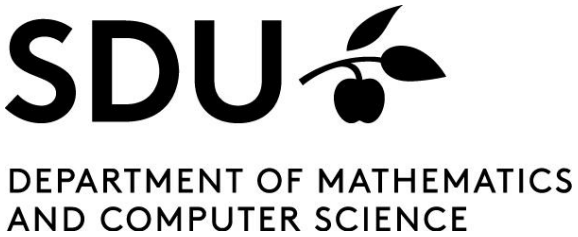
Models of Computation, Languages, and Recursion

Kevin Schewior

Website: https://sites.google.com/view/kschewior/home

Slides largely by Fabrizio Montesi



Objectives

- We will use abstract models to solve some common computational problems related to languages.
- You will encounter these topics again, for example in the following courses:
  - Compiler Construction (5th semester)
  - Complexity and Computability (6th semester)
  - Concurrency Theory (4th-year elective)

Models of Computation

- Programming is essentially precisely formulating a method that a computer can run.
- Achieving such precision can be challenging if you start from scratch.
- Researchers have developed (and still develop) a toolbox of *models* that you can use to formulate solutions.

Languages

- Algorithms are concretely defined using programming languages.
- Computers first recognise whether programs are valid in their syntax and then execute their intended semantics (meaning).
- Languages in general represent a big field of study in Computer Science and are largely applied in the industry.

Recursion

- If a definition or a statement includes a self-reference, we call it recursive.
- Example:
  - A sentence may be the connection of two sentences.
  - An expression can be the sum of two expressions.
- Recursion is pervasive in Math, CS, language, art, …

Examples of problems

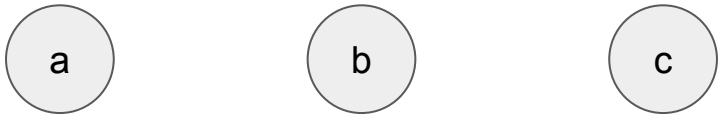
Recall: a string is a sequence of characters (may contain spaces).

- Given a string s, determine if it is a valid first name.
- Given a string s, determine if it is a valid e-mail address.
- Given a string s, determine if it is a valid full name.

Deterministic Finite Automaton (DFA)
An informal introduction, with an example

A DFA consists of a finite set of states.

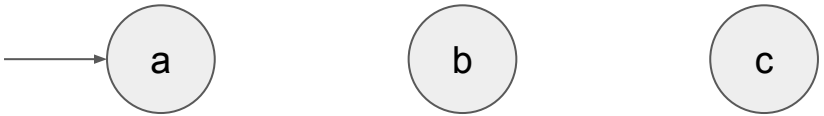
Example:



Deterministic Finite Automaton (DFA)
An informal introduction, with an example

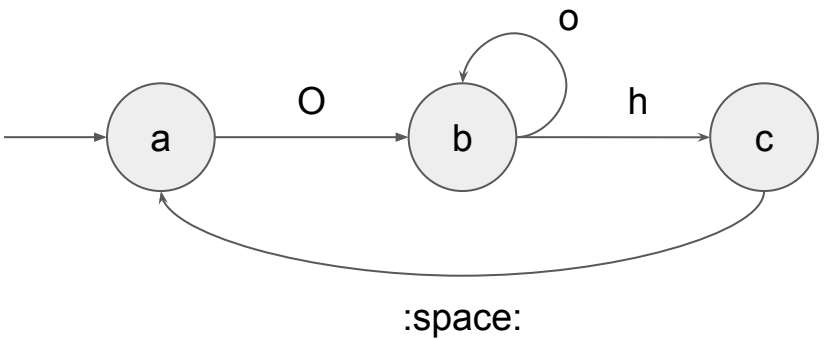
One of these states is the **start state**.

Example (a is the start state):



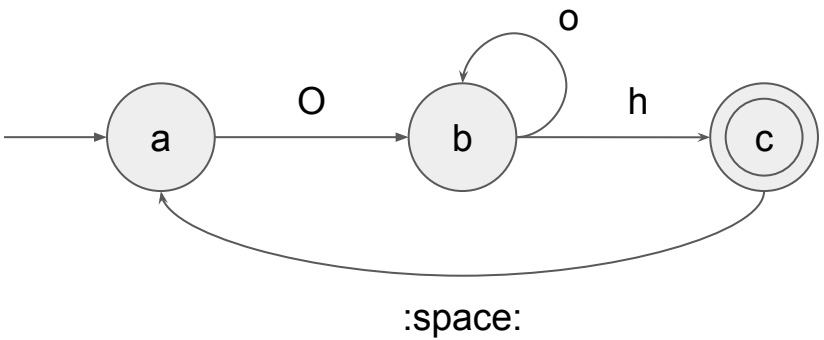
Deterministic Finite Automaton (DFA)
An informal introduction, with an example

States are connected by transition arrows, which are labelled with a character.
Example:



Deterministic Finite Automaton (DFA)
An informal introduction, with an example

Some states are accept states (denoted with a double circle).
Example (c is an accept state):



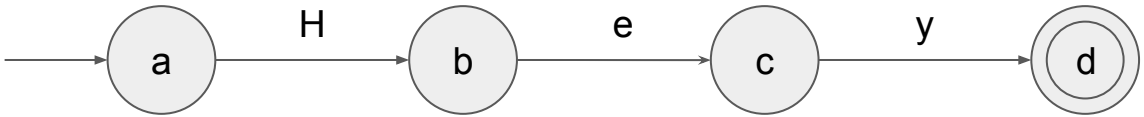
Semantics of DFAs

What does a DFA do?
It recognises if a string is member of a language.

The idea is: take your string, and follow the transitions of the DFA character by character. When you reach the end of your string, check if you are in an accept state. If you are, then the string is a member of the language.

Semantics of DFAs

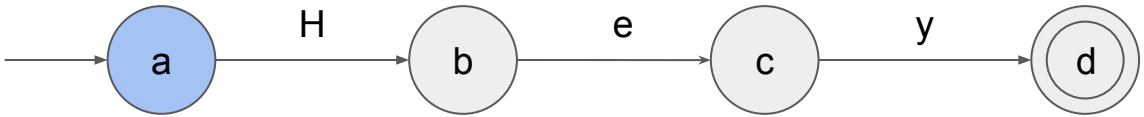
Example:



Input: Hey

Semantics of DFAs

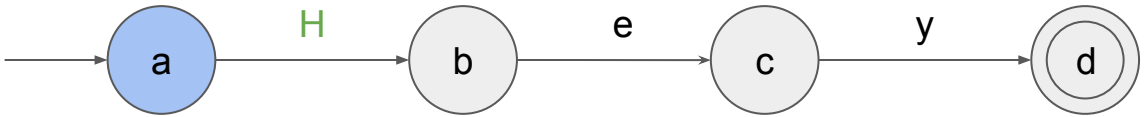
Example:



Input: |Hey

Semantics of DFAs

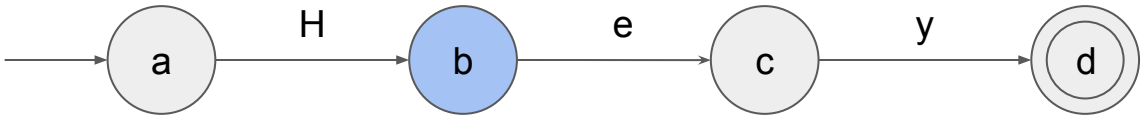
Example:



Input: |Hey

Semantics of DFAs

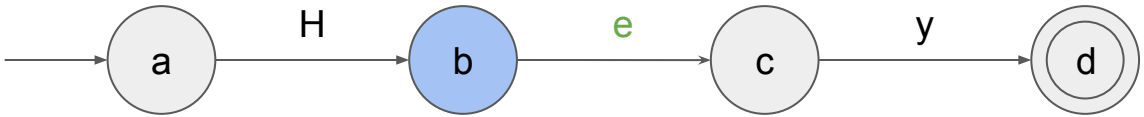
Example:



