

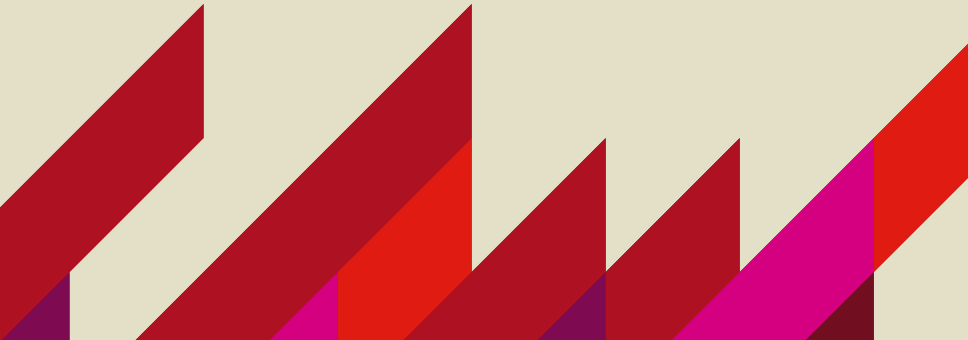
DMTH237 Discrete Mathematics II

Week 5&6: Fridays — 29 March, 5 April 2019

NON-DETERMINISTIC FINITE STATE MACHINES



Non-Deterministic Finite State Machines (NDFSMs)



Multiple Transitions

Do you notice anything **peculiar** about this **FSA**?

There **seems to be a mistake** because there are **two transitions** labelled '1' coming out of **state A**.

If we are in **state A** and we **happen to read** a '1', do we **go to state B** or to **state D**?

We **appear to have a choice**.

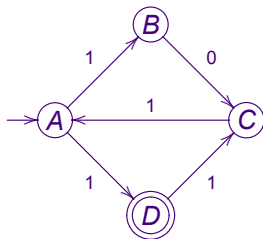
Does this mean we have to toss a coin so that **sometimes we go to state B** but on other occasions, with the **same input string**, we go to **D**?

There is no mistake.

This is an example of what is known as a **non-deterministic FSA**.

It appears to incorporate the **element of chance**.

But what possible use could there be for a **machine** whose **behaviour is unpredictable**?



NDFSA acceptance

Well, let's ask: what strings **can be accepted** by this **NDFSA**?
First we have to modify our definition of 'accepted'.

Definition (non-deterministic FSA)

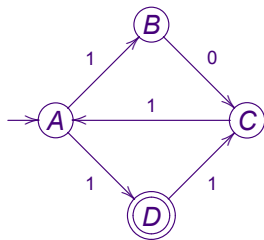
A **string** is **accepted** by a **non-deterministic FSA** if **it is possible** to **terminate** in an **accepting state**.

What can be **accepted**? $(101 + 111)^*(101 + 111)^*1(101 + 111)^*1$

1011 could terminate in **B**, or in the **black-hole**, or in **D**. Going **multiply around the top loop** accepts strings **matching regular expression**: $(101)^*1$.

1111 if it just **goes around the bottom loop**.
Any sequence of 1s **matching the regular expression** $(111)^*1$ will be **accepted**.

Going around **either loop**, many times, **in any order**: e.g., '1011111011011'.



NDFSA: easier to design

The point of considering **non-deterministic machines** is that, in many cases, they are **much easier to design** than **deterministic** ones.

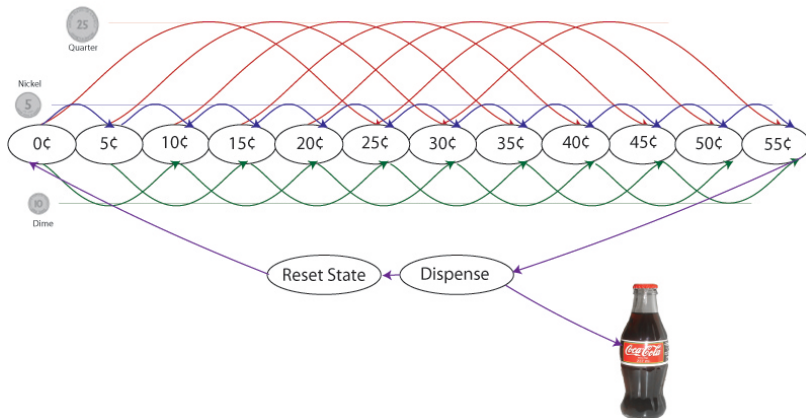
Try to construct a **deterministic FSA** to **accept** the **language** $(101 + 111)^*1$.

