COMP1600, week 7: Deterministic Finite Automata (DFAs)

convenors: Dirk Pattinson, Pascal Bercher

lecturer: Pascal Bercher

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

Australian National University

Semester 2, 2024

Overview of Week 7

- Motivation
- Introduction to Automata and Formal Languages
- Deterministic Finite Automata — Formally —
- Language of an Automaton
- Minimisation of DFAs
- Limitations of FSAs

Motivation

The Story So Far . . .

Logic.

- language and proofs to speak about systems *precisely*
- useful to *express properties* and *do proofs*

Establish *properties of programs*.

- *Functional Programs.* Main tool: (structural) induction / Dafny
- *Imperative Programs.* Main tool: Hoare Logic / Dafny

Q. Is there a *general* notion of computation that encompasses *both*?

First Shot: Your Laptop

Abstract Characteristics.

- can do computation
- has memory – a finite amount
- has (lots of) internal states

From Laptops to Formal Models

Concrete (your laptop)

- realistic (it exists!)
- complex
- hard to analyse

Abstract (mathematical model)

- exists only as a model
- simple
- easy to analyse

Q. What is a "*good*" simple model of computation?

- should be able to differentiate different problem solving capabilities
- should match what really exists (possibly by a long shot)
- should be *conceptually simple*

First Answer: Finite State Automata

Basic Components.

- internal states – finitely many
- state transitions – triggered by reading input
- simplifying assumption: just *one* output: yes/no

Data.

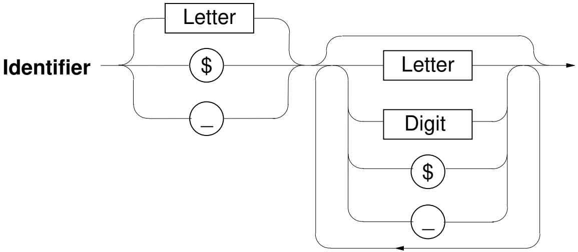
- basic input: strings (what you type in, text/XML/JSON file etc.)
- characters: drawn from *finite* set (alphabet, e.g., letters, numbers)

Example: Java Identifiers

From Oracle's Java Language Specification.

An identifier is a sequence of one or more characters. The first character must be a valid first character (letter, $, _) in an identifier of the Java programming language, hereafter in this chapter called simply "Java". Each subsequent character in the sequence must be a valid nonfirst character (letter, digit, $, _) in a Java identifier.

Graphical Specification

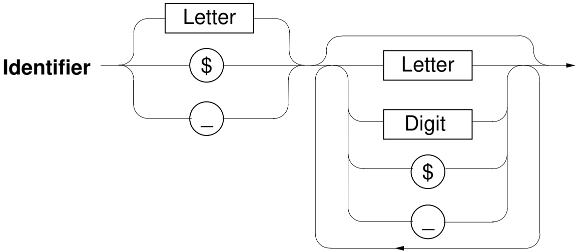


Example: Java Identifiers

From Oracle's Java Language Specification.

An identifier is a sequence of one or more characters. The first character must be a valid first character (letter, $, _) in an identifier of the Java programming language, hereafter in this chapter called simply "Java". Each subsequent character in the sequence must be a valid nonfirst character (letter, digit, $, _) in a Java identifier.

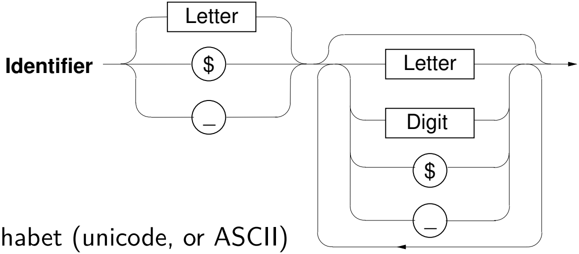
Graphical Specification



Q. Can you see a "machine" that *recognises* Java? identifiers?

Java Identifiers

Example: Main Components



Data.

- drawn form a finite alphabet (unicode, or ASCII)

Control.

- "yes" if I can get from the left to the right, "no" otherwise
- have *states* after taking a transition (implicit in diagram)

Computational Problem with yes/no answer:

- is a given sequence of characters a valid Java identifier?

Preview.

Next two weeks. Finite Automata

- start with simplest model: finite automata
- relate to regular languages, non-determinism
- conclusion: finite automata "too simple"

The week after. Pushdown automata

- like finite automata, but some more memory
- useful for e.g. specifying syntax of programming languages
- still "too simple" for general computation

Then. Turing machines

- *The* most widely accepted model of computation
- *infinite* memory
- idea: buy another hard disk whenever your computation runs out of memory
- *limits* of what can be computed

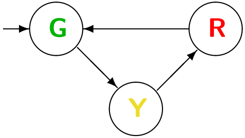
Introduction to Automata and Formal Languages

Finite State Automata: First Example

The simplest useful abstraction of a **"computing machine"** consists of:

- A fixed, finite set of **states**
- A **transition relation** over the states

Example: a traffic light Finite State Automaton (FSA) has 3 states:



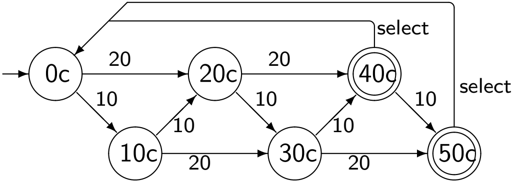
G names state in which light is green.
Y names state in which light is yellow.
R names state in which light is red.

System designs are often in terms of state machines.
There are many extensions, e.g., how long does it stay red, how long green, etc.

Second Example: Vending Machine

Operation

- accept 10c and 20c coins
- delivers if it has received at least 40c and selection is made



Note.

- transitions are *labelled*
- new ingredient: *final states* (doubly circled)

Computation. Sequences of actions (labels) from initial to final state.

Language Examples

Main Idea.

- input: a string over a fixed character set
- operation: transitions labelled with characters
- output: yes if in final state after reading the input

More Generally.

- Setup: Fix a finite set of characters (an alphabet)
- Problem: A set of strings (called *language*) that are "valid" or "good"
- Task: decide computationally which strings are "good"

Example Languages.

1. A finite set: $\{a, aa, ab, aaa, aab, aba, abb\}$

Language Examples

Main Idea.

- input: a string over a fixed character set
- operation: transitions labelled with characters
- output: yes if in final state after reading the input

More Generally.

- Setup: Fix a finite set of characters (an alphabet)
- Problem: A set of strings (called *language*) that are "valid" or "good"
- Task: decide computationally which strings are "good"

Example Languages.

1. A finite set: $\{a, aa, ab, aaa, aab, aba, abb\}$
2. All valid payments: $\{(20, 20), \ldots (10, 10, 10, 10, 10)\}$ (also finite!)

Language Examples

Main Idea.

- input: a string over a fixed character set
- operation: transitions labelled with characters
- output: yes if in final state after reading the input

More Generally.

- Setup: Fix a finite set of characters (an alphabet)
- Problem: A set of strings (called *language*) that are "valid" or "good"
- Task: decide computationally which strings are "good"

Example Languages.

1. A finite set: $\{a, aa, ab, aaa, aab, aba, abb\}$
2. All valid payments: $\{(20, 20), \ldots (10, 10, 10, 10, 10)\}$ (also finite!)
3. Palindromes over 0/1: $\{\varepsilon, 0, 1, 00, 11, 010, 101, 000, 111, 0110, ...\}$ (infinite)

Language Examples

Main Idea.