Input: H|ey

Semantics of DFAs

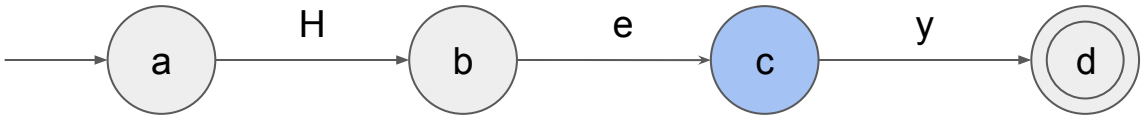
Example:



Input: H|ey

Semantics of DFAs

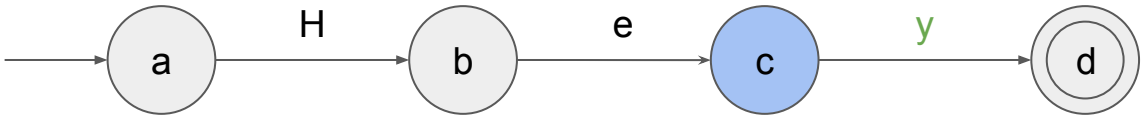
Example:



Input: He|y

Semantics of DFAs

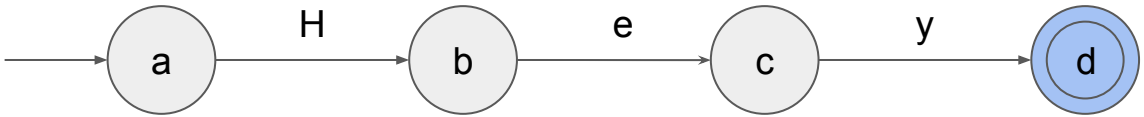
Example:



Input: He|y

Semantics of DFAs

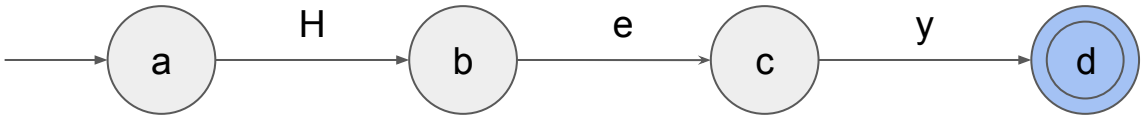
Example:



Input: Hey|

Semantics of DFAs

Example:



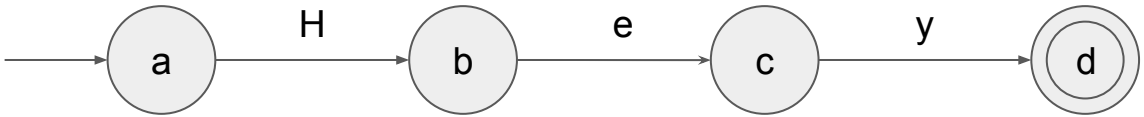
Input: Hey|

We reached the end of the string. We are in an accept state.

So the string is accepted; it is member of the language.

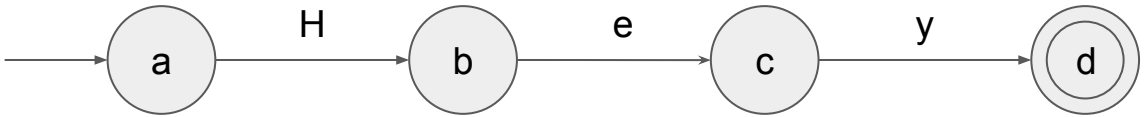
Heey

Is Heey also accepted?
Why?



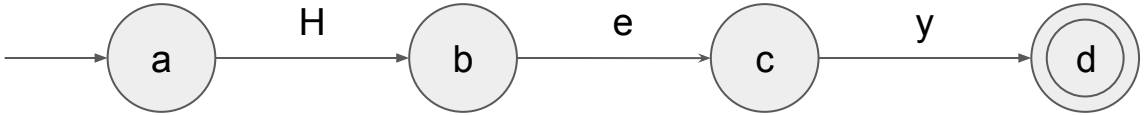
Rejecting a string

No, Heey is not accepted.
Why?



Rejecting a string

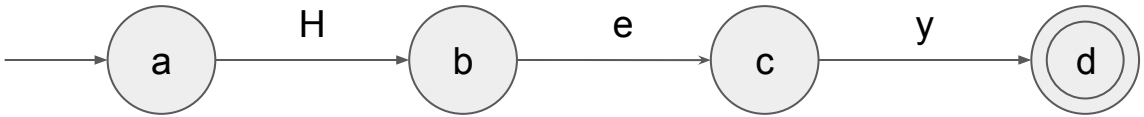
No, Heey is not accepted.
Why?



Because after we reach state c, we have no valid transition to read another letter e. So **we do not reach the end of the string**.

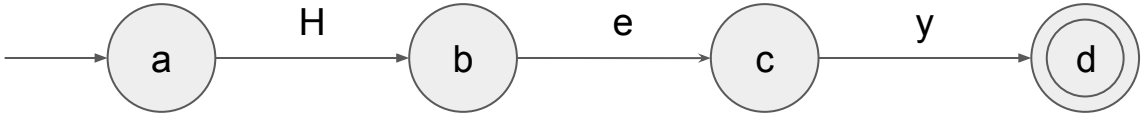
Rejecting a string

This string is not accepted either: He
Why?



Rejecting a string

This string is not accepted either: He
Why?

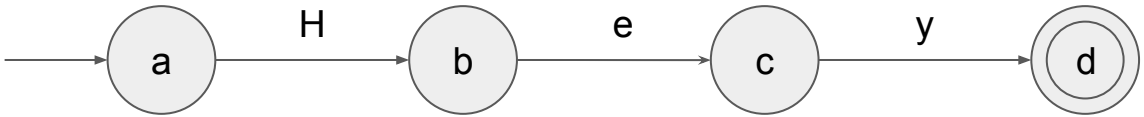


Because when we reach the end of the string, we are not in an accept state (we are in state c).

DFAs are executable

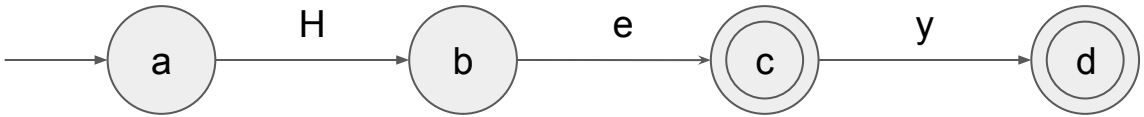
- There exists an algorithm that, given any DFA A and a string s, returns true if A recognises s and false otherwise. It follows the same idea we studied.
- This means that we can just think in terms of DFAs. Computers will be able to run any DFA we design!
- So let’s have some fun with the design of DFAs!

How do we make this automaton accept both He and Hey?



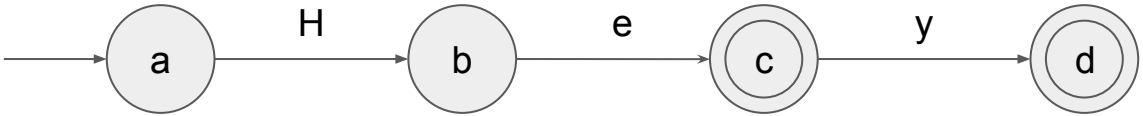
How do we make this automaton accept both He and Hey?

