eventual State Relation: Example

The "double digits" automaton *DD*:

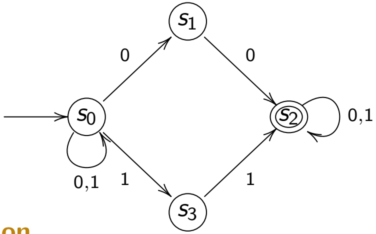


Eventual State Relation.

- $(s_0, \epsilon, s_0) \in R^*$ by definition

Eventual State Relation: Example

The "double digits" automaton *DD*:

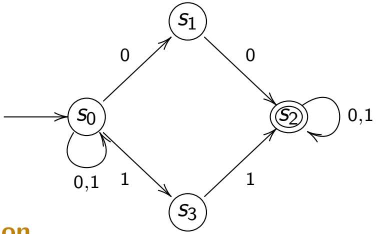


Eventual State Relation.

- $(s_0, \epsilon, s_0) \in R^*$ by definition
- $s_0 \xrightarrow{0} s_0 \xrightarrow{0} s_0 \xrightarrow{1} s_0$, hence $(s_0, 001, s_0) \in R^*$.

Eventual State Relation: Example

The "double digits" automaton *DD*:

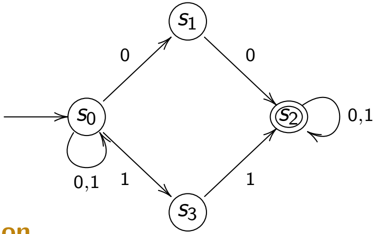


Eventual State Relation.

- $(s_0, \epsilon, s_0) \in R^*$ by definition
- $s_0 \xrightarrow{0} s_0 \xrightarrow{0} s_0 \xrightarrow{1} s_0$, hence $(s_0, 001, s_0) \in R^*$.
- $s_0 \xrightarrow{0} s_0 \xrightarrow{0} s_0 \xrightarrow{1} s_3$, hence $(s_0, 001, s_3) \in R^*$.

Eventual State Relation: Example

The "double digits" automaton *DD*:

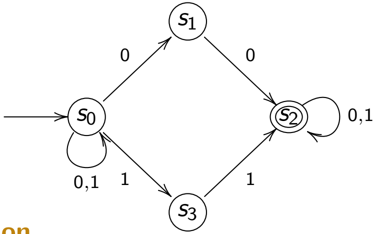


Eventual State Relation.

- $(s_0, \epsilon, s_0) \in R^*$ by definition
- $s_0 \xrightarrow{0} s_0 \xrightarrow{0} s_0 \xrightarrow{1} s_0$, hence $(s_0, 001, s_0) \in R^*$.
- $s_0 \xrightarrow{0} s_0 \xrightarrow{0} s_0 \xrightarrow{1} s_3$, hence $(s_0, 001, s_3) \in R^*$.
- $s_0 \xrightarrow{0} s_1 \xrightarrow{0} s_2 \xrightarrow{1} s_2$, hence $(s_0, 001, s_2) \in R^*$.

Eventual State Relation: Example

The "double digits" automaton *DD*:



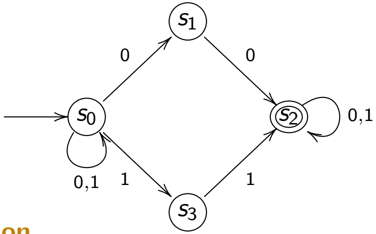
Eventual State Relation.

- $(s_0, \epsilon, s_0) \in R^*$ by definition
- $s_0 \xrightarrow{0} s_0 \xrightarrow{0} s_0 \xrightarrow{1} s_0$, hence $(s_0, 001, s_0) \in R^*$.
- $s_0 \xrightarrow{0} s_0 \xrightarrow{0} s_0 \xrightarrow{1} s_3$, hence $(s_0, 001, s_3) \in R^*$.
- $s_0 \xrightarrow{0} s_1 \xrightarrow{0} s_2 \xrightarrow{1} s_2$, hence $(s_0, 001, s_2) \in R^*$.

Q1. What about $s_0 \xrightarrow{0} s_0 \xrightarrow{0} s_1 \xrightarrow{1} \lightning$? So, does $001 \in L(DD)$ hold?

Eventual State Relation: Example

The "double digits" automaton *DD*:



Eventual State Relation.

- $(s_0, \epsilon, s_0) \in R^*$ by definition
- $s_0 \xrightarrow{0} s_0 \xrightarrow{0} s_0 \xrightarrow{1} s_0$, hence $(s_0, 001, s_0) \in R^*$.
- $s_0 \xrightarrow{0} s_0 \xrightarrow{0} s_0 \xrightarrow{1} s_3$, hence $(s_0, 001, s_3) \in R^*$.
- $s_0 \xrightarrow{0} s_1 \xrightarrow{0} s_2 \xrightarrow{1} s_2$, hence $(s_0, 001, s_2) \in R^*$.

Q1. What about $s_0 \xrightarrow{0} s_0 \xrightarrow{0} s_1 \xrightarrow{1} \lightning$? So, does $001 \in L(DD)$ hold?

Q2. Does $0110 \in L(DD)$ hold?

An Important (but Unsurprising) Theorem about R^*

For all states s, s' and for all strings $\alpha, \beta \in \Sigma^*$

$$R^*(s, \alpha\beta, s') \text{ if and only if } \exists s''.\ R^*(s, \alpha, s'') \land R^*(s'', \beta, s')$$

The proof is similar to the corresponding result for N^* in DFAs.
(You could do it as an exercise!)