It is **not so easy**.

- ▶ So, the **advantage** of non-deterministic FSAs over deterministic ones is that they are **easier to design**.
- ▶ But how do we **implement** them?
 - ▶ use a **random number generator** whenever we have a choice;
 - ▶ work in parallel — follow **multiple paths simultaneously**;
 - ▶ need a method for **converting non-deterministic FSAs** into **deterministic** ones.

State changes w/o input

Finite State Machine:
Soda Machine State Diagram

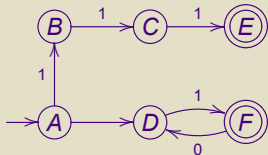


Null transitions

Having multiple transitions, for the same input, is **not the only way** an **FSA** can be **non-deterministic**. It can have **null transitions**.

Consider the follow **state diagram** for an **acceptor**.

Example



Note the **two transitions coming out of state A**. We have **not forgotten to write the input character** that produces the **transition from A to D**.

It can be made **spontaneously, without the need to read any input**. **Accepts:** $111 + (10)^*1$.

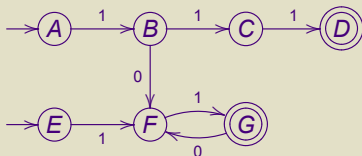
However there is **danger** in using an **arrow without a label** for a **null transition**. If we simply **forget to label an arrow** it could be **interpreted wrongly**. To avoid this, we use the **null-string** symbol ' λ ' to **represent a null transition**.

Run the **machine** on the **string '10101'**. When we **start with a null transition** ... — it is **accepted** because **it is possible** to end in an **accepting state**.

NDFSA: Multiple Initial States

This next **non-deterministic machine** offers a **choice of where to start**.

Example



It accepts the strings '111' and those such as '101', '10101', '1010101', '101010101', ... by **starting at A**.

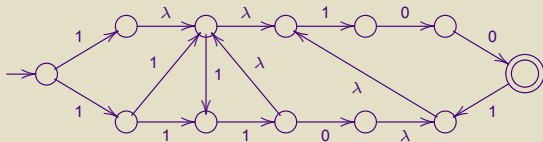
It also accepts strings such as '11', '1101', '110101', ... by **starting at E**.

The **language** accepted by this **machine** is: $111 + 10(10)^*1 + 1(10)^*1$.
 This could be written using more parentheses as: $1(1 + (\lambda + 0)(10)^*)1$,
 which while shorter is **not necessarily any clearer**.
 This is even shorter and remains quite clear: $111 + (10 + 1)(10)^*1$.

Completing the Nulls

In the following **NDFSA** it is possible to move **spontaneously** along several **null transitions one after the other**.

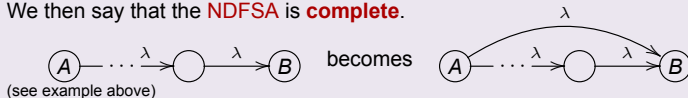
Example



Definition (completing the nulls)

If there is a **sequence of null transitions** from **state A** to **state B** add a **direct null transition** from **A** to **B** (unless there is already a **direct null** from **A** to **B**).

We then say that the **NDFSA** is **complete**.



scanning for nulls

Working with a **transition table** is **much more reliable** than using a **state diagram** as it is not so easy to **miss a null** in a **table**.
We use the same system to **complete the nulls** as to find **accessible states**.