This way:

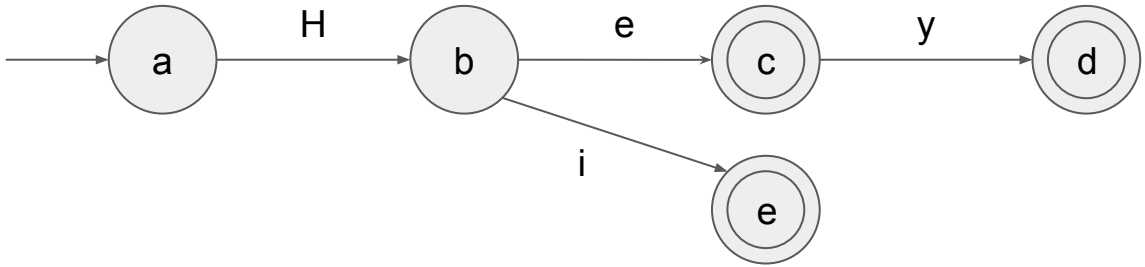


The language of a DFA

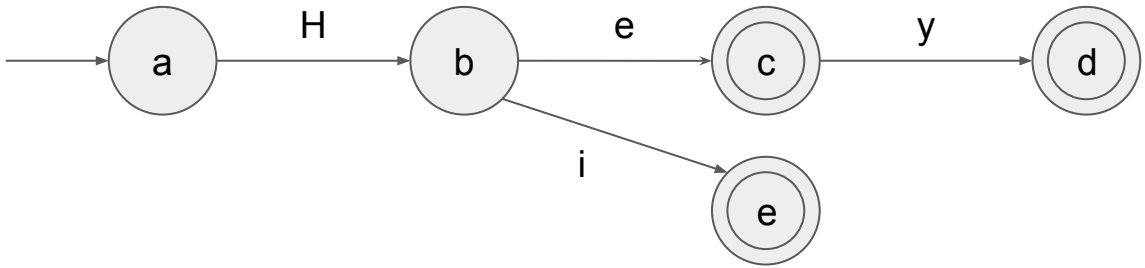
The language of a DFA is the set of all strings that it recognises.
For example, the language of this DFA is {He, Hey}



What's the language of this DFA?



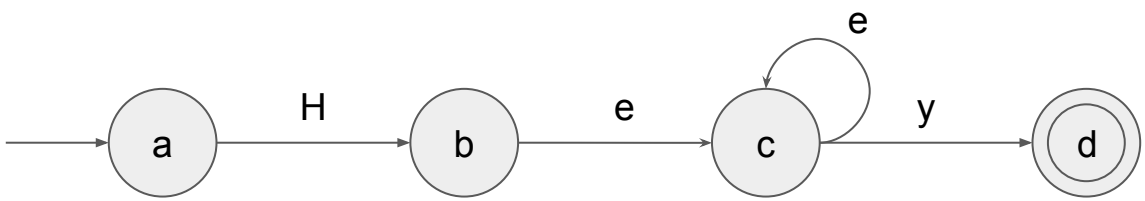
What's the language of this DFA?



It's {He, Hi, Hey}

DFAs can have loops

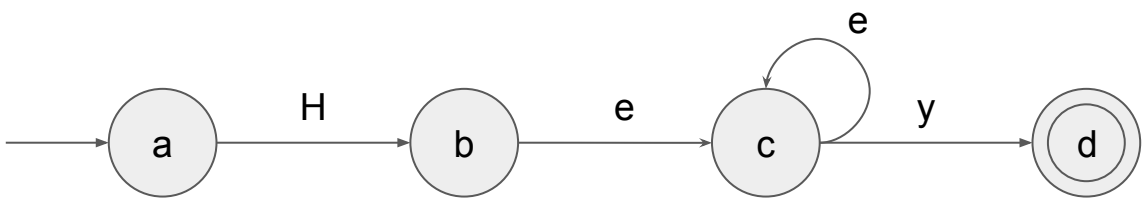
This DFA has a loop:



What's its language now?

DFAs can have loops

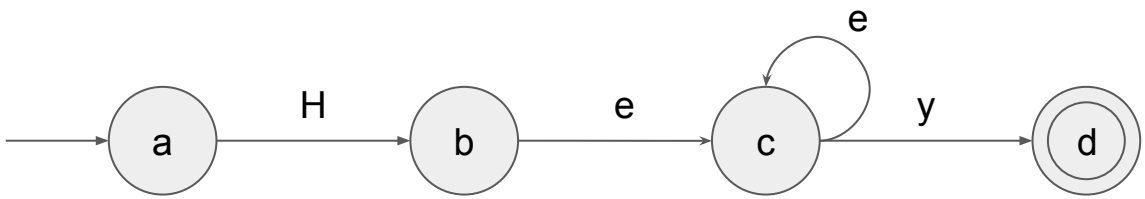
This DFA has a loop:



What's its language now?
All strings that start with He, followed by an arbitrary number of e, and end with a y.

DFAs can have loops

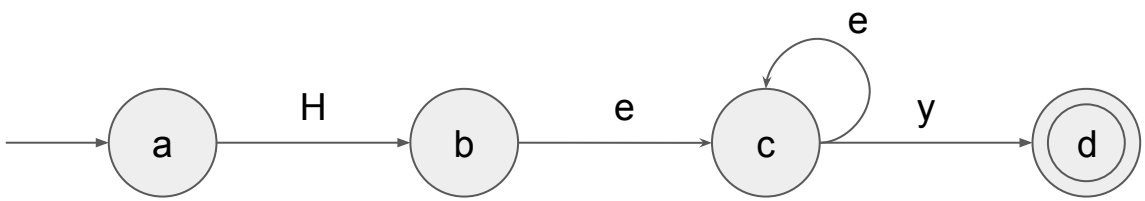
This DFA has a loop:



Examples of accepted strings:
Hey, Heey, Heeeeeey, Heeeeeeeeeeeeeeeey

DFAs can have loops

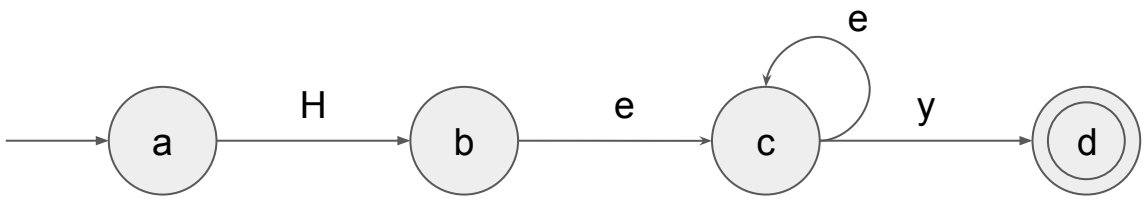
This DFA has a loop:



How big is the language of this DFA?

DFAs can have loops

This DFA has a loop:

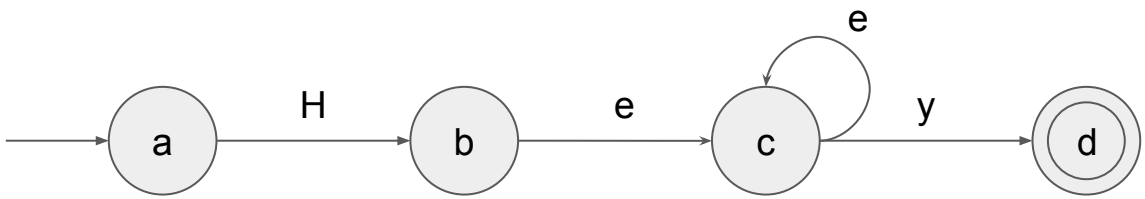


How big is the language of this DFA?

It’s infinite! Because we can always build a string with an extra e in the middle.