- input: a string over a fixed character set
- operation: transitions labelled with characters
- output: yes if in final state after reading the input

More Generally.

- Setup: Fix a finite set of characters (an alphabet)
- Problem: A set of strings (called *language*) that are "valid" or "good"
- Task: decide computationally which strings are "good"

Example Languages.

1. A finite set: $\{a, aa, ab, aaa, aab, aba, abb\}$
2. All valid payments: $\{(20, 20), \ldots (10, 10, 10, 10, 10)\}$ (also finite!)
3. Palindromes over 0/1: $\{\varepsilon, 0, 1, 00, 11, 010, 101, 000, 111, 0110, \ldots\}$ (infinite)
4. All syntactically valid Java programs: $\{\ldots\}$ (later! :))

Language Examples

Main Idea.

- input: a string over a fixed character set
- operation: transitions labelled with characters
- output: yes if in final state after reading the input

More Generally.

- Setup: Fix a finite set of characters (an alphabet)
- Problem: A set of strings (called *language*) that are "valid" or "good"
- Task: decide computationally which strings are "good"

Example Languages.

1. A finite set: $\{a, aa, ab, aaa, aab, aba, abb\}$
2. All valid payments: $\{(20, 20), \ldots (10, 10, 10, 10, 10)\}$ (also finite!)
3. Palindromes over 0/1: $\{\varepsilon, 0, 1, 00, 11, 010, 101, 000, 111, 0110, \ldots\}$ (infinite)
4. All syntactically valid Java programs: $\{\ldots\}$ (later! :))
5. All Java (etc.) programs that terminate: $\{\ldots\}$ (later! :))

Language Examples

Main Idea.

- input: a string over a fixed character set
- operation: transitions labelled with characters
- output: yes if in final state after reading the input

More Generally.

- Setup: Fix a finite set of characters (an alphabet)
- Problem: A set of strings (called *language*) that are "valid" or "good"
- Task: decide computationally which strings are "good"

Example Languages.

1. A finite set: $\{a, aa, ab, aaa, aab, aba, abb\}$
2. All valid payments: $\{(20, 20), \ldots (10, 10, 10, 10, 10)\}$ (also finite!)
3. Palindromes over 0/1: $\{\varepsilon, 0, 1, 00, 11, 010, 101, 000, 111, 0110, ...\}$ (infinite)
4. All syntactically valid Java programs: $\{\ldots\}$ (later! :))
5. All Java (etc.) programs that terminate: $\{\ldots\}$ (later! :))
6. MANY more!

Language Examples

Main Idea.

- input: a string over a fixed character set
- operation: transitions labelled with characters
- output: yes if in final state after reading the input

More Generally.

- Setup: Fix a finite set of characters (an alphabet)
- Problem: A set of strings (called *language*) that are "valid" or "good"
- Task: decide computationally which strings are "good"

Example Languages.

1. A finite set: $\{a, aa, ab, aaa, aab, aba, abb\}$
2. All valid payments: $\{(20, 20), \ldots (10, 10, 10, 10, 10)\}$ (also finite!)
3. Palindromes over 0/1: $\{\varepsilon, 0, 1, 00, 11, 010, 101, 000, 111, 0110, \ldots\}$ (infinite)
4. All syntactically valid Java programs: $\{\ldots\}$ (later! :))
5. All Java (etc.) programs that terminate: $\{\ldots\}$ (later! :))
6. MANY more!

Languages in this sense are called *formal* languages.

Terminology

Alphabet.
A finite set (of symbols). Usually denoted by Σ.

Strings over an alphabet Σ
finite sequence of characters (elements of Σ), can be the empty sequence.
E.g. for $\Sigma = \{a, b, c\}$, $ababc$ is a string over Σ, and so is ε.

Languages over alphabet Σ
are just sets of strings over Σ.
(The language of an automaton is the set of strings accepted by it.)

Words of the language
just another name for the elements (strings) of the language.

Notation:

- Σ^* is the set of all strings over Σ.
- Therefore, every language with alphabet Σ is some **subset** of Σ^*.

Automata

First Model of Computation. Deterministic Finite Automata

- solve computational problem: given string (word) w, is w accepted?

Basic Ingredients. (see e.g. traffic light and vending machine example)

- The **alphabet** of a DFA is a finite set of ***input tokens*** that an automaton acts on.

Automata

First Model of Computation. Deterministic Finite Automata

- solve computational problem: given string (word) w, is w accepted?

Basic Ingredients. (see e.g. traffic light and vending machine example)

- The **alphabet** of a DFA is a finite set of ***input tokens*** that an automaton acts on.
- A DFA consists of a finite set of **states** (a primitive notion)

Automata

First Model of Computation. Deterministic Finite Automata

- solve computational problem: given string (word) w, is w accepted?

Basic Ingredients. (see e.g. traffic light and vending machine example)

- The **alphabet** of a DFA is a finite set of ***input tokens*** that an automaton acts on.
- A DFA consists of a finite set of **states** (a primitive notion)
- One of the states is the **initial** state — where the automaton starts

Automata

First Model of Computation. Deterministic Finite Automata

- solve computational problem: given string (word) w, is w accepted?

Basic Ingredients. (see e.g. traffic light and vending machine example)

- The **alphabet** of a DFA is a finite set of ***input tokens*** that an automaton acts on.
- A DFA consists of a finite set of **states** (a primitive notion)
- One of the states is the **initial** state — where the automaton starts
- At least one of the states is a **final** state

Automata

First Model of Computation. Deterministic Finite Automata

- solve computational problem: given string (word) w, is w accepted?

Basic Ingredients. (see e.g. traffic light and vending machine example)

- The **alphabet** of a DFA is a finite set of ***input tokens*** that an automaton acts on.
- A DFA consists of a finite set of **states** (a primitive notion)
- One of the states is the **initial** state — where the automaton starts
- At least one of the states is a **final** state
- A **transition function** *(next state function):*

$$State \times Token \;\rightarrow\; State$$

Automata

First Model of Computation. Deterministic Finite Automata

- solve computational problem: given string (word) w, is w accepted?

Basic Ingredients. (see e.g. traffic light and vending machine example)

- The **alphabet** of a DFA is a finite set of ***input tokens*** that an automaton acts on.
- A DFA consists of a finite set of **states** (a primitive notion)
- One of the states is the **initial** state — where the automaton starts
- At least one of the states is a **final** state
- A **transition function** *(next state function):*

$$State \times Token \ \rightarrow \ State$$

Q. What's the difference between a DFA and an FSA?

Automata

First Model of Computation. Deterministic Finite Automata

- solve computational problem: given string (word) w, is w accepted?

Basic Ingredients. (see e.g. traffic light and vending machine example)

- The **alphabet** of a DFA is a finite set of ***input tokens*** that an automaton acts on.
- A DFA consists of a finite set of **states** (a primitive notion)
- One of the states is the **initial** state — where the automaton starts
- At least one of the states is a **final** state
- A **transition function** *(next state function):*

$$State \times Token \rightarrow State$$

Q. What's the difference between a DFA and an FSA?

Disclaimer: We often use FSA when a specific FSA is meant (like a DFA), because it should be clear from context which is meant.

Recurring Theme

Diagrammatic Notation.