Language of an NFA

Language of an NFA, revisited

Let $A = (\Sigma, S, s_0, F, R)$ be a NFA.

Acceptance, formally. A string w is *accepted* by A if

$$\exists s \in F.\ R^*(s_0, w, s)$$

(Compare with the definition of acceptance for NFAs earlier)

Language of an NFA, revisited

Let $A = (\Sigma, S, s_0, F, R)$ be a NFA.

Acceptance, formally. A string w is *accepted* by A if

$$\exists s \in F.\ R^*(s_0, w, s)$$

(Compare with the definition of acceptance for NFAs earlier)

Language of an NFA.
The *language* accepted by A is the set of all strings accepted by A

$$L(A) = \{w \in \Sigma^* \mid \exists s \in F.\ R^*(s_0, w, s)\}$$

Informally. That is, $w \in L(A)$ iff *there exists* a path through the diagram for A, from s_0 to a final state s ($s \in F$), such that the symbols on the path match the symbols in w

Language of an NFA, Comment

Some comments (related to languages):

- Identifying the language of an NFA is not always easy!
- ... and neither is constructing an NFA given a language.
- We recommend practising:
  - Take some language and draw the NFA.
  - Take some NFA and identify its language.

Careful:

Q. Can *every* language be recognised* by an NFA?
(*Recall that "recognising" is a synonym for "accepting".)

On the Power of Non-Determinism!

Q. Is there a language that is accepted by an NFA for which we *cannot* find a DFA that (also) accepts it?

- it seems easier to construct NFAs
- but in examples, DFAs did also exist

On the Power of Non-Determinism!

Q. Is there a language that is accepted by an NFA for which we *cannot* find a DFA that (also) accepts it?

- it seems easier to construct NFAs
- but in examples, DFAs did also exist

A. No.

Theorem.
If language L is accepted by a NFA, then there is some DFA which accepts the same language. Or more formally:
Let A be an NFA. Then, there exists a DFA A', such that $L(A) = L(A')$.

On the Power of Non-Determinism!

Q. Is there a language that is accepted by an NFA for which we *cannot* find a DFA that (also) accepts it?

- it seems easier to construct NFAs
- but in examples, DFAs did also exist

A. No.

Theorem.
If language L is accepted by a NFA, then there is some DFA which accepts the same language. Or more formally:
Let A be an NFA. Then, there exists a DFA A', such that $L(A) = L(A')$.

Proof.
We provide an algorithm that, given an arbitrary NFA A, creates a DFA A', such that $L(A) = L(A')$. (In the worst-case, it might take exponential time.)

Determinisation of NFAs

NFA to DFA Construction

Assumption. We have an NFA with state set $\{q_0, \dots, q_n\}$.

Basic Idea.

- consider all possible runs of the NFA in parallel
- as a consequence, can be in a *set* of states

NFA to DFA Construction

Assumption. We have an NFA with state set $\{q_0, \ldots, q_n\}$.

Basic Idea.

- consider all possible runs of the NFA in parallel
- as a consequence, can be in a *set* of states

Construction.

- A *state* of the DFA is a *set of states* of the NFA:
  - E.g., the DFA state $\{q_3, q_7\}$ corresponds to being in q_3 or q_7 in the NFA.
  - Signifies the states that the NFA *can* be in after reading some input.

NFA to DFA Construction

Assumption. We have an NFA with state set $\{q_0, \ldots, q_n\}$.

Basic Idea.

- consider all possible runs of the NFA in parallel
- as a consequence, can be in a *set* of states

Construction.

- A *state* of the DFA is a *set of states* of the NFA:
  - E.g., the DFA state $\{q_3, q_7\}$ corresponds to being in q_3 or q_7 in the NFA.
  - Signifies the states that the NFA *can* be in after reading some input.
- Transition function: records possible next states.
  - E.g., from DFA state $\{q_3, q_7\}$ (=NFA states q_3 and q_7) when reading letter x, successor state equals the union of transitions (with x) from q_3 and q_7.

NFA to DFA Construction

Assumption. We have an NFA with state set $\{q_0, \dots, q_n\}$.

Basic Idea.

- consider all possible runs of the NFA in parallel
- as a consequence, can be in a *set* of states

Construction.

- A *state* of the DFA is a *set of states* of the NFA:
  - E.g., the DFA state $\{q_3, q_7\}$ corresponds to being in q_3 or q_7 in the NFA.
  - Signifies the states that the NFA *can* be in after reading some input.
- Transition function: records possible next states.
  - E.g., from DFA state $\{q_3, q_7\}$ (=NFA states q_3 and q_7) when reading letter x, successor state equals the union of transitions (with x) from q_3 and q_7.
- DFA *final states* are state sets that *contain* a final NFA state.

Subset Construction: The Finer Points

Input. Let NFA $A = (\Sigma, S, s_0, F, R)$.

Subset Construction.

- DFA states are *subsets* of S but each subset plays the role of a single state!

Subset Construction: The Finer Points

Input. Let NFA $A = (\Sigma, S, s_0, F, R)$.

Subset Construction.

- DFA states are *subsets* of S but each subset plays the role of a single state!
- Transitions: for a DFA state in $Q \subseteq S$ and a letter $x \in \Sigma$:

$$\begin{aligned} N(Q,x) &= \{s_1 \in S \mid s \xrightarrow{x} s_1 \text{ for some } s \in Q\} \\ &= \{s_1 \in S \mid (s,x,s_1) \in R \text{ for some } s \in Q\} \end{aligned}$$

Example.