DFAs can have infinite languages

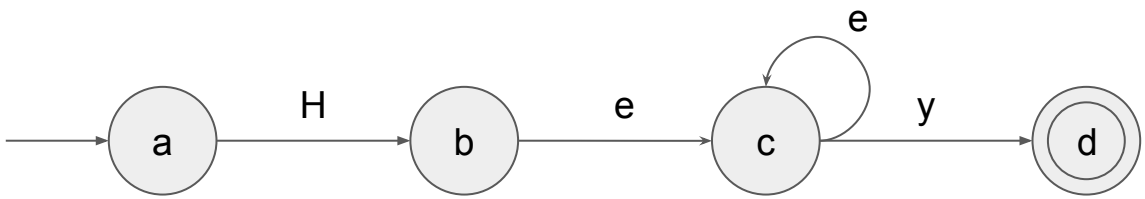
This DFA has an infinite language:



Can the strings in the language also be infinite?

DFAs can have infinite languages

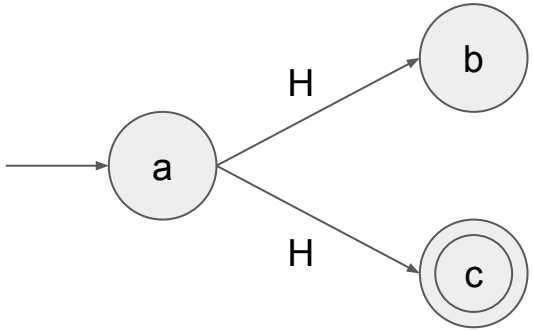
This DFA has an infinite language:



Can the strings in the language also be infinite?
No! All accepted strings are finite. Recall that we need to reach the end of the string to accept.

Remember the D in DFA (D = Deterministic)

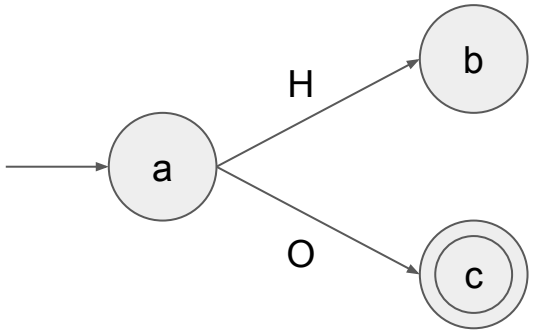
This is **NOT** a DFA:



Because we do not know which transition to follow when we meet an H at state a.

Remember the D in DFA (D = Deterministic)

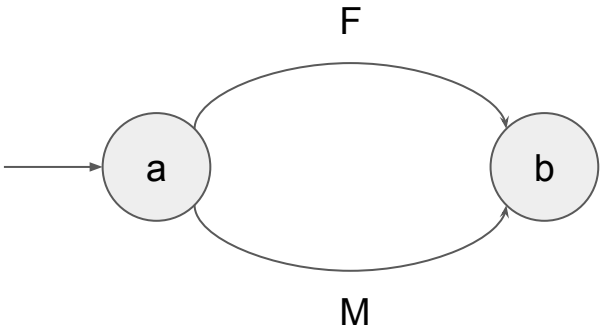
This is a DFA:



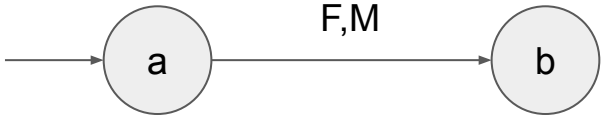
Because each transition from state a has a different character.

Useful Abbreviations

- Different characters may bring to the same state.
- Example:



- We abbreviate this as:

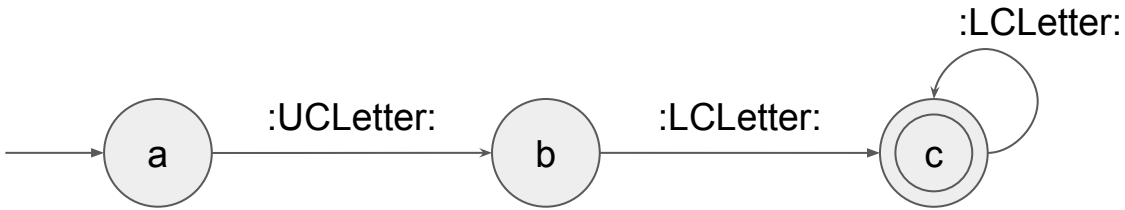


Useful Abbreviations

- We will use :this notation: for character classes.
- Examples:
  - :UCLetter: is any uppercase letter.
  - :LCLetter: is any lowercase letter.
  - :space: is any space character.

A DFA for recognising a first name

- Examples of accepted strings: Fabrizio, Joan, Kim, Lene, Rolf, Luís, Jacopo.



Some models are truly great

- Useful models have useful properties.
- One of these properties for DFAs is **concatenation**.
- You can always concatenate two DFAs and obtain a language that is still recognisable by a DFA.

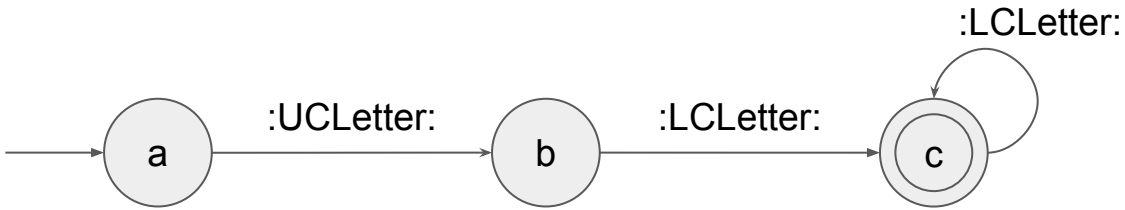
Example of concatenation (plus a space)

- How can we recognise valid full names?
- Example: Homer Simpson

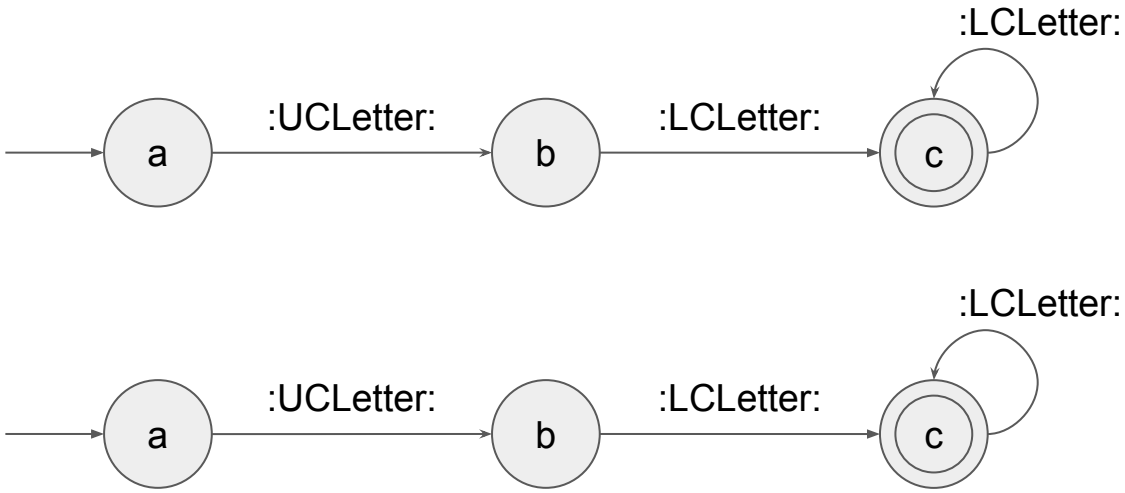
Example of concatenation (plus a space)

- How can we recognise valid full names?
- Example: Homer Simpson
- Idea: we can think of a full name as the concatenation of multiple names, separated by spaces.