- useful for humans
- e.g. the transition diagram of the vending machine

Mathematical Notation.

- useful for formal manipulation (e.g. proving theorems)
- useful for computer implementation

Glue between Diagrams and Maths

- both notions convey *precisely* the same information
- crucial: being able to switch back and forth!

Deterministic Finite Automata — Formally —

Formal Definition of DFA

A Deterministic Finite State Automaton (DFA) consists of five parts:

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

- a finite input **alphabet** Σ, the set of tokens
- a finite set of **states** S
- an **initial** state $s_0 \in S$ (we start here)
- a set of **final** states $F \subseteq S$ (helps defining which strings are accepted)
- a **transition function** $N : S \times \Sigma \rightarrow S$

Formal Definition of DFA

A Deterministic Finite State Automaton (DFA) consists of five parts:

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

- a finite input **alphabet** Σ, the set of tokens
- a finite set of **states** S
- an **initial** state $s_0 \in S$ (we start here)
- a set of **final** states $F \subseteq S$ (helps defining which strings are accepted)
- a **transition function** $N : S \times \Sigma \rightarrow S$

Aside. Having a transition *function* is what makes the automaton deterministic.

- It requires a unique successor for each state-token pair
- (Also, we have a successor for each state-token pair)

Finite State Automata as String Acceptors

Idea. A finite state automaton

- works on *strings* over an alphabet Σ
- determines which strings in Σ^* are "good" (accepted) and which strings are "bad" (rejected). (Recall our two initial examples)

Acceptance Informally. Let $A = (\Sigma, S, s_0, F, N)$ be a DFA. Then A *accepts* the string $w = x_1 x_2 \dots x_n$ iff there is a sequence of states

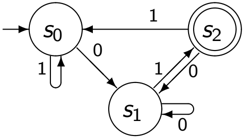
$$s_0 \xrightarrow{x_1} s_1 \xrightarrow{x_2} \dots \xrightarrow{x_{n-1}} s_{n-1} \xrightarrow{x_n} s_n$$

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

Informally. Run the automaton from the starting state, move states according to the individual letters of the word, and accept iff you end up in a final state.

Example 1

As a diagram.



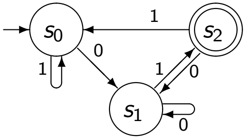
In Mathematical Notation.

- Alphabet –
- States –
- Initial state –
- Final states –
- Transition function (as a table) – on the right:

	0	1
s_0		
s_1		
s_2		

Example 1

As a diagram.



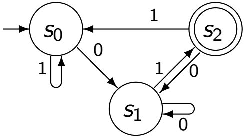
In Mathematical Notation.

- Alphabet – $\{0, 1\}$
- States – $\{s_0, s_1, s_2\}$
- Initial state – s_0 (shown with arrow w/out origin)
- Final states – $\{s_2\}$
- Transition function (as a table) – on the right:

	0	1
s_0	s_1	s_0
s_1	s_1	s_2
s_2	s_1	s_0

Example 1

As a diagram.



In Mathematical Notation.

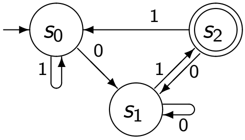
- Alphabet – $\{0, 1\}$
- States – $\{s_0, s_1, s_2\}$
- Initial state – s_0 (shown with arrow w/out origin)
- Final states – $\{s_2\}$
- Transition function (as a table) – on the right:

	0	1
s_0	s_1	s_0
s_1	s_1	s_2
s_2	s_1	s_0

Q1. Which strings are accepted by this automaton?

Example 1

As a diagram.



In Mathematical Notation.

- Alphabet – $\{0, 1\}$
- States – $\{s_0, s_1, s_2\}$
- Initial state – s_0 (shown with arrow w/out origin)
- Final states – $\{s_2\}$
- Transition function (as a table) – on the right:

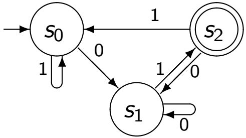
	0	1
s_0	s_1	s_0
s_1	s_1	s_2
s_2	s_1	s_0

Q1. Which strings are accepted by this automaton?
Q2. What changes if we re-name the states?

Example 1, cont'd

Recall. $N : S \times \Sigma \to S$ is the transition function.

	0	1
s_0	s_1	s_0
s_1	s_1	s_2
s_2	s_1	s_0



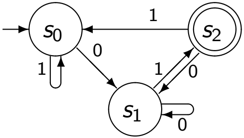
Single Steps of the automaton

- $N(s_0, 0)$ is the state that the automation transitions to from state s_0 reading letter 0. Here: $N(s_0, 0) = s_1$.

Example 1, cont'd

Recall. $N : S \times \Sigma \to S$ is the transition function.

	0	1
s_0	s_1	s_0
s_1	s_1	s_2
s_2	s_1	s_0



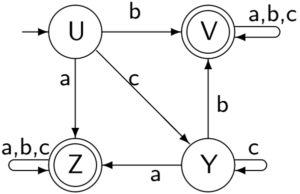
Single Steps of the automaton

- $N(s_0, 0)$ is the state that the automation transitions to from state s_0 reading letter 0. Here: $N(s_0, 0) \; = \; s_1$.

Multiple Steps of the automaton

- $N(N(s_0, 0),\ 1)$ is the state of the automation when starting in s_0 and reading first 0, then 1. Here: $N(N(s_0, 0),\ 1) = s_2$.
- Later, we will simplify this using the *Eventual State Function*. Then, $N^*(s_0, 01) = s_2$.

Example 2



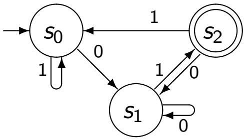
	a	b	c
$\rightarrow$ U	Z	V	Y
$\odot$ V	V	V	V
Y	Z	V	Y
$\odot$ Z	Z	Z	Z

(the table carries the same information as the diagram)

Q. What is the language of this automaton?

Eventual State Function

Revisit example 1:



- Input 0101 takes the DFA from s_0 to s_2,
  Input 1011 takes the DFA from s_1 to s_0, etc.
- A complete list of such possibilities is a function from a given state and a string to an 'eventual state.'

This is the idea of ***Eventual State Function.***
(Called N^* rather than N to reflect the transitive closure.)

Eventual State Function — Definition

Definition.
Let A be a DFA with states S, alphabet Σ, and transition function N.
The *eventual state function* for A is of type

$$N^* : S \times \Sigma^* \to S$$

and is defined inductively by:

$$N^*(s, \epsilon) = s \quad \text{(N1)}$$
$$N^*(s, x\alpha) = N^*(N(s, x), \alpha) \quad \text{(N2)}$$

(where $x \in \Sigma$)

Informally.
$N^*(s, w)$ is the state A reached by starting in state s and reading string w.

An Important (but Unsurprising) Theorem about N^*

The "Append Theorem". For all states $s \in S$ and for all strings $\alpha, \beta \in \Sigma^*$

$$N^*(s, \alpha\beta) \;=\; N^*(N^*(s, \alpha), \beta)$$

Informally: In other words, appending string β to a string α can be understood as first reaching the state after processing α, and then continuing from there to process β. (Instead of processing $\alpha\beta$ directly.)