Example (scanning for nulls)

	0	1	λ
A	B	A	BD
B	F	C	F
C		F	
D	AC		A
E		ABC	
F	C		BE

- ▶ A: which has **null transitions**; so ...
- ▶ A, B, D: ... write down its **nulls**.
- ▶ A, B, D: Look at next **state** in the list; it has **null** to **F**
- ▶ A, B, D, F: ... so add **F** to the list.
- ▶ A, B, D, F: Next **state**, has **null** to **A**, already listed;
- ▶ A, B, D, F, E: **F** adds **E** to the list.
- ▶ A, B, D, F, E: no more to add.

Hence there can be **null transitions** from **state A** to all of **BDEF**. Similarly from **B** to **EF**, and from **D** to **ABEF**, and from **F** to **BE**, as already stated. This **completes** the λ -column.

Removing the Nulls

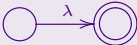
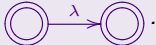
Suppose we have a **complete NDFSA**.
We now **remove** any **null transitions** as follows.

Procedure (removing the nulls)

If there is a **null transition** from **A** to **B** and a **non-null transition** from **B** to **C** then **add a non-null transition** from **A** to **C** associated with the **same input character** as the original **non-null transition**.

In **state diagrams**:  becomes 

and similarly  becomes 

while for **accepting states**  becomes .

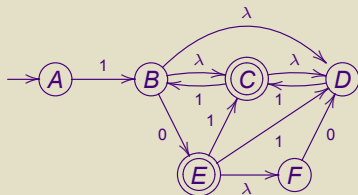
With an **initial state**  becomes .

Having done this, each **null transition** is now **completely redundant**, so can be **simply removed**.

removing nulls, example

Consider the following **complete NDFSA**.

Example (removing nulls)



→ A

	0	1	λ
A		BCD	
B	EF		CD
C		BCD	D
D		CD	
E		CD	F
F	D		

→ A

	0	1
A		B
B	E	BC
C		BC
D		C
E	D	CD
F	D	

In the **transition table**, **augment** any **state** names that can **λ-transition** by **including their target states**. Now when in **state B** reading a '1' we could use the '1' directly from **B** (though in this case there is no appropriate **1-transition**) or first move by a **λ** to either **C** or **D**, and then use the '1'. Take the **union of sets** in the '1' column for the **B**, **C** and **D** rows.

This is: $\emptyset \cup \{B, C, D\} \cup \{C, D\} = \{B, C, D\}$, which we write as **BCD**.

Similarly **state C** can **λ-transition** to **D**, so take the **union** of rows **C** and **D**.

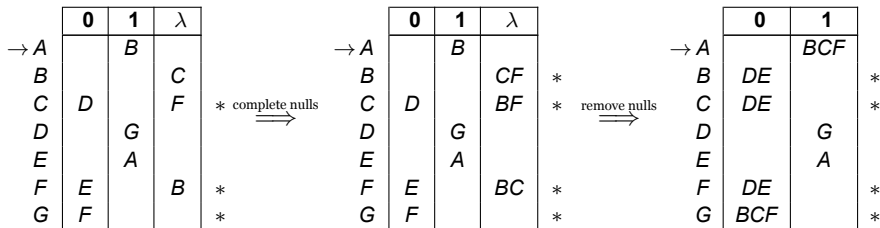
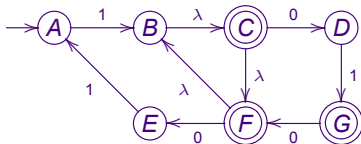
Similarly **state E** can **λ-transition** to **F**, so take the **union** of rows **E** and **F**.

Row **A** is unchanged, as are rows **D** and **F**.

The resulting **NDFSA** has no **nulls**, which have all been replaced by **multiple transitions**.

incomplete NDFSA

Consider this **incomplete NDFSA**.



Note how **states B, C and F** can all **null-transition** to each other, and that **B, C** become **accepting** because they can **null-transition** to an **accepting state**. Upon **removing nulls** these three **states** all **transition equivalently**.

We now have **multiple transitions**. Next we'll see **how to combine** these.

Combining Multiple Transitions

Suppose now we have a **non-deterministic FSA**, M , with all its **null transitions** removed, but having **multiple transitions** from a **state** for the same **input character**.