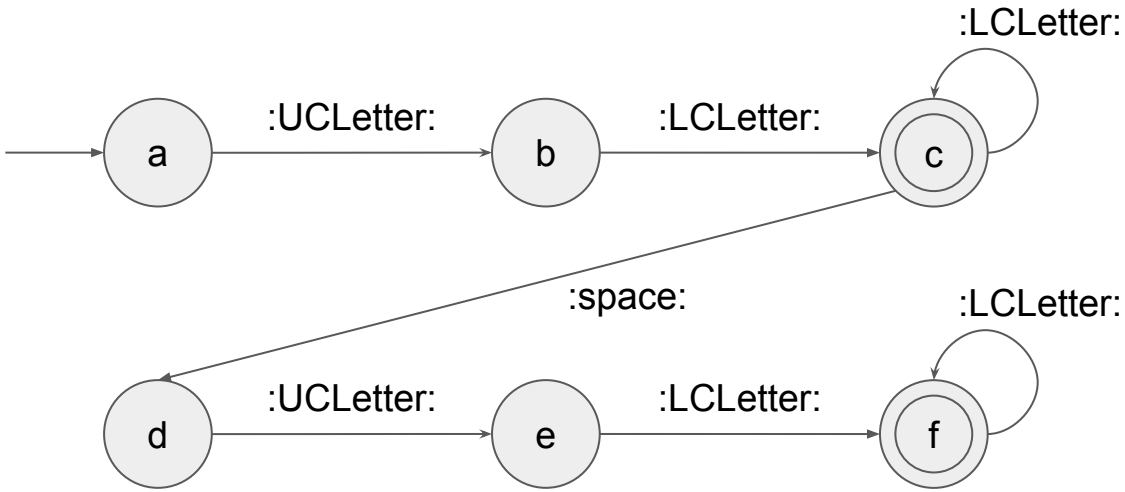
A DFA for Full Names, attempt 1



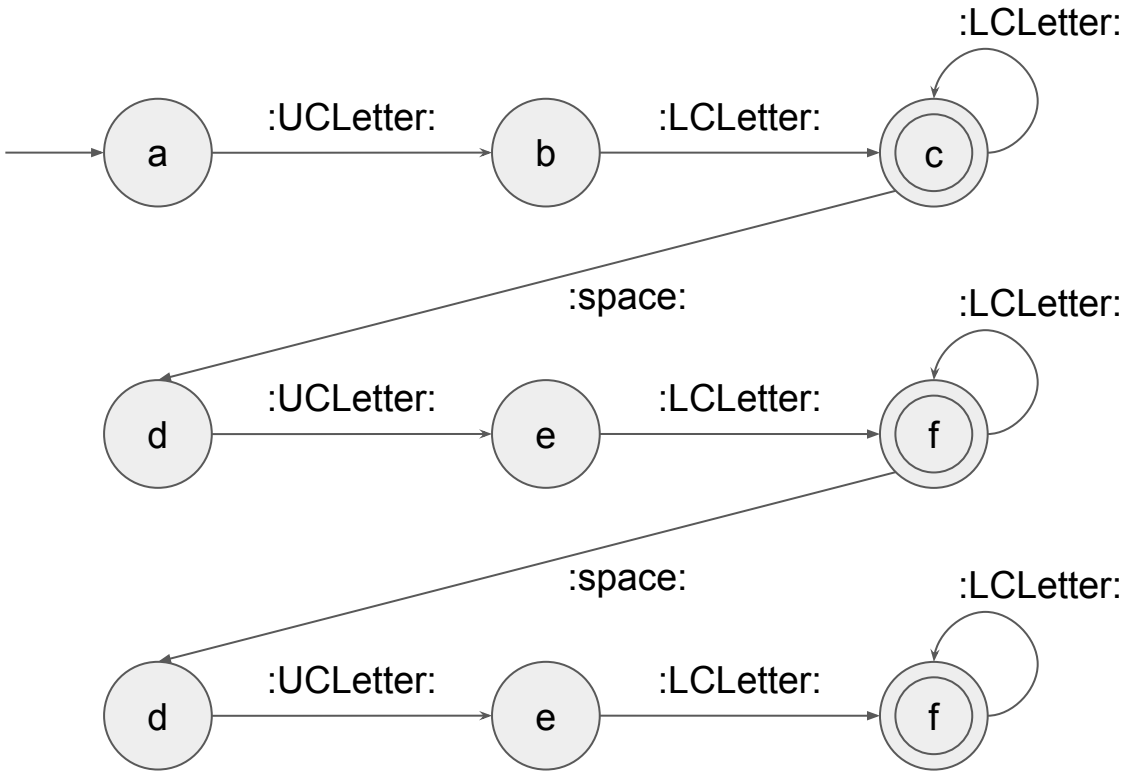
A DFA for Full Names, attempt 1



A DFA for Full Names, attempt 1

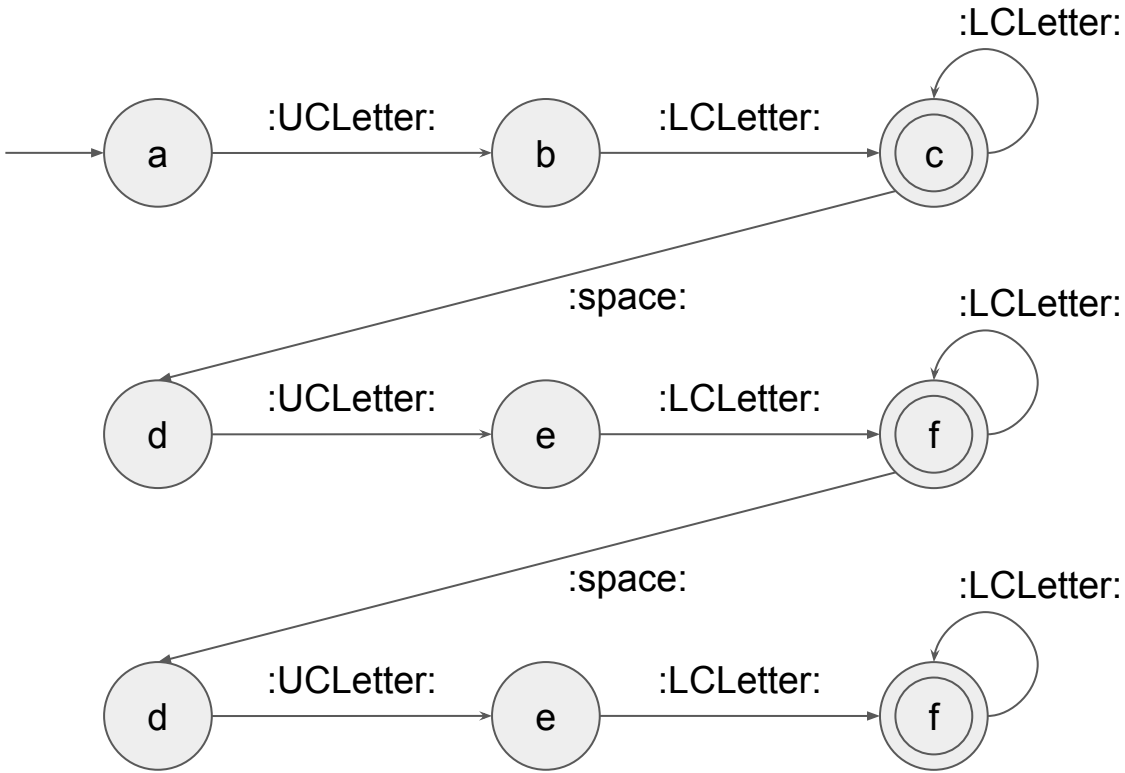


A DFA for Full Names, attempt 2



Problems, problems…

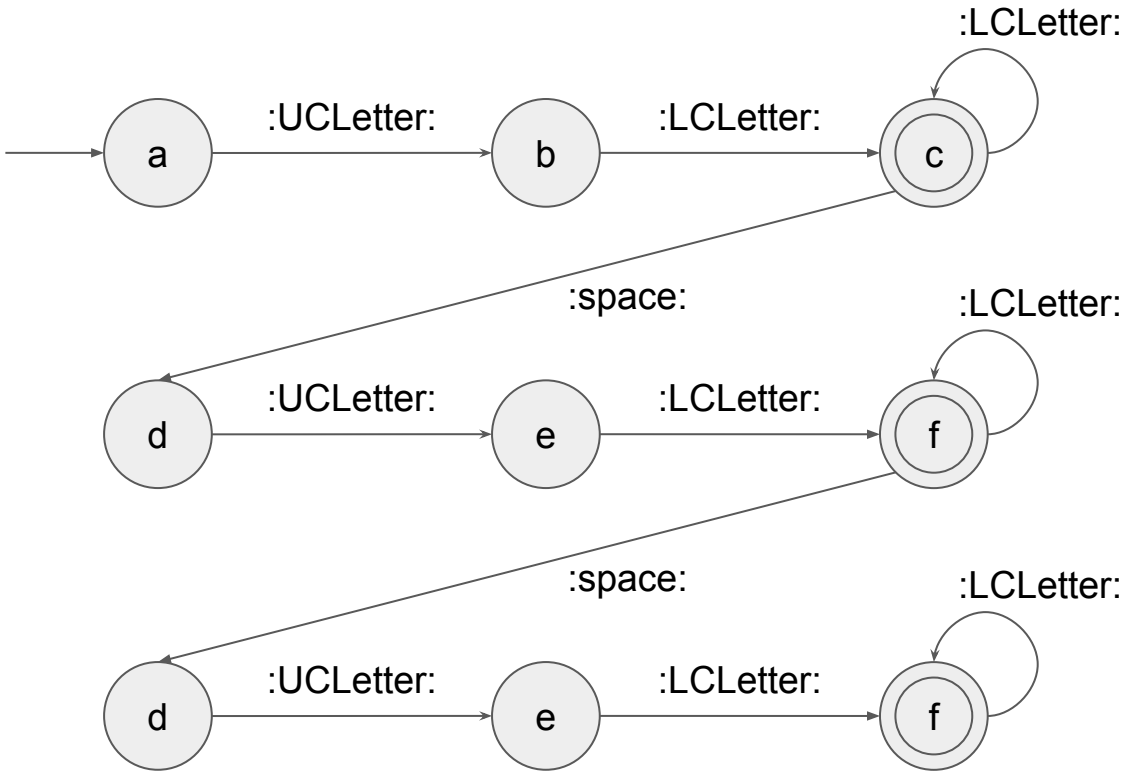
Does that DFA recognise Charles John Huffam Dickens?