- Let $\{q_3, q_7\} \subseteq S$ be a DFA state (i.e., q_3 and q_7 are NFA states).

Subset Construction: The Finer Points

Input. Let NFA $A = (\Sigma, S, s_0, F, R)$.

Subset Construction.

- DFA states are *subsets* of S but each subset plays the role of a single state!
- Transitions: for a DFA state in $Q \subseteq S$ and a letter $x \in \Sigma$:

$$\begin{aligned} N(Q, x) &= \{s_1 \in S \mid s \xrightarrow{x} s_1 \text{ for some } s \in Q\} \\ &= \{s_1 \in S \mid (s, x, s_1) \in R \text{ for some } s \in Q\} \end{aligned}$$

Example.

- Let $\{q_3, q_7\} \subseteq S$ be a DFA state (i.e., q_3 and q_7 are NFA states).
- Let $(q_3, 0, q_3) \in R$, $(q_3, 0, q_5) \in R$, $(q_3, 1, q_{42}\} \in R$ (and no others for q_3)

Subset Construction: The Finer Points

Input. Let NFA $A = (\Sigma, S, s_0, F, R)$.

Subset Construction.

- DFA states are *subsets* of S but each subset plays the role of a single state!
- Transitions: for a DFA state in $Q \subseteq S$ and a letter $x \in \Sigma$:

$$\begin{aligned} N(Q, x) &= \{s_1 \in S \mid s \xrightarrow{x} s_1 \text{ for some } s \in Q\} \\ &= \{s_1 \in S \mid (s, x, s_1) \in R \text{ for some } s \in Q\} \end{aligned}$$

Example.

- Let $\{q_3, q_7\} \subseteq S$ be a DFA state (i.e., q_3 and q_7 are NFA states).
- Let $(q_3, 0, q_3) \in R$, $(q_3, 0, q_5) \in R$, $(q_3, 1, q_{42}\} \in R$ (and no others for q_3)
- Let $(q_7, 0, q_8) \in R$ (and none else, also not for letter 1)

Subset Construction: The Finer Points

Input. Let NFA $A = (\Sigma, S, s_0, F, R)$.

Subset Construction.

- DFA states are *subsets* of S but each subset plays the role of a single state!
- Transitions: for a DFA state in $Q \subseteq S$ and a letter $x \in \Sigma$:

$$\begin{aligned} N(Q, x) &= \{s_1 \in S \mid s \xrightarrow{x} s_1 \text{ for some } s \in Q\} \\ &= \{s_1 \in S \mid (s, x, s_1) \in R \text{ for some } s \in Q\} \end{aligned}$$

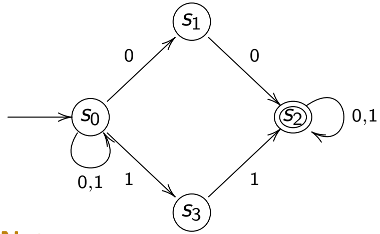
Example.

- Let $\{q_3, q_7\} \subseteq S$ be a DFA state (i.e., q_3 and q_7 are NFA states).
- Let $(q_3, 0, q_3) \in R$, $(q_3, 0, q_5) \in R$, $(q_3, 1, q_{42}\} \in R$ (and no others for q_3)
- Let $(q_7, 0, q_8) \in R$ (and none else, also not for letter 1)

$\rightarrow$ Then, we get $\{q_3, q_7\} \xrightarrow{0} \{q_3, q_5, q_8\}$ and $\{q_3, q_7\} \xrightarrow{1} \{q_{42}\}$

Determinisation: Example

The "double digits" automaton



Subset Construction: transition table

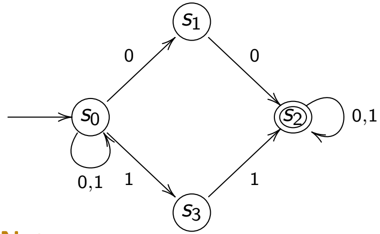
	0	1
$\to \{s_0\}$		

Note.

- don't have transition for *all* states, just those reachable from $\{s_0\}$
- all others are not relevant (cf. elimination of unreachable states)
- having all states would require $2^4 = 16$ entries.

Determinisation: Example

The "double digits" automaton



Subset Construction: transition table

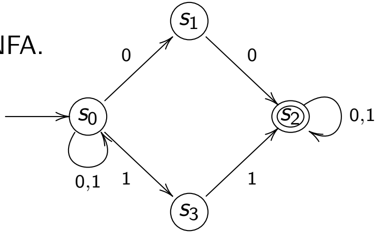
	0	1
$\rightarrow \{s_0\}$	$\{s_0, s_1\}$	$\{s_0, s_3\}$
$\{s_0, s_1\}$	$\{s_0, s_1, s_2\}$	$\{s_0, s_3\}$
$\{s_0, s_3\}$	$\{s_0, s_1\}$	$\{s_0, s_2, s_3\}$
$\odot\ \{s_0, s_1, s_2\}$	$\{s_0, s_1, s_2\}$	$\{s_0, s_2, s_3\}$
$\odot\ \{s_0, s_2, s_3\}$	$\{s_0, s_1, s_2\}$	$\{s_0, s_2, s_3\}$

Note.