We transform this **non-deterministic machine** M into a **deterministic** one (we shall call this M^Δ , with the Greek delta representing 'deterministic') by considering **sets of states** as the **states** of the **deterministic machine**.

Definition (deterministic machine M^Δ)

If an **NDFSA** M has a **set of states** S , the new **machine** M^Δ is defined to have

- ▶ $\wp(S)$, the **power set** of S , as its **set of states**;
- ▶ if X is a **subset** of S and c is an **input character**, then $M^\Delta(X, c)$ is defined to be the **subset** of all **states** in S that **can be reached** from some **state** in X by **reading the character** c .
In other words: $M^\Delta(X, c) = \bigcup_{x \in X} M(x, c) \in \wp(S)$.
- ▶ A **subset** $X \in \wp(S)$ is **accepting** in M^Δ if there is $x \in X$ such that x is **accepting** in M .

The resulting **FSA**, M^Δ , will clearly be **deterministic** because from each set of **states** of the original **machine** M there is **exactly one subset** of **states** of M that we can reach for a given **input character**. So we will now have a **deterministic FSA** that is **equivalent** to M , in the sense that it **accepts precisely the same language** as M .

Convert NDFSA to FSA

Convert the following **non-deterministic FSA** to a **deterministic** one:

Example (NDFSA $\rightarrow$ FSA)

	<table><tr><th>0</th><th>1</th></tr><tr><td>$\rightarrow A$ <i>CD</i></td><td><i>D</i></td></tr><tr><td><i>B</i> <i>AD</i></td><td><i>B</i></td></tr><tr><td><i>C</i> <i>AC</i></td><td><i>C</i></td></tr><tr><td><i>D</i> <i>A</i></td><td></td></tr></table>	0	1	$\rightarrow A$ <i>CD</i>	<i>D</i>	<i>B</i> <i>AD</i>	<i>B</i>	<i>C</i> <i>AC</i>	<i>C</i>	<i>D</i> <i>A</i>		*						
0	1																	
$\rightarrow A$ <i>CD</i>	<i>D</i>																	
<i>B</i> <i>AD</i>	<i>B</i>																	
<i>C</i> <i>AC</i>	<i>C</i>																	
<i>D</i> <i>A</i>																		
$\rightarrow 0$	<table><tr><th>0</th><th>1</th></tr><tr><td>1</td><td>2</td></tr><tr><td>1</td><td>3</td></tr><tr><td>2</td><td>0</td></tr><tr><td>3</td><td>6</td></tr><tr><td>4</td><td>3</td></tr><tr><td>5</td><td>5</td></tr><tr><td>6</td><td>6</td></tr></table>	0	1	1	2	1	3	2	0	3	6	4	3	5	5	6	6	<i>A</i> <i>CD</i> <i>D</i> <i>AC</i> <i>C</i> $\emptyset$ <i>ACD</i>
0	1																	
1	2																	
1	3																	
2	0																	
3	6																	
4	3																	
5	5																	
6	6																	
$\rightarrow 0$	<table><tr><th>0</th><th>1</th></tr><tr><td>1</td><td>2</td></tr><tr><td>1</td><td>3</td></tr><tr><td>2</td><td>0</td></tr><tr><td>3</td><td>6</td></tr><tr><td>4</td><td>3</td></tr><tr><td>5</td><td>5</td></tr><tr><td>6</td><td>6</td></tr></table>	0	1	1	2	1	3	2	0	3	6	4	3	5	5	6	6	* <i>A</i> <i>CD</i> <i>D</i> * <i>AC</i> <i>C</i> $\emptyset$ * <i>ACD</i>
0	1																	
1	2																	
1	3																	
2	0																	
3	6																	
4	3																	
5	5																	
6	6																	

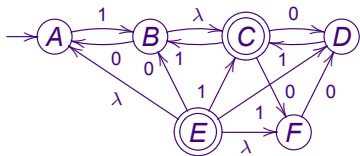
We don't need all $2^4 = 16$ **subsets** in the **power set**. Just identify the **subsets** that **we actually need**, starting from the **initial state**. Fill in the table, **numbering consecutively** the **subsets of states** that can be reached. The **empty set** corresponds to a 'black-hole'. When the table is complete identify the **accepting states** as those **subsets** including an **accepting state** in the **NDFSA**.