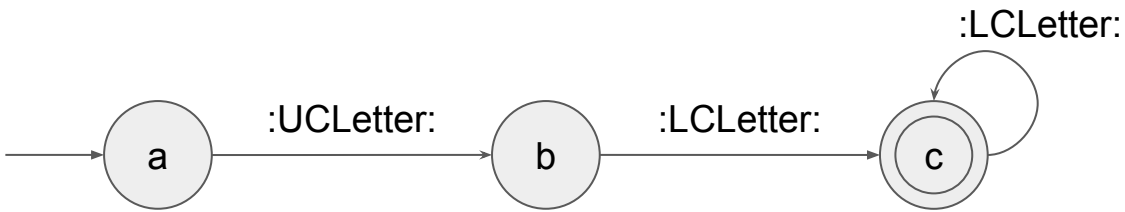
Problems, problems…

Does that DFA recognise Charles John Huffam Dickens?

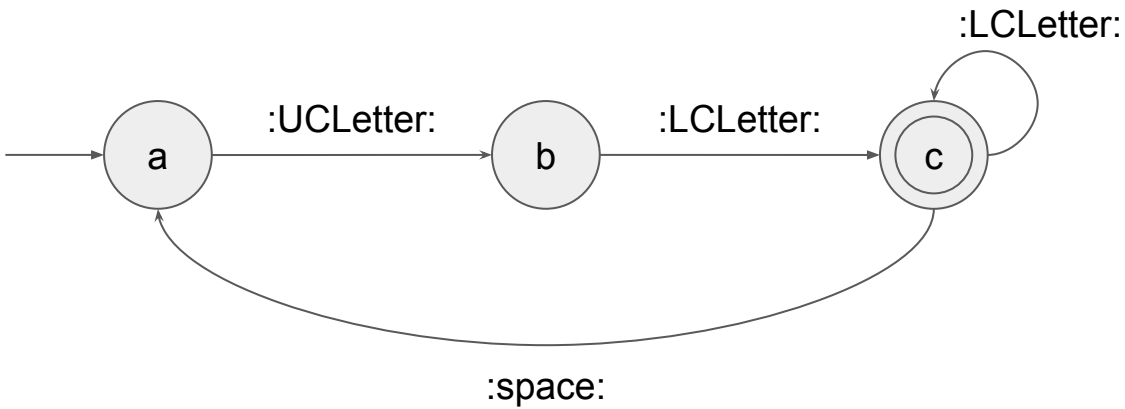
No.



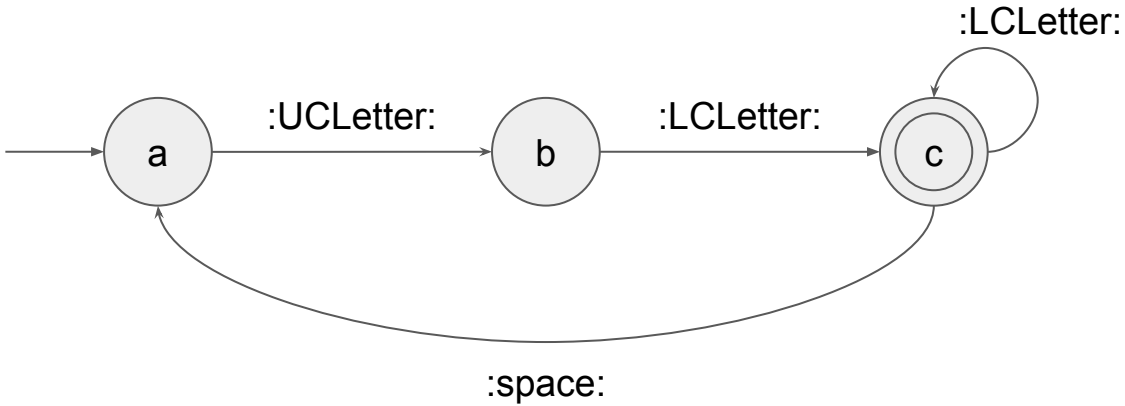
A DFA for Full Names, The Recursive Attempt



A DFA for Full Names, The Recursive Attempt

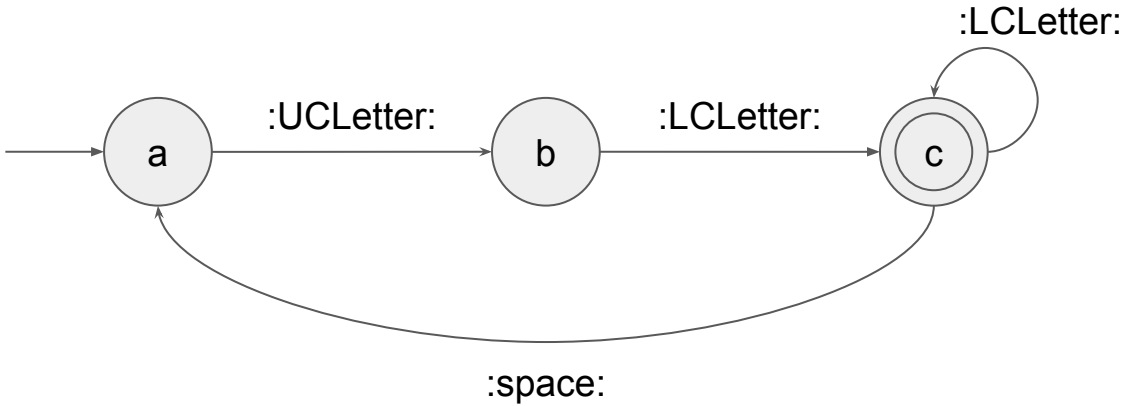


A DFA for Full Names, The Recursive Attempt



Great!!!