- don't have transition for *all* states, just those reachable from $\{s_0\}$
- all others are not relevant (cf. elimination of unreachable states)
- having all states would require $2^4 = 16$ entries.
- Once the table is complete replace each DFA state set by a simple name

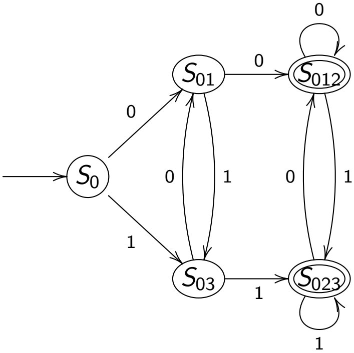
Determinisation Example, as Diagrams

Double Digits, as NFA.



Double Digits as DFA.

	0	1
$\rightarrow \{s_0\}$	$\{s_0, s_1\}$	$\{s_0, s_3\}$
$\{s_0, s_1\}$	$\{s_0, s_1, s_2\}$	$\{s_0, s_3\}$
$\{s_0, s_3\}$	$\{s_0, s_1\}$	$\{s_0, s_2, s_3\}$
$\odot \{s_0, s_1, s_2\}$	$\{s_0, s_1, s_2\}$	$\{s_0, s_2, s_3\}$
$\odot \{s_0, s_2, s_3\}$	$\{s_0, s_1, s_2\}$	$\{s_0, s_2, s_3\}$



NFAs with ε-transitions

More Expressive Power: ϵ-transitions

Extra Ingredient: Spontaneous transitions that don't "consume" a letter

- NFAs that may change state *without* consuming a symbol.
- NFAs of this kind are called *NFAs with ϵ-transitions*
- can convert NFAs with ϵ-transitions to (standard) NFAs

More Expressive Power: ϵ-transitions

Extra Ingredient: Spontaneous transitions that don't "consume" a letter

- NFAs that may change state *without* consuming a symbol.
- NFAs of this kind are called *NFAs with ϵ-transitions*
- can convert NFAs with ϵ-transitions to (standard) NFAs

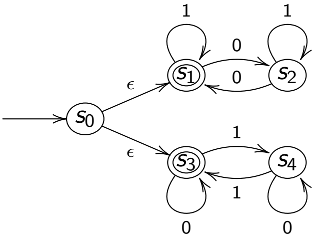
Formal Definition. An NFA with ϵ-transitions is an NFA, but the transition relation has the form

$$R \subseteq S \times \Sigma \cup \{\epsilon\} \times S$$

- cf. NFAs with transition relation $R \subseteq S \times \Sigma \times S$
- $R(s, \epsilon, s')$ is a spontaneous transition (without reading input symbol)
- ϵ is *not* an element of the alphabet!

ϵ-NFA: Example

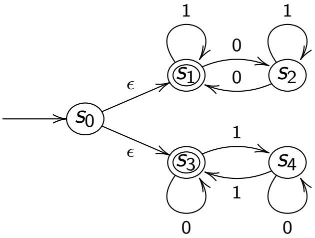
General Pattern. ϵ-transitions say "or"



Q. What does that automaton do?

ϵ-NFA: Example

General Pattern. ϵ-transitions say "or"



Q. What does that automaton do?

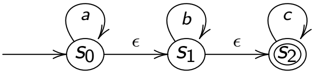
A. Interpretation:

- "top" automaton (with start state s_1) requires even number of 0's
- "bottom" automaton (with start state s_3) requires even number of 1's

$\rightarrow$ entire automaton (with start state s_0) accepts *either* an even number of 1's *or* an even number of 0's

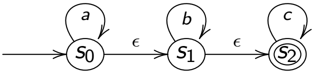
Example and Acceptance

Language of this Automaton?



Example and Acceptance

Language of this Automaton?



Acceptance Informally. An ϵ-NFA A *accepts* a word $w = x_1 \dots x_n$ if there is a sequence of states

$$s_0 \xrightarrow{\epsilon^*} s_1 \xrightarrow{x_1} s_1' \xrightarrow{\epsilon^*} s_2 \xrightarrow{x_2} s_2' \dots s_n \xrightarrow{x_n} s_n' \xrightarrow{\epsilon^*} f$$

where s_0 is the starting state, $f \in F$ is an accepting state and

- $s_i \xrightarrow{x} s_j$ if there is an x-transition from s_i to s_j, i.e., $(s_i, x, s_j) \in R$
- $s_i \xrightarrow{\epsilon^*} s_j$ if there is a sequence of ϵ-transitions from s_i to s_j.

In particular: the empty string $\epsilon \in L(A)$ if $s_0 \xrightarrow{\epsilon^*} f$ for a final state $f \in F$.

Eventual State Relation for ϵ-NFAs

ϵ-closure. For an ϵ-NFA (Σ, S, s_0, F, R), the *ϵ-closure* of a state $s \in S$ is given by: $\quad \text{eclose}(s) = \{s' \in S \mid \text{ there is a sequence of } \epsilon\text{-transitions from } s \text{ to } s'\}$
(Note that it always holds: $\text{eclose}(s) \supseteq \{s\}$ as base-case.)