Summary of conversion NDFSA $\rightarrow$ FSA

Given the **state diagram** for a **non-deterministic FSA** there are six stages in **converting it** into a **reduced deterministic FSA** presented in **standard form**.

1. construct the **state table**
2. complete the **nulls**
3. remove the **nulls**
4. remove **multiple transitions**
5. combine **equivalent states**
6. put into **standard form**

A worked example



→ A

	0	1	λ
A		B	
B	A		C
C	DF	B	
D		C	
E	B	CD	AF
F	D		

→ A

	0	1
A		B
B	ADF	B
C	DF	B
D		C
E	BD	BCD
F	D	

→ 0

	0	1
1	1	2
1	1	1
2	3	2
3	4	5
4	1	6
5	3	2
6	7	2
7	4	6

A
 ∅
 * B
 ADF
 D
 * BC
 * C
 DF

	0	1	≡ ₀	0	1	≡ ₁	0	1	≡ ₂	0	1	≡ ₃
0	1	2	0	0	1	0	1	2	0	1	2	0
1	1	1	0	0	0	1	1	1	1	1	1	1
2	3	2	1	0	1	2	0	2	2	3	2	2
3	4	5	0	0	1	0	0	2	3	0	2	3
4	1	6	0	0	1	0	1	2	0	1	2	0
5	3	2	1	0	1	2	0	2	2	3	2	2
6	7	2	1	0	1	2	0	2	2	3	2	2
7	4	6	0	0	1	0	0	2	3	0	2	3

Reduced

→ 0

	0	1
1	1	2
1	1	1
2	3	2
3	0	2

*

1. construct **state table**; 2. **complete the nulls** — nothing to do; 3. **remove the nulls**;
4. **remove multiple transitions**; 5. **combine multiple transitions**;
6. put into **standard form** — nothing to do.

Index of 'jargon' terms

- 1-transition, 12
- λ -transition, 12
- accept, 5
- accepted, 4, 7
- accepting, 13, 14
- accepting state, 4, 7, 13, 15
- accepting states, 11, 15
- acceptor, 7
- accepts, 4
- accessible states, 10
- arrow, 7
- black-hole, 4, 15
- complete NDFSA, 11, 12
- complete the nulls, 10, 17
- completes, 10
- deterministic, 5, 14, 15
- deterministic FSA, 5, 14, 16
- deterministic machine, 14
- empty set, 15
- equivalent, 14
- equivalent states, 16
- FSA, 3, 7, 14
- incomplete NDFSA, 13
- initial state, 11, 15
- input, 7
- input character, 7, 11, 14
- input string, 3
- language, 5, 8, 14
- machine, 3, 7, 8, 14
- multiple transitions, 12–14, 16, 17
- NDFSA, 4, 9, 12, 14, 15
- non-deterministic, 7
- non-deterministic FSA, 4, 14–16
- non-deterministic FSAs, 5
- non-deterministic machine, 8, 14
- non-deterministic machines, 5
- null, 9, 10
- null transition, 7, 9, 11
- null transitions, 9–11, 14
- null-string, 7
- null-transition, 13
- nulls, 10, 12, 16
- power set, 14, 15
- reduced, 16
- regular expression, 4
- remove multiple transitions, 17
- remove the nulls, 17
- removing nulls, 13
- set, 14
- set of states, 14
- sets, 12
- standard form, 16, 17
- state, 3, 7, 9, 10, 12, 14
- state diagram, 7, 10, 16
- state diagrams, 11
- state table, 16, 17
- states, 12–14
- string, 4, 7
- subset, 14
- subsets, 15
- subsets of states, 15
- table, 10
- terminate, 4
- transition, 7, 11
- transition equivalently, 13
- transition table, 10, 12
- transitions, 3, 7
- union, 12