An Important (but Unsurprising) Theorem about N^*

The "Append Theorem". For all states $s \in S$ and for all strings $\alpha, \beta \in \Sigma^*$

$$N^*(s, \alpha\beta) \; = \; N^*(N^*(s, \alpha), \beta)$$

Informally: In other words, appending string β to a string α can be understood as first reaching the state after processing α, and then continuing from there to process β. (Instead of processing $\alpha\beta$ directly.)

Proof by induction on the length of α (and arbitrary β).
Base case: $\alpha = \epsilon$

$$\begin{aligned} \text{LHS} &= N^*(s, \epsilon\beta) = N^*(s, \beta) \\ \text{RHS} &= N^*(N^*(s, \epsilon), \beta) \\ &= N^*(s, \beta) = \text{LHS} && \text{(by (N1))} \end{aligned}$$

Proof cont'd: Step case:

Recall that we want to show:
For all $s \in S$ and all $\alpha, \beta \in \Sigma^*$ holds $N^*(s, \alpha\beta) = N^*(N^*(s, \alpha), \beta)$

Step Case. Show that $N^*(s, (x\alpha)\beta) = N^*(N^*(s, x\alpha), \beta)$

Proof cont'd: Step case:

Recall that we want to show:
For all $s \in S$ and all $\alpha, \beta \in \Sigma^*$ holds $N^*(s, \alpha\beta) = N^*(N^*(s, \alpha), \beta)$

Step Case. Show that $N^*(s, (x\alpha)\beta) = N^*(N^*(s, x\alpha), \beta)$

$$\mathrm{LHS} = N^*(s, (x\alpha)\beta)$$

Proof cont'd: Step case:

Recall that we want to show:
For all $s \in S$ and all $\alpha, \beta \in \Sigma^*$ holds $N^*(s, \alpha\beta) = N^*(N^*(s, \alpha), \beta)$

Step Case. Show that $N^*(s, (x\alpha)\beta) = N^*(N^*(s, x\alpha), \beta)$

$$\begin{aligned} \text{LHS} &= N^*(s, (x\alpha)\beta) \\ &= N^*(s, x(\alpha\beta)) \end{aligned}$$

Proof cont'd: Step case:

Recall that we want to show:

For all $s \in S$ and all $\alpha, \beta \in \Sigma^*$ holds $N^*(s, \alpha\beta) = N^*(N^*(s, \alpha), \beta)$

Step Case. Show that $N^*(s, (x\alpha)\beta) = N^*(N^*(s, x\alpha), \beta)$

$$\begin{aligned} \text{LHS} &= N^*(s, (x\alpha)\beta) \\ &= N^*(s, x(\alpha\beta)) \\ &= N^*(N(s, x), \alpha\beta) && \text{(by (N2))} \end{aligned}$$

Proof cont'd: Step case:

Recall that we want to show:
For all $s \in S$ and all $\alpha, \beta \in \Sigma^*$ holds $N^*(s, \alpha\beta) = N^*(N^*(s, \alpha), \beta)$

Step Case. Show that $N^*(s, (x\alpha)\beta) = N^*(N^*(s, x\alpha), \beta)$

$$\begin{aligned}
\mathrm{LHS} &= N^*(s, (x\alpha)\beta) \\
&= N^*(s, x(\alpha\beta)) \\
&= N^*(N(s, x), \alpha\beta) && \text{(by (N2))} \\
&= N^*(N^*(N(s, x), \alpha), \beta) && \text{(by IH)}
\end{aligned}$$

Proof cont'd: Step case:

Recall that we want to show:
For all $s \in S$ and all $\alpha, \beta \in \Sigma^*$ holds $N^*(s, \alpha\beta) = N^*(N^*(s, \alpha), \beta)$

Step Case. Show that $N^*(s, (x\alpha)\beta) = N^*(N^*(s, x\alpha), \beta)$

$$\begin{aligned} \mathrm{LHS} &= N^*(s, (x\alpha)\beta) \\ &= N^*(s, x(\alpha\beta)) \\ &= N^*(N(s, x), \alpha\beta) && \text{(by (N2))} \\ &= N^*(N^*(N(s, x), \alpha), \beta) && \text{(by IH)} \end{aligned}$$

$$\mathrm{RHS} = N^*(N^*(s, x\alpha), \beta)$$

Proof cont'd: Step case:

Recall that we want to show:
For all $s \in S$ and all $\alpha, \beta \in \Sigma^*$ holds $N^*(s, \alpha\beta) = N^*(N^*(s, \alpha), \beta)$

Step Case. Show that $N^*(s, (x\alpha)\beta) = N^*(N^*(s, x\alpha), \beta)$

$$\begin{aligned}
\text{LHS} &= N^*(s, (x\alpha)\beta) \\
&= N^*(s, x(\alpha\beta)) \\
&= N^*(N(s, x), \alpha\beta) && \text{(by (N2))} \\
&= N^*(N^*(N(s, x), \alpha), \beta) && \text{(by IH)}
\end{aligned}$$

$$\begin{aligned}
\text{RHS} &= N^*(N^*(s, x\alpha), \beta) \\
&= N^*(N^*(N(s, x), \alpha), \beta) && \text{(by (N2))}
\end{aligned}$$

Proof cont'd: Step case:

Recall that we want to show:
For all $s \in S$ and all $\alpha, \beta \in \Sigma^*$ holds $N^*(s, \alpha\beta) = N^*(N^*(s, \alpha), \beta)$

Step Case. Show that $N^*(s, (x\alpha)\beta) = N^*(N^*(s, x\alpha), \beta)$

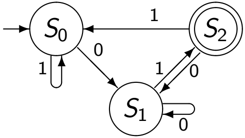
$$\begin{aligned} \text{LHS} &= N^*(s, (x\alpha)\beta) \\ &= N^*(s, x(\alpha\beta)) \\ &= N^*(N(s, x), \alpha\beta) && \text{(by (N2))} \\ &= N^*(N^*(N(s, x), \alpha), \beta) && \text{(by IH)} \end{aligned}$$

$$\begin{aligned} \text{RHS} &= N^*(N^*(s, x\alpha), \beta) \\ &= N^*(N^*(N(s, x), \alpha), \beta) && \text{(by (N2))} \end{aligned}$$

Corollary — when β is a single token

$$N^*(s, \alpha y) \; = \; N(N^*(s, \alpha), y)$$

Example



$$
\begin{aligned}
N^*(S_1, 1011) &= N^*(N(S_1, 1), 011) \\
&= N^*(S_2, 011) \\
&= N^*(S_1, 11) \\
&= N^*(S_2, 1) \\
&= N^*(S_0, \epsilon) \\
&= S_0
\end{aligned}
$$

Language of an Automaton

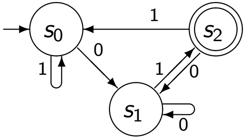
Language of an Automaton, Revisited

Recall:
The language of an automaton is the set of strings accepted by it.

Acceptance, formally:
Let $A = (\Sigma, S, s_0, F, N)$ be a DFA and w be a string in Σ^*. Then,

- w is *accepted* by A iff $N^*(s_0, w) \in F$
- Thus, $L(A) = \{w \in \Sigma^* \mid N^*(s_0, w) \in F\}$

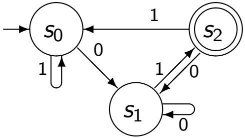
Example 1 again



Q. Which strings are accepted?

- e.g. 0011101 takes the machine from state s_0 through states s_1, s_1, s_2, s_0, s_0, s_1 to s_2 (a final state).
- $N^*(s_0, 0011101) = N^*(s_1, 011101) = N^*(s_1, 11101) = \dots N^*(s_1, 1) = s_2$
- others: 01, 001, 101, 0001, 0101, 00101101 . . .
- Thus, $L(A) = \{w \in \Sigma^* \mid w = \alpha 01, \alpha \in \Sigma^*\}$ (where A is our DFA)

Example 1 (cont'd)



Accepted Strings.