Eventual State Relation for ϵ-NFAs

ϵ-closure. For an ϵ-NFA (Σ, S, s_0, F, R), the *ϵ-closure* of a state $s \in S$ is given by: $\text{eclose}(s) = \{s' \in S \mid \text{ there is a sequence of } \epsilon\text{-transitions from } s \text{ to } s'\}$
(Note that it always holds: $\text{eclose}(s) \supseteq \{s\}$ as base-case.)

and the *eventual state relation* is given by

$$R^*(s, \epsilon, s') \iff s' \in \text{eclose}(s)$$

Eventual State Relation for ϵ-NFAs

ϵ-closure. For an ϵ-NFA (Σ, S, s_0, F, R), the *ϵ-closure* of a state $s \in S$ is given by: $\quad \text{eclose}(s) = \{s' \in S \mid \text{ there is a sequence of } \epsilon\text{-transitions from } s \text{ to } s'\}$
(Note that it always holds: $\text{eclose}(s) \supseteq \{s\}$ as base-case.)

and the *eventual state relation* is given by

$$
\begin{aligned}
R^*(s, \epsilon, s') &\iff s' \in \text{eclose}(s) \\
R^*(s, x\alpha, s') &\iff \text{ there are } s_0 \text{ and } s_1 \text{ such that} \\
&\qquad s_0 \in \text{eclose}(s), (s_0, x, s_1) \in R, (s_1, \alpha, s') \in R^*
\end{aligned}
$$

Eventual State Relation for ϵ-NFAs

ϵ-closure. For an ϵ-NFA (Σ, S, s_0, F, R), the *ϵ-closure* of a state $s \in S$ is given by: $\text{eclose}(s) = \{s' \in S \mid$ there is a sequence of ϵ-transitions from s to $s'\}$
(Note that it always holds: $\text{eclose}(s) \supseteq \{s\}$ as base-case.)

and the *eventual state relation* is given by

$$R^*(s, \epsilon, s') \iff s' \in \text{eclose}(s)$$

$$R^*(s, x\alpha, s') \iff \text{there are } s_0 \text{ and } s_1 \text{ such that } s_0 \in \text{eclose}(s), (s_0, x, s_1) \in R, (s_1, \alpha, s') \in R^*$$

Acceptance (and language) for DFAs / NFAs:
A string w is *accepted* by an ϵ-NFA A (in symbols: $w \in L(A)$) if $(s_0, w, f) \in R^*$ for some final state $f \in F$, that is

$$L(A) = \{w \in \Sigma^* \mid \exists f \in F.(s_0, w, f) \in R^*\}$$

Relationship Between NFAs and ϵ-NFAs

Q. Are there languages *only* accepted by ϵ-NFAs?

A. No.

Relationship Between NFAs and ϵ-NFAs

Q. Are there languages *only* accepted by ϵ-NFAs?

A. No. Every ϵ-NFA $A = (\Sigma, S, s_0, F, R)$ can be converted to an NFA A' without ϵ-transitions so that $L(A) = L(A')$.

Relationship Between NFAs and ϵ-NFAs

Q. Are there languages *only* accepted by ϵ-NFAs?

A. No. Every ϵ-NFA $A = (\Sigma, S, s_0, F, R)$ can be converted to an NFA A' without ϵ-transitions so that $L(A) = L(A')$.

Construction. Define $A' = (\Sigma, S, s_0, F', R')$, such that:

- We make $s \in S$ an accepting state *in A'* if s can reach an accepting state *in A* by ϵ-transitions:

$$F' = \{s \in S \mid \text{eclose}(s) \cap F \neq \emptyset\}$$

Relationship Between NFAs and ϵ-NFAs

Q. Are there languages *only* accepted by ϵ-NFAs?

A. No. Every ϵ-NFA $A = (\Sigma, S, s_0, F, R)$ can be converted to an NFA A' without ϵ-transitions so that $L(A) = L(A')$.

Construction. Define $A' = (\Sigma, S, s_0, F', R')$, such that:

- We make $s \in S$ an accepting state *in A'* if s can reach an accepting state *in A* by ϵ-transitions:

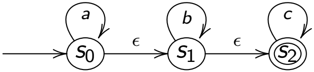
$$F' = \{s \in S \mid \text{eclose}(s) \cap F \neq \emptyset\}$$

- Put an arc $s \xrightarrow{x} t$ *into A'* if there is some $s' \in \text{eclose}(s)$, such that $s' \xrightarrow{x} t$ *in A*. Formally:

$$R' = \{(s, x, t) \mid (s', x, t) \in R \text{ for some } s' \in \text{eclose}(s)\}$$

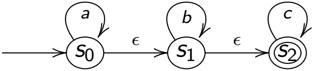
(double-check that A and A' accept the same strings!)

Example for ϵ-Elimination



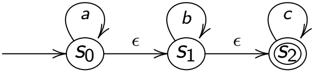
- Make $s \in S$ an accepting state *in A'* if s can reach an accepting state *in A* by ϵ-transitions: $F' = \{s \in S \mid \text{eclose}(s) \cap F \neq \emptyset\}$
  - $\rightarrow$ All states here can reach a goal state with only ϵ-transitions!

Example for ϵ-Elimination



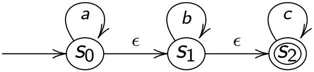
- Make $s \in S$ an accepting state *in A'* if s can reach an accepting state *in A* by ϵ-transitions: $F' = \{s \in S \mid \text{eclose}(s) \cap F \neq \emptyset\}$
  - $\rightarrow$ All states here can reach a goal state with only ϵ-transitions!
- Put an arc $s \xrightarrow{x} t$ *into A'* if there is a transition $s' \xrightarrow{x} t$ *in A* with $s' \in \text{eclose}(s)$: $R' = \{(s, x, t) \mid (s', x, t) \in R \text{ for some } s' \in \text{eclose}(s)\}$