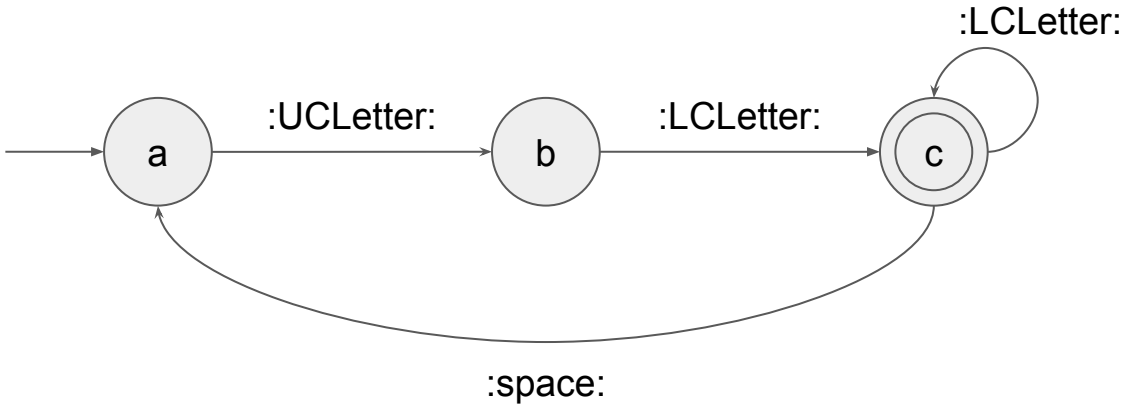
A DFA for Full Names, The Recursive Attempt



Great!!!

Oh no...

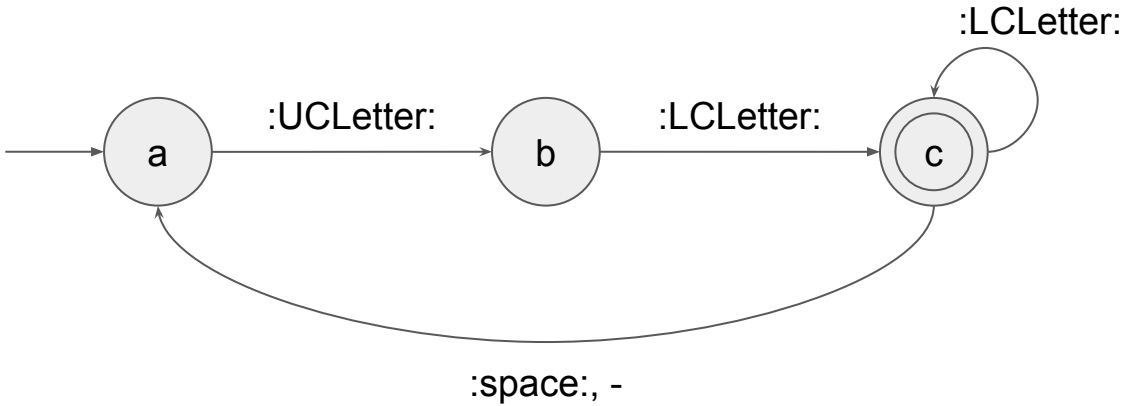
A DFA for Full Names, The Recursive Attempt



Great!!!

Oh no… What about Anne Elizabeth Alice Louise Mountbatten-Windsor?

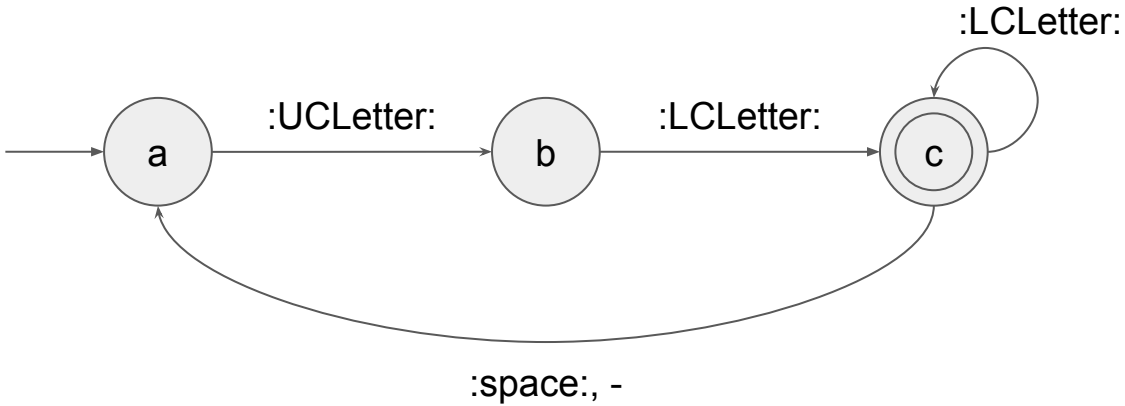
A DFA for Full Names, The Recursive Attempt



Great!!!

Oh no… What about Anne Elizabeth Alice Louise Mountbatten-Windsor?

A DFA for Full Names, The Recursive Attempt



Phew!

Let’s take a step back

A step back: What have we seen?

- DFA: a powerful model for recognising if a string is in a language.
- Simple execution: follow the arrows, reach the end of the string, check if in accept state.
- Looks like we can recognise a lot of things.

A step back: What have we seen?

- DFA: a powerful model for recognising if a string is in a language.
- Simple execution: follow the arrows, reach the end of the string, check if in accept state.
- Looks like we can recognise a lot of things.
- Cool!

A step back: What have we seen?

- DFA: a powerful model for recognising if a string is in a language.
- Simple execution: follow the arrows, reach the end of the string, check if in accept state.
- Looks like we can recognise a lot of things.
- Cool!
- But a (computer) scientist should ask:
  - “A lot of things”? Which things, exactly?

The science of DFAs

- **Q:** What kind of strings can I recognise with a DFA?

The science of DFAs

- **Q:** What kind of strings can I recognise with a DFA?
- How do we answer?..

The science of DFAs

- **Q:** What kind of strings can I recognise with a DFA?
- How do we answer?..
- We need to understand the **limitations** of DFAs.

The science of DFAs

- **Q:** What kind of strings can I recognise with a DFA?
- How do we answer?..
- We need to understand the **limitations** of DFAs.
- So another interesting question is:
  - **Q:** What are things that **cannot possibly** be recognised with a DFA?

The science of DFAs

- **Q:** What kind of strings can I recognise with a DFA?
- How do we answer?..
- We need to understand the **limitations** of DFAs.
- So another interesting question is:
  - **Q:** What are things that **cannot possibly** be recognised with a DFA?
- Which is way more fun. Breaking stuff is the best part of being a scientist.