01, 001, 101, 0001, 0101, 00101101 ...

Strings that are not accepted.

ϵ, 0, 1, 00, 10, 11, 100 ...

Q. What do the accepted strings have in common? How do we justify this?

Proving an Acceptance Predicate — in General

Our Claim. The automaton A accepts precisely the strings that are elements of the language $L = \{w \in \Sigma^* \mid P(w)\}$.

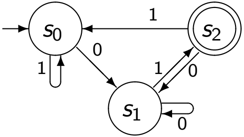
(P is sometimes called an *acceptance predicate*.)

Proof Obligations.

1. Show that any string satisfying P is accepted by A.
2. Show any string accepted by A satisfies P.

Q. Do we really need both? Isn't the first obligation enough?

Proving an Acceptance Predicate for A_1



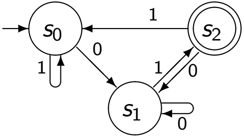
Proof obligation 1:
If a string ends in 01, then it is accepted by A_1. That is:

$$\text{For all } \alpha \in \Sigma^*,\ N^*(s_0, \alpha 01) \in F$$

Proof obligation 2:
If a string is accepted by A_1, then it ends in 01. That is:

$$\text{For all } w \in \Sigma^*,\ \text{if } N^*(s_0, w) \in F \text{ then } \ \exists \alpha \in \Sigma^*.\ \ w = \alpha 01$$

Part 1: $\forall \alpha \in \Sigma^*,\ N^*(s_0, \alpha 01) \in F$



Lemma:

$$\forall s \in S.\ N^*(s, 01) = s_2$$

Proof by cases:

$$N^*(s_0, 01) = N^*(s_1, 1) = s_2$$
$$N^*(s_1, 01) = N^*(s_1, 1) = s_2$$
$$N^*(s_2, 01) = N^*(s_1, 1) = s_2$$

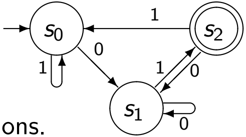
So, by the Append Theorem above, we get

$$N^*(s_0, \alpha 01) = N^*(N^*(s_0, \alpha), 01) = s_2$$

Why? Because we covered all possible cases for $N^*(s_0, \alpha)$

□

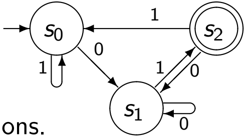
Part 2: $N^*(s_0, w) = s_2 \Rightarrow \exists\alpha.\ w = \alpha 01$



Proof. Suppose $N^*(s_0, w) = s_2$.

The shortest path in the DFA from s_0 to s_2 has two transitions.
Thus, we can assume that $w = \alpha xy$ for some $\alpha \in \Sigma^*$.
Thus, suppose $N^*(s_0, \alpha xy) = s_2$.

Part 2: $N^*(s_0, w) = s_2 \Rightarrow \exists \alpha.\ w = \alpha 01$



Proof. Suppose $N^*(s_0, w) = s_2$.

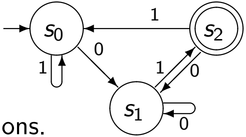
The shortest path in the DFA from s_0 to s_2 has two transitions.
Thus, we can assume that $w = \alpha xy$ for some $\alpha \in \Sigma^*$.
Thus, suppose $N^*(s_0, \alpha xy) = s_2$.

By corollary to the Append Theorem (case of single token):

$$N^*(s_0, \alpha xy) = N(N^*(s_0, \alpha x), y) = s_2$$

By the definition of N, y must be 1 and $N^*(s_0, \alpha x)$ must be s_1.
Thus, $s_2 = N^*(s_0, \alpha xy) = N(N^*(s_0, \alpha x), y) = N(s_1, 1)$, so w ends on 1.

Part 2: $N^*(s_0, w) = s_2 \Rightarrow \exists\alpha.\ w = \alpha 01$



Proof. Suppose $N^*(s_0, w) = s_2$.

The shortest path in the DFA from s_0 to s_2 has two transitions.
Thus, we can assume that $w = \alpha xy$ for some $\alpha \in \Sigma^*$.
Thus, suppose $N^*(s_0, \alpha xy) = s_2$.

By corollary to the Append Theorem (case of single token):

$$N^*(s_0, \alpha xy) = N(N^*(s_0, \alpha x), y) = s_2$$

By the definition of N, y must be 1 and $N^*(s_0, \alpha x)$ must be s_1.
Thus, $s_2 = N^*(s_0, \alpha xy) = N(N^*(s_0, \alpha x), y) = N(s_1, 1)$, so w ends on 1.

Similarly,

$$N(N^*(s_0, \alpha), x) = s_1$$

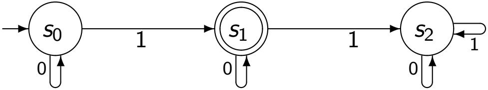
and x is 0 (again by the definition of N), so w ends on 01.

□

Another Example

Which language does D accept?

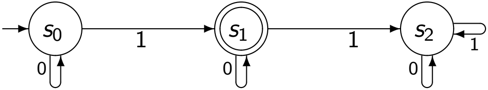
D:



Another Example

Which language does D accept?

D:

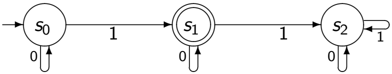


D accepts the language of bitstrings containing ***exactly one 1-bit.***

Proof obligations:

- Show that if a bitstring contains exactly one 1-bit then it is accepted by D.
- Show that if a string is accepted by D it contains exactly one 1-bit.

Mapping to Mathematics



Expressed mathematically, the main conclusion is

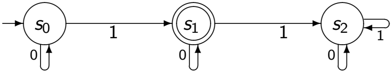
$$L(D) = \{w \in \Sigma^* \mid w = 0^n 1 0^m, n, m \geq 0\}$$

The two subgoals are

1. If $w = 0^n 1 0^m$ then $N^*(s_0, w) = s_1$.
2. If $N^*(s_0, w) = s_1$ then $w = 0^n 1 0^m$.

For this DFA the phrase "w is accepted by D" is captured by the expression $N^*(s_0, w) = s_1$.