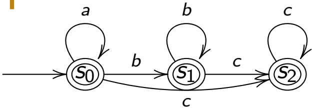
Example for ϵ-Elimination



- Make $s \in S$ an accepting state *in A'* if s can reach an accepting state *in A* by ϵ-transitions: $F' = \{s \in S \mid \text{eclose}(s) \cap F \neq \emptyset\}$
  - $\rightarrow$ All states here can reach a goal state with only ϵ-transitions!
- Put an arc $s \xrightarrow{x} t$ *into A'* if there is a transition $s' \xrightarrow{x} t$ *in A* with $s' \in \text{eclose}(s)$: $R' = \{(s, x, t) \mid (s', x, t) \in R \text{ for some } s' \in \text{eclose}(s)\}$
  - $\rightarrow$ Test $s_0 \xrightarrow{a} s_1$: Does $s' \in \text{eclose}(s_0)$ exist, s.t. $s' \xrightarrow{a} s_1$?
  - $\rightarrow$ Test $s_0 \xrightarrow{b} s_1$: Does $s' \in \text{eclose}(s_0)$ exist, s.t. $s' \xrightarrow{b} s_1$?
  - $\rightarrow$ Test $s_0 \xrightarrow{c} s_1$: Does $s' \in \text{eclose}(s_0)$ exist, s.t. $s' \xrightarrow{c} s_1$?
  - $\rightarrow$ Test $s_0 \xrightarrow{a} s_2$: Does $s' \in \text{eclose}(s_0)$ exist, s.t. $s' \xrightarrow{a} s_2$?
  - $\rightarrow$ Test $s_0 \xrightarrow{b} s_2$: Does $s' \in \text{eclose}(s_0)$ exist, s.t. $s' \xrightarrow{b} s_2$?
  - $\rightarrow$ Test $s_0 \xrightarrow{c} s_2$: Does $s' \in \text{eclose}(s_0)$ exist, s.t. $s' \xrightarrow{c} s_2$?
  - $\rightarrow$ Test $s_1 \xrightarrow{a} s_2$: Does $s' \in \text{eclose}(s_1)$ exist, s.t. $s' \xrightarrow{a} s_2$?
  - $\rightarrow$ Test $s_1 \xrightarrow{b} s_2$: Does $s' \in \text{eclose}(s_1)$ exist, s.t. $s' \xrightarrow{b} s_2$?
  - $\rightarrow$ Test $s_1 \xrightarrow{c} s_2$: Does $s' \in \text{eclose}(s_1)$ exist, s.t. $s' \xrightarrow{c} s_2$?

Example for ϵ-Elimination





- Make $s \in S$ an accepting state *in A'* if s can reach an accepting state *in A* by ϵ-transitions: $F' = \{s \in S \mid \mathrm{eclose}(s) \cap F \neq \emptyset\}$
  - $\rightarrow$ All states here can reach a goal state with only ϵ-transitions!
- Put an arc $s \xrightarrow{x} t$ *into A'* if there is a transition $s' \xrightarrow{x} t$ *in A* with $s' \in \mathrm{eclose}(s)$: $R' = \{(s, x, t) \mid (s', x, t) \in R \text{ for some } s' \in \mathrm{eclose}(s)\}$
  - $\rightarrow$ Test $s_0 \xrightarrow{a} s_1$: Does $s' \in \mathrm{eclose}(s_0)$ exist, s.t. $s' \xrightarrow{a} s_1$? No
  - $\rightarrow$ Test $s_0 \xrightarrow{b} s_1$: Does $s' \in \mathrm{eclose}(s_0)$ exist, s.t. $s' \xrightarrow{b} s_1$? Yes! s_1
  - $\rightarrow$ Test $s_0 \xrightarrow{c} s_1$: Does $s' \in \mathrm{eclose}(s_0)$ exist, s.t. $s' \xrightarrow{c} s_1$? No
  - $\rightarrow$ Test $s_0 \xrightarrow{a} s_2$: Does $s' \in \mathrm{eclose}(s_0)$ exist, s.t. $s' \xrightarrow{a} s_2$? No
  - $\rightarrow$ Test $s_0 \xrightarrow{b} s_2$: Does $s' \in \mathrm{eclose}(s_0)$ exist, s.t. $s' \xrightarrow{b} s_2$? No
  - $\rightarrow$ Test $s_0 \xrightarrow{c} s_2$: Does $s' \in \mathrm{eclose}(s_0)$ exist, s.t. $s' \xrightarrow{c} s_2$? Yes! s_2
  - $\rightarrow$ Test $s_1 \xrightarrow{a} s_2$: Does $s' \in \mathrm{eclose}(s_1)$ exist, s.t. $s' \xrightarrow{a} s_2$? No
  - $\rightarrow$ Test $s_1 \xrightarrow{b} s_2$: Does $s' \in \mathrm{eclose}(s_1)$ exist, s.t. $s' \xrightarrow{b} s_2$? No
  - $\rightarrow$ Test $s_1 \xrightarrow{c} s_2$: Does $s' \in \mathrm{eclose}(s_1)$ exist, s.t. $s' \xrightarrow{c} s_2$? Yes! s_2

Regular Expressions

Regular Expressions

Regular Expressions are another way to describe formal languages.

Basic Operators used to construct new expressions from old:

- ϵ, the empty string, and every letter of the alphabet

Examples.

Regular Expressions

Regular Expressions are another way to describe formal languages.

Basic Operators used to construct new expressions from old:

- ϵ, the empty string, and every letter of the alphabet
- Kleene star and Plus: repeat strings from an expression

Examples.