An experiment to test DFAs

- Let us design an experiment to test what DFAs can do.
- We take some 1st year students and a lecturer in CS.
- We try to come up with a DFA that recognises a language I will propose.
- Maybe we will succeed, maybe we will fail.
- What happens if we fail?

An experiment to test DFAs

- Let us design an experiment to test what DFAs can do.
- We take some 1st year students and a lecturer in CS.
- We try to come up with a DFA that recognises a language I will propose.
- Maybe we will succeed, maybe we will fail.
- If we fail, we will blame the entire model of DFAs.

An experiment to test DFAs

- Let us design an experiment to test what DFAs can do.
- We take some 1st year students and a lecturer in CS.
- We try to come up with a DFA that recognises a language I will propose.
- Maybe we will succeed, maybe we will fail.
- If we fail, we will blame the entire model of DFAs.
  (In the real world, you should prove it, with Math. That's what you'll do in DM553.)

The language of balanced parentheses

- Recognising arithmetic expressions is useful, e.g., (3+2)*4, (4*7)+(4/(3-1)), etc.
- In this language (and many others!), parentheses have to be balanced.
- So a simpler, yet still interesting, problem is that of recognising strings with balanced parentheses.

The language of balanced parentheses

- Some correct strings: (), (()), ((())), ()()(), ()(()), (())((()()())).
- Some incorrect strings: (,), ((), ()), (()(), (())),))((
- Intuitively, a string is in the language if each left parenthesis has a matching right parenthesis and the matched pairs are well nested.

The language of balanced parentheses

- Some correct strings: (), (()), ((())), ()()(), ()(()), (())((()()())).
- Some incorrect strings: (,), ((), ()), (()(), (())),))((
- Intuitively, a string is in the language if each left parenthesis has a matching right parenthesis and the matched pairs are well nested.
- OK, let's try to come up with a DFA that recognises this.

The language of balanced parentheses

- There is no DFA for balanced parentheses.
- Why?
- We need to remember how many open parentheses we have, and this number has no bound. (We cannot predict how many there can be.)
- Since a DFA has a finite number of states, there are always cases where we do not have enough memory.

Context-Free Grammar (CFG)

- Another model for recognising strings.

Context-Free Grammar (CFG)
An informal introduction, with an example

- A CFG has a set of rules (also called rewrite rules, or productions) that look like those below.

$$
\begin{aligned}
S &\rightarrow \text{Hello } T \\
S &\rightarrow \text{Hey } T \\
T &\rightarrow \text{there}
\end{aligned}
$$

Context-Free Grammar (CFG)
An informal introduction, with an example

- The capital letters that appear on the left are called non-terminal characters (non-terminals for short).
- S is the start non-terminal.

$$
\begin{aligned}
S &\rightarrow \text{Hello } T \\
S &\rightarrow \text{Hey } T \\
T &\rightarrow \text{there}
\end{aligned}
$$

Context-Free Grammar (CFG)
An informal introduction, with an example

- All other characters are called terminals, which make up for the actual content of strings that the CFG can recognise.

$$
\begin{aligned}
S &\rightarrow \text{Hello } T \\
S &\rightarrow \text{Hey } T \\
T &\rightarrow \text{there}
\end{aligned}
$$

Context-Free Grammar (CFG)
Semantics

- The idea is: Start from any of the rules for S. Then you “apply” (example coming) that rule to “expand” S. Go on applying rules to expand the non-terminals until what you obtain is exactly the input string.
- The chain of applications that you obtain is called a derivation.

Example of CFG

Input string: Hey there

$$
\begin{aligned}
S &\rightarrow \text{Hello } T \\
S &\rightarrow \text{Hey } T \\
T &\rightarrow \text{there}
\end{aligned}
$$

Example of CFG

Input string: Hey there

$S \rightarrow$ Hello T
$S \rightarrow$ Hey T
$T \rightarrow$ there

S

Example of CFG

Input string: Hey there

$S \rightarrow$ Hello T
$S \rightarrow$ Hey T
$T \rightarrow$ there

$S \rightarrow$ Hey T

By the 2nd rule.

Example of CFG

Input string: Hey there

$S \rightarrow$ Hello T
$S \rightarrow$ Hey T
$T \rightarrow$ there

$S \rightarrow$ Hey T $\rightarrow$ Hey there

By the 3rd rule.

Example of CFG

Input string: Hey there

$S \rightarrow$ Hello T
$S \rightarrow$ Hey T
$T \rightarrow$ there

$S \rightarrow$ Hey T $\rightarrow$ Hey there

Starting from S, we reached exactly the input string, so the string is accepted.

The language of a CFG

The language of a CFG is the set of all strings that can be derived by that CFG (starting from S).

The language of this CFG

$$
\begin{aligned}
S &\rightarrow \text{Hello T} \\
S &\rightarrow \text{Hey T} \\
T &\rightarrow \text{there}
\end{aligned}
$$

is {Hello there, Hey there}.

Another CFG

$$
\begin{aligned}
S &\rightarrow :\text{UCLetter}:L \\
L &\rightarrow :\text{LCLetter}:L \\
L &\rightarrow :\text{LCLetter}:
\end{aligned}
$$

What does it recognise?

Recall:

- :UCLetter: is any uppercase letter
- :LCLetter: is any lowercase letter

Another CFG

$$
\begin{aligned}
S &\rightarrow :\text{UCLetter}:L \\
L &\rightarrow :\text{LCLetter}:L \\
L &\rightarrow :\text{LCLetter}:
\end{aligned}
$$

How do we transform it to a CFG that accepts full names, like our previous DFA?

Another CFG

$$
\begin{aligned}
S &\rightarrow \text{:UCLetter:L } S \\
S &\rightarrow \text{:UCLetter:L} \\
L &\rightarrow \text{:LCLetter:L} \\
L &\rightarrow \text{:LCLetter:}
\end{aligned}
$$

What is this?

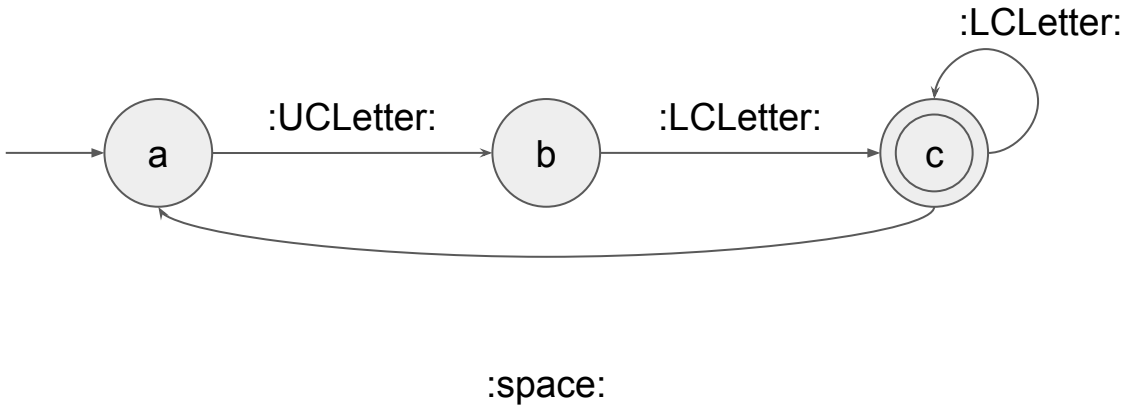
Another CFG

$$
\begin{aligned}
S &\rightarrow \text{:UCLetter:L } S \\
S &\rightarrow \text{:UCLetter:L} \\
L &\rightarrow \text{:LCLetter:L} \\
L &\rightarrow \text{:LCLetter:}
\end{aligned}
$$

What is this? Recursion!

Another CFG