Proving these subgoals

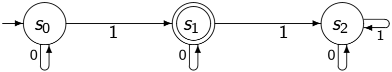


The first subgoal ("If $w = 0^n10^m$ then $N^*(s_0, w) = s_1$") can be shown using the following two lemmas, which are easily proved by induction:

$$\textbf{Lemma 1:} \quad \forall n \geq 0.\ N^*(s_0, 0^n) \ = \ s_0$$

$$\textbf{Lemma 2:} \quad \forall n \geq 0.\ N^*(s_1, 0^n) \ = \ s_1$$

Proving these subgoals



The first subgoal ("If $w = 0^n10^m$ then $N^*(s_0, w) = s_1$") can be shown using the following two lemmas, which are easily proved by induction:

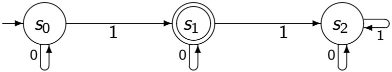
$$\textbf{Lemma 1:} \quad \forall n \geq 0.\ N^*(s_0, 0^n) = s_0$$

$$\textbf{Lemma 2:} \quad \forall n \geq 0.\ N^*(s_1, 0^n) = s_1$$

Therefore $N^*(s_0, 0^n10^m)$

$$\begin{aligned}
&= N^*(N^*(s_0, 0^n), 10^m) && \text{(by Append Theorem)} \\
&= N^*(s_0, 10^m) && \text{(by Lemma 1)} \\
&= N^*(N(s_0, 1), 0^m) && \text{(by Def. of } N^*\text{)} \\
&= N^*(s_1, 0^m) && \text{(by Def. of } N\text{)} \\
&= s_1 && \text{(by Lemma 2)}
\end{aligned}$$

Proving these subgoals



The first subgoal ("If $w = 0^n10^m$ then $N^*(s_0, w) = s_1$") can be shown using the following two lemmas, which are easily proved by induction:

$$\textbf{Lemma 1:} \quad \forall n \geq 0.\ N^*(s_0, 0^n) = s_0$$

$$\textbf{Lemma 2:} \quad \forall n \geq 0.\ N^*(s_1, 0^n) = s_1$$

Therefore $N^*(s_0, 0^n10^m)$

$$\begin{aligned}
&= N^*(N^*(s_0, 0^n), 10^m) && \text{(by Append Theorem)} \\
&= N^*(s_0, 10^m) && \text{(by Lemma 1)} \\
&= N^*(N(s_0, 1), 0^m) && \text{(by Def. of } N^*\text{)} \\
&= N^*(s_1, 0^m) && \text{(by Def. of } N\text{)} \\
&= s_1 && \text{(by Lemma 2)}
\end{aligned}$$

The second subgoal ("$N^*(s_0, w) = s_1$ then $w = 0^n10^m$"), more formally:

$$\forall w: \ N^*(s_0, w) = s_1 \implies \exists n, m \geq 0.\ w = 0^n10^m$$

can be proved in a similar fashion to Example 1 on earlier slides.

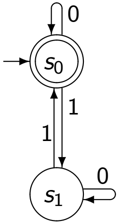
Minimisation of DFAs

Equivalence of Automata

Two automata are said to be **equivalent** if they accept the same language.

Example:

A_5:

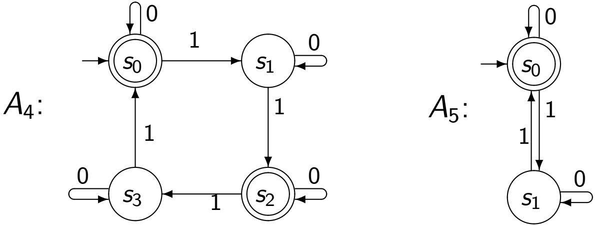


Q1. What language does A_5 accept?

Equivalence of Automata

Two automata are said to be **equivalent** if they accept the same language.

Example:



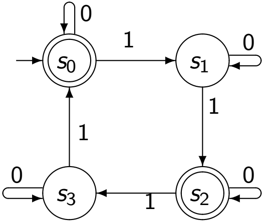
Q1. What language does A_5 accept?
Q2. What language does A_4 accept? Can A_4 be simplified?

Equivalence of Automata

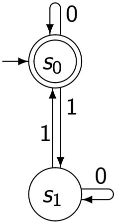
Two automata are said to be **equivalent** if they accept the same language.

Example:

A_4:



A_5:



Q1. What language does A_5 accept?
Q2. What language does A_4 accept? Can A_4 be simplified?
Q3. When are two states equivalent?

Equivalence of States

Two states s_j and s_k of an FSA are equivalent if for all input strings w

$$N^*(s_j, w) \in F \text{ if and only if } N^*(s_k, w) \in F$$

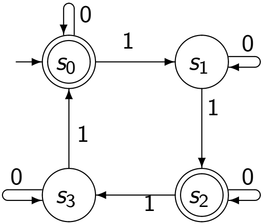
Equivalence of States

Two states s_j and s_k of an FSA are equivalent if for all input strings w

$$N^*(s_j, w) \in F \text{ if and only if } N^*(s_k, w) \in F$$

Example. In A_4, s_2 is equivalent to s_0 and s_1 is equivalent to s_3.

A_4:



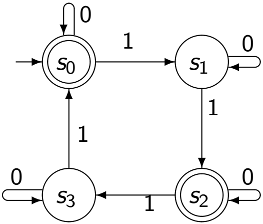
Equivalence of States

Two states s_j and s_k of an FSA are equivalent if for all input strings w

$$N^*(s_j, w) \in F \text{ if and only if } N^*(s_k, w) \in F$$

Example. In A_4, s_2 is equivalent to s_0 and s_1 is equivalent to s_3.

A_4:



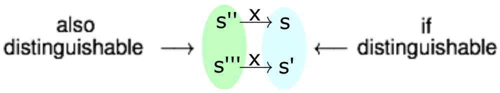
Q. Can we use this to define when/whether two DFAs are equivalent?

DFA State Minmimization

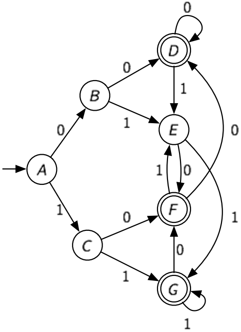
The *Table Filling Algorithm* identifies equivalent and non-equivalent (called distinguishable) pairs of states.

- Any final state can't be equivalent to a non-final state (they are distinguishable).
- If s and s' are distinguishable and there exist states s'', s''', and symbol x such that
  - $N^*(s'', x) = s$
  - $N^*(s''', x) = s'$

  then s'' and s''' are also distinguishable:

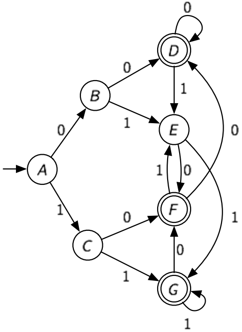


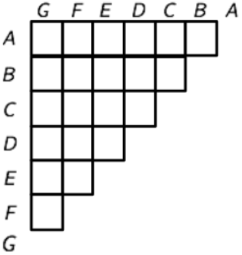
Identifying pairs of (In)distinguishable States: An Example



	G	F	E	D	C	B	A
A							
B							
C							
D							
E							
F							
G							

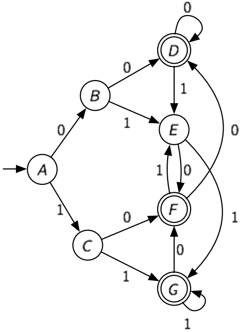
Identifying pairs of (In)distinguishable States: An Example





- Fill in $\times$ (black) whenever one component of pair is final, and other is not.

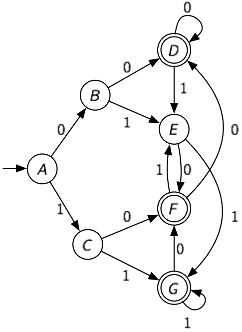
Identifying pairs of (In)distinguishable States: An Example



	G	F	E	D	C	B	A
A							
B							
C							
D							
E							
F							
G							

- Fill in × (black) whenever one component of pair is final, and other is not.
- Fill in × (blue) if 1 moves the pair of states to a distinguishable pair

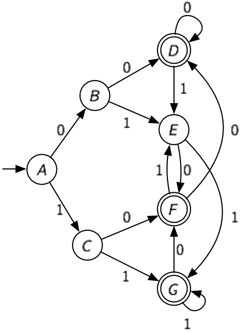
Identifying pairs of (In)distinguishable States: An Example



	G	F	E	D	C	B	A
A							
B							
C							
D							
E							
F							
G							

- Fill in × (black) whenever one component of pair is final, and other is not.
- Fill in × (blue) if 1 moves the pair of states to a distinguishable pair
- Fill in × (red) if 0 moves the pair of states to a distinguishable pair

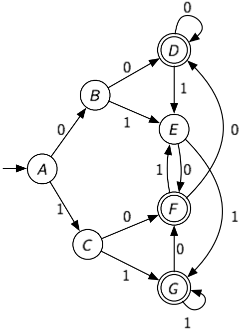
Identifying pairs of (In)distinguishable States: An Example



	G	F	E	D	C	B	A
A							
B							
C							
D							
E							
F							
G							

- Fill in × (black) whenever one component of pair is final, and other is not.
- Fill in × (blue) if 1 moves the pair of states to a distinguishable pair
- Fill in × (red) if 0 moves the pair of states to a distinguishable pair
- Repeat until no progress (any two states without a × sign are equivalent!)

Identifying pairs of (In)distinguishable States: An Example



	G	F	E	D	C	B	A
A	×	×	× (blue)	×	× (blue)	× (red)	
B	×	×	× (blue)	×	× (blue)		
C	×	×		×			
D	× (blue)		×				
E	×	×					
F	× (blue)						
G							

- Fill in × (black) whenever one component of pair is final, and other is not.
- Fill in × (blue) if 1 moves the pair of states to a distinguishable pair
- Fill in × (red) if 0 moves the pair of states to a distinguishable pair
- Repeat until no progress (any two states without a × sign are equivalent!)

Table-filling Algorithm

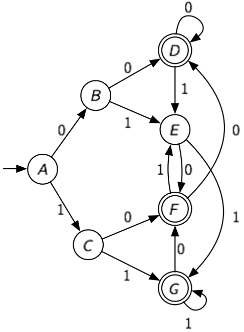
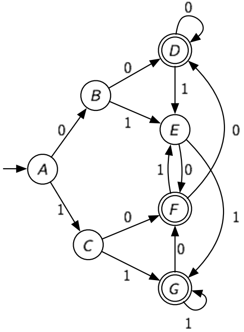
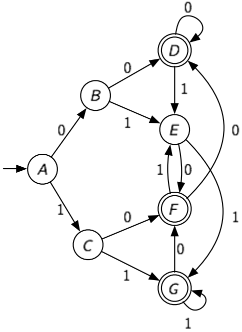


Table-filling Algorithm



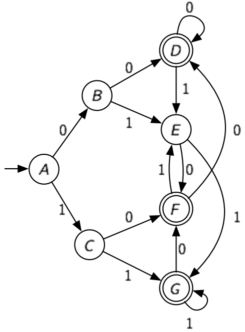
- Delete states not reachable from start states.

Table-filling Algorithm



- Delete states not reachable from start states.
- Find distinguishable and equivalent pairs of states as described on the previous slide.

Table-filling Algorithm

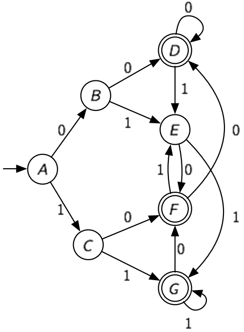


- Delete states not reachable from start states.
- Find distinguishable and equivalent pairs of states as described on the previous slide.
- Identify equivalence classes of equivalent states. In this example: $\{A\},\{B\},\{C,E\},\{D,F\},\{G\}$

	G	F	E	D	C	B	A
A	×	×	×	×	×	×	
B	×	×	×	×	×		
C	×	×		×			
D	×		×				
E	×	×					
F	×						
G							

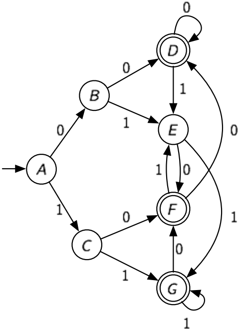
Color-blind?
red: A/B,
blue: A/E, A/C, B/E, B/C, D/G, F/G

Table-filling Algorithm



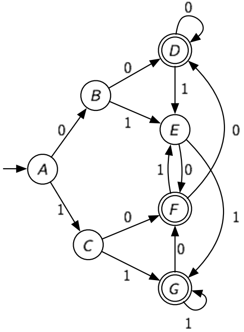
- Delete states not reachable from start states.
- Find distinguishable and equivalent pairs of states as described on the previous slide.
- Identify equivalence classes of equivalent states. In this example: $\{A\}, \{B\}, \{C, E\}, \{D, F\}, \{G\}$
- Collapse each equivalence class of states to a single state.

Table-filling Algorithm



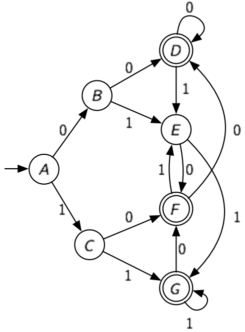
- Delete states not reachable from start states.
- Find distinguishable and equivalent pairs of states as described on the previous slide.
- Identify equivalence classes of equivalent states. In this example: $\{A\}, \{B\}, \{C, E\}, \{D, F\}, \{G\}$
- Collapse each equivalence class of states to a single state.
- Delete parallel transitions with same label.

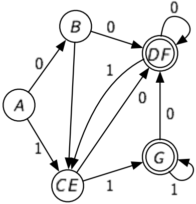
Table-filling Algorithm



- Delete states not reachable from start states.
- Find distinguishable and equivalent pairs of states as described on the previous slide.
- Identify equivalence classes of equivalent states. In this example: $\{A\}, \{B\}, \{C, E\}, \{D, F\}, \{G\}$
- Collapse each equivalence class of states to a single state.
- Delete parallel transitions with same label.
- The resulting transition diagram will be a DFA.

Table-filling Algorithm





- Delete states not reachable from start states.
- Find distinguishable and equivalent pairs of states as described on the previous slide.
- Identify equivalence classes of equivalent states. In this example: $\{A\}, \{B\}, \{C, E\}, \{D, F\}, \{G\}$
- Collapse each equivalence class of states to a single state.
- Delete parallel transitions with same label.
- The resulting transition diagram will be a DFA.

Table-filling Algorithm: Improvement

Homework / Open Discussion:

- Are we allowed to delete states that cannot reach any final state?

Table-filling Algorithm: Improvement

Homework / Open Discussion:

- Are we allowed to delete states that cannot reach any final state?
- Construct at least two examples and execute the algorithm (both with and without deleting such states):

Table-filling Algorithm: Improvement

Homework / Open Discussion:

- Are we allowed to delete states that cannot reach any final state?
- Construct at least two examples and execute the algorithm (both with and without deleting such states):
  - One DFA D has exactly 2 states and $L(D) = \{1^n \mid n \geq 0\}$.
    Q. Can there be a DFA with just 1 state?

Table-filling Algorithm: Improvement

Homework / Open Discussion:

- Are we allowed to delete states that cannot reach any final state?
- Construct at least two examples and execute the algorithm (both with and without deleting such states):
  - One DFA D has exactly 2 states and $L(D) = \{1^n \mid n \geq 0\}$.
    Q. Can there be a DFA with just 1 state?
  - Another DFA D' that has exactly 4 states but still $L(D) = L(D')$.

Table-filling: Other Uses

- Test equivalence of languages accepted by 2 DFAs.

Table-filling: Other Uses

- Test equivalence of languages accepted by 2 DFAs.
  - Given $A = (\Sigma, S_A, s_{A0}, F_A, N_A)$ and $B = (\Sigma, S_B, s_{B0}, F_B, N_B)$:

Table-filling: Other Uses

- Test equivalence of languages accepted by 2 DFAs.
  - Given $A = (\Sigma, S_A, s_{A0}, F_A, N_A)$ and $B = (\Sigma, S_B, s_{B0}, F_B, N_B)$:
    - Rename states in S_B so that S_A and S_B are disjoint.

Table-filling: Other Uses

- Test equivalence of languages accepted by 2 DFAs.
  - Given $A = (\Sigma, S_A, s_{A0}, F_A, N_A)$ and $B = (\Sigma, S_B, s_{B0}, F_B, N_B)$:
    - Rename states in S_B so that S_A and S_B are disjoint.
    - View A and B together as one DFA
      (Ignore the fact that there are two start states)

Table-filling: Other Uses

- Test equivalence of languages accepted by 2 DFAs.
  - Given $A = (\Sigma, S_A, s_{A0}, F_A, N_A)$ and $B = (\Sigma, S_B, s_{B0}, F_B, N_B)$:
    - Rename states in S_B so that S_A and S_B are disjoint.
    - View A and B together as one DFA
      (Ignore the fact that there are two start states)
    - Run table-filling on $S_A \cup S_B$.

Table-filling: Other Uses

- Test equivalence of languages accepted by 2 DFAs.
  - Given $A = (\Sigma, S_A, s_{A0}, F_A, N_A)$ and $B = (\Sigma, S_B, s_{B0}, F_B, N_B)$:
    - Rename states in S_B so that S_A and S_B are disjoint.
    - View A and B together as one DFA
      (Ignore the fact that there are two start states)
    - Run table-filling on $S_A \cup S_B$.
    - s_{A0} and s_{B0} are indistinguishable $\Leftrightarrow L(A) = L(B)$.Why?
      If w distinguishes s_{A0} from s_{B0} then w cannot be in both $L(A)$ and $L(B)$

Table-filling: Other Uses

- Test equivalence of languages accepted by 2 DFAs.
  - Given $A = (\Sigma, S_A, s_{A0}, F_A, N_A)$ and $B = (\Sigma, S_B, s_{B0}, F_B, N_B)$:
    - Rename states in S_B so that S_A and S_B are disjoint.
    - View A and B together as one DFA
      (Ignore the fact that there are two start states)
    - Run table-filling on $S_A \cup S_B$.
    - s_{A0} and s_{B0} are indistinguishable $\Leftrightarrow L(A) = L(B)$.Why?
      If w distinguishes s_{A0} from s_{B0} then w cannot be in both $L(A)$ and $L(B)$

Table-filling: Other Uses

- Test equivalence of languages accepted by 2 DFAs.
  - Given $A = (\Sigma, S_A, s_{A0}, F_A, N_A)$ and $B = (\Sigma, S_B, s_{B0}, F_B, N_B)$:
    - Rename states in S_B so that S_A and S_B are disjoint.
    - View A and B together as one DFA
      (Ignore the fact that there are two start states)
    - Run table-filling on $S_A \cup S_B$.
    - s_{A0} and s_{B0} are indistinguishable $\Leftrightarrow L(A) = L(B)$.Why?
      If w distinguishes s_{A0} from s_{B0} then w cannot be in both $L(A)$ and $L(B)$
- Minimality test: Suppose a DFA A cannot be minimised further by table-filling. Then, A has the least number of states among all DFAs that accept $L(A)$

Limitations of FSAs

What is not acceptable for an FSA?

Q. Is an FSA a sufficiently "powerful" model of computation?

- E.g., $L =$ All syntactically valid Java programs
- or $L =$ All Java (etc.) programs that terminate
- Can DFAs always return the correct yes/no answer?

What is not acceptable for an FSA?

Q. Is an FSA a sufficiently “powerful” model of computation?

- E.g., $L =$ All syntactically valid Java programs
- or $L =$ All Java (etc.) programs that terminate
- Can DFAs always return the correct yes/no answer?

Technical Analysis. Properties of languages accepted by a DFA.

What is not acceptable for an FSA?

Q. Is an FSA a sufficiently "powerful" model of computation?

- E.g., $L =$ All syntactically valid Java programs
- or $L =$ All Java (etc.) programs that terminate
- Can DFAs always return the correct yes/no answer?

Technical Analysis. Properties of languages accepted by a DFA.

A very important example:

- $L = \{ a^n b^n \mid n \in \mathbb{N}_0\}$
- $L = \{\epsilon, ab, aabb, aaabbb, a^4 b^4, a^5 b^5, ...\}$

Claim. There is no FSA that recognises this language.

What is not acceptable for an FSA?

Q. Is an FSA a sufficiently "powerful" model of computation?

- E.g., $L =$ All syntactically valid Java programs
- or $L =$ All Java (etc.) programs that terminate
- Can DFAs always return the correct yes/no answer?

Technical Analysis. Properties of languages accepted by a DFA.

A very important example:

- $L \ = \ \{ \ a^n b^n \mid n \in \mathbb{N}_0 \}$
- $L \ = \ \{\epsilon, ab, aabb, aaabbb, a^4 b^4, a^5 b^5, \ldots\}$

Claim. There is no FSA that recognises this language.
(because an FSA's memory is limited.)

Q. How do you answer the question above now?

Proof of Claim

Proof by contradiction.
Suppose A is an FSA that accepts L. That is, $L = L(A)$.

Then each of the following are states of A:

$$N^*(s_0, a),\ N^*(s_0, a^2),\ N^*(s_0, a^3)\ \ldots$$

But A only has finitely many states, so some state must repeat:

There are distinct i and j such that $N^*(s_0, a^i) \ = \ N^*(s_0, a^j)$.

- That is, the automaton *cannot* tell those a^i and a^j apart.
- But then how can it tell whether b^i or b^j follow?

Proof by Contradiction (cont'd)

Since $a^i b^i$ is accepted, we know

$$N^*(s_0, a^i b^i) \in F$$

By the *append theorem*

$$N^*(N^*(s_0, a^i), b^i) \;=\; N^*(s_0, a^i b^i) \in F$$

Now, since $N^*(s_0, a^i) = N^*(s_0, a^j)$

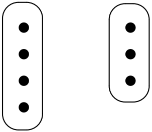
$$N^*(N^*(s_0, a^j), b^i) \;=\; N^*(s_0, a^j b^i) \in F$$

So $a^j b^i$ is accepted by A but $a^j b^i$ is not in L, contradicting the initial assumption.

Pigeonhole Principle

The proof used the ***pigeonhole principle***:

> ***No function from one set to a smaller finite set can be one-to-one.***



(Finiteness is not really necessary — no function from one set to another with smaller *cardinality* can be one-to-one.)

"You cannot fit $n+1$ pigeons into n holes"