- a^* indicates 0 or more as: $\{\alpha \mid \alpha \in \{a\}^*\}$
- a^+ indicates 1 or more as: $\{a\alpha \mid \alpha \in \{a\}^*\}$

Regular Expressions

Regular Expressions are another way to describe formal languages.

Basic Operators used to construct new expressions from old:

- ϵ, the empty string, and every letter of the alphabet
- Kleene star and Plus: repeat strings from an expression
- vertical bar (pipe): choose either the left or right expression
- concatenation, for sequencing expressions

Examples.

- a^* indicates 0 or more as: $\{\alpha \mid \alpha \in \{a\}^*\}$
- a^+ indicates 1 or more as: $\{a\alpha \mid \alpha \in \{a\}^*\}$
- $\mathrm{yes} \mid \mathrm{no}$ is the language with just the 2 given strings: { yes, no }

Regular Expressions

Regular Expressions are another way to describe formal languages.

Basic Operators used to construct new expressions from old:

- ϵ, the empty string, and every letter of the alphabet
- Kleene star and Plus: repeat strings from an expression
- vertical bar (pipe): choose either the left or right expression
- concatenation, for sequencing expressions
- parentheses, for grouping

Examples.

- a^* indicates 0 or more as: $\{\alpha \mid \alpha \in \{a\}^*\}$
- a^+ indicates 1 or more as: $\{a\alpha \mid \alpha \in \{a\}^*\}$
- $\mathrm{yes} \mid \mathrm{no}$ is the language with just the 2 given strings: { yes, no }
- $(0 \mid 1)^*$ indicates the set of binary numerals: $\{\alpha \mid \alpha \in \{0, 1\}^*\}$

Regular Expressions — More Examples

- A single zero or binary numerals without leading zero:

Regular Expressions — More Examples

- A single zero or binary numerals without leading zero: $0|(1(0|1)^*)$

Regular Expressions — More Examples

- A single zero or binary numerals without leading zero: $0|(1(0|1)^*)$
- The set of strings over $\{a, b, c\}$ with just one c:

Regular Expressions — More Examples

- A single zero or binary numerals without leading zero: $0|(1(0|1)^*)$
- The set of strings over $\{a, b, c\}$ with just one c: $(a \mid b)^* c (a \mid b)^*$

Regular Expressions — More Examples

- A single zero or binary numerals without leading zero: $0|(1(0|1)^*)$
- The set of strings over $\{a, b, c\}$ with just one c: $(a \mid b)^* c (a \mid b)^*$
- The language of bit-strings that have an even number of 1s:

Regular Expressions — More Examples

- A single zero or binary numerals without leading zero: $0|(1(0|1)^*)$
- The set of strings over $\{a, b, c\}$ with just one c: $(a \mid b)^* c (a \mid b)^*$
- The language of bit-strings that have an even number of 1s: $0^*(10^*10^*)^*$
  (Zero is even, so $0 \dots 0$ should be accepted. Thus, $(0^*10^*10^*)^*$ is wrong)

Regular Expressions — More Examples

- A single zero or binary numerals without leading zero: $0|(1(0|1)^*)$
- The set of strings over $\{a, b, c\}$ with just one c: $(a \mid b)^* c (a \mid b)^*$
- The language of bit-strings that have an even number of 1s: $0^*(10^*10^*)^*$
  (Zero is even, so $0 \ldots 0$ should be accepted. Thus, $(0^*10^*10^*)^*$ is wrong)
- What's $((z^*(x^* \mid y^*)\ z))^*$?

Regular Expressions — More Examples

- A single zero or binary numerals without leading zero: $0|(1(0|1)^*)$
- The set of strings over $\{a, b, c\}$ with just one c: $(a \mid b)^* c (a \mid b)^*$
- The language of bit-strings that have an even number of 1s: $0^*(10^*10^*)^*$
  (Zero is even, so $0 \ldots 0$ should be accepted. Thus, $(0^*10^*10^*)^*$ is wrong)
- What's $((z^*(x^* \mid y^*)\ z))^*$?
  The set of strings over $\{x, y, z\}$ with no x and y adjacent.

Regular Expressions — More Examples

- A single zero or binary numerals without leading zero: $0|(1(0|1)^*)$
- The set of strings over $\{a, b, c\}$ with just one c: $(a \mid b)^* c (a \mid b)^*$
- The language of bit-strings that have an even number of 1s: $0^*(10^*10^*)^*$
  (Zero is even, so $0 \ldots 0$ should be accepted. Thus, $(0^*10^*10^*)^*$ is wrong)
- What's $((z^*(x^* \mid y^*)\ z))^*$?
  The set of strings over $\{x, y, z\}$ with no x and y adjacent.
- What's $1 \mid (0\ (\ \epsilon\ |(.(0 \mid 1)^*1))))$? (Here, $\Sigma = \{\ .\ ,\ 0\ ,\ 1\ \}$)

Regular Expressions — More Examples

- A single zero or binary numerals without leading zero: $0|(1(0|1)^*)$
- The set of strings over $\{a, b, c\}$ with just one c: $(a \mid b)^* c(a \mid b)^*$
- The language of bit-strings that have an even number of 1s: $0^*(10^*10^*)^*$
  (Zero is even, so $0 \dots 0$ should be accepted. Thus, $(0^*10^*10^*)^*$ is wrong)
- What's $((z^*(x^* \mid y^*)\ z))^*$?
  The set of strings over $\{x, y, z\}$ with no x and y adjacent.
- What's $1 \mid (0\ (\ \epsilon\ |(.(0 \mid 1)^*1))))$? (Here, $\Sigma = \{\ .\ ,\ 0\ ,\ 1\ \}$)
  The binary fractional numerals between 0 and 1 with no trailing zeroes. E.g. 1, 0, 0.1, 0.110011 but not .1 or 0.10. The last 1 in the expression is required to prevent redundant zeros at the end.

The Definition of Regular Expressions

Key Concept.

- regular expressions are purely *syntactical* – just like formulae
- *but:* every expression denotes a set of strings – this is the meaning.

Definition.

The regular expressions over alphabet Σ and the sets that they denote are:

- $\emptyset$ is a regular expression and denotes the empty set $\emptyset$
- ϵ is a regular expression and denotes the set $\{\epsilon\}$
- for each $a \in \Sigma$, a is a regular expression and denotes the set $\{a\}$

The Definition of Regular Expressions

Key Concept.

- regular expressions are purely *syntactical* – just like formulae
- *but:* every expression denotes a set of strings – this is the meaning.

Definition.

The regular expressions over alphabet Σ and the sets that they denote are:

- $\emptyset$ is a regular expression and denotes the empty set $\emptyset$
- ϵ is a regular expression and denotes the set $\{\epsilon\}$
- for each $a \in \Sigma$, a is a regular expression and denotes the set $\{a\}$
- If α and β are regular expressions denoting languages R and S resp., then:
  - $\alpha \mid \beta$ denotes $R \cup S$
  - $\alpha\,\beta$ denotes RS which is $\{ww' \mid w \in R \land w' \in S\}$
  - α^* denotes R^*, i.e., the set of *finitely* many $r_i \in R$, concatenated, i.e., R^* is (inductively) defined as $\{\epsilon\} \cup RR^*$

Regular Expressions and DFAs

Key Insights.

- For every DFA A, we have regular expression r with $L(A) = L(r)$.
  (We didn't show or even state that yet!)
- For every regular expression r, we have a DFA A with $L(r) = L(A)$.
  (You will see this in the next few slides.)
- Recall that we already showed the equivalence of DFAs and NFAs.
- Thus, the "power" of NFAs / DFAs are *completely described* by regular expressions (and vice versa). In other words:

  DFAs, NFAs, and regular expressions are all equally expressive.

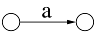
Regular Expressions to ϵ-NFAs

Key Insight.

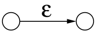
- regular expressions are an *inductively defined structure*
- e.g., representable by an inductive data type in Haskell
- as a consequence, we can give *inductive definition* of the corresponding automaton

Construction. (start state on left, final state on right)

- When the regular expression is a symbol a of the alphabet (language is $\{a\}$) the automaton is



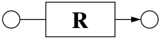
- When the regular expression is ϵ (language is $\{\epsilon\}$) the automaton is



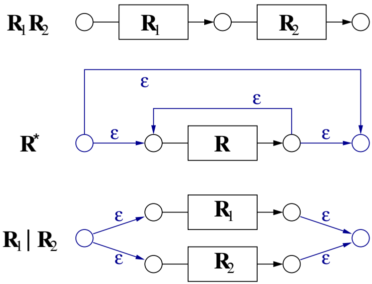
- When the regular expression is $\emptyset$ (language is $\emptyset$) the automaton has no edges

Regular Expressions to NFAs, ctd

Suppose the NFA corresponding to some regular expression R is:



Then, we can inductively define the NFAs corresponding to composite regular expressions as follows:

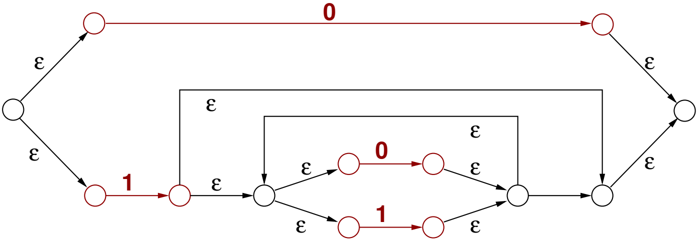


Example

Given the regular expression for binary numerals without leading zeros, $(0 \mid 1(0|1)^*)$, the previous algorithm (the inductive definition) gives this NFA:

Example

Given the regular expression for binary numerals without leading zeros, $(0 \mid 1(0|1)^*)$, the previous algorithm (the inductive definition) gives this NFA:



Closing the Loop

Given a finite alphabet Σ and a language $L \subseteq \Sigma^*$. The following are equivalent:

- L can be described by a regular expression
- L can be recognised by an ϵ-NFA
- L can be recognised by an NFA
- L can be recognised by a DFA ...

as we can convert regular expressions into ϵ-NFAs into NFAs into DFAs.

Missing Link.

Construction of regular expressions from DFAs (not covered in this course).

Summary

Summary.

Starting Point. Finite Automata

- motivated by computers having finite memory (only)
- solving simple problems: is string s accepted?

Limitations of Finite Automata

- Some languages can't be recognised, e.g., $L = \{a^n b^n \mid n \geq 0\}$

Characterisation of expressive power

- can go back and forth between automata and regular expressions

Q. Are finite automata a "good" model of computation?

- if yes, why?
- if not, why not? What is missing?

Literature.

- *Introduction to Automata Theory, Languages, and Computation*
  By Hopcroft, Motwani, and Ullman.

  A classic text that has been re-worked from a standard textbook.
- *Introduction To The Theory Of Computation*
  by Michael Sipser

  The part on Automata and Languages covers (more than) what we have discussed here.
- There are tons of exercises one can practice with.
  (Online and in our repository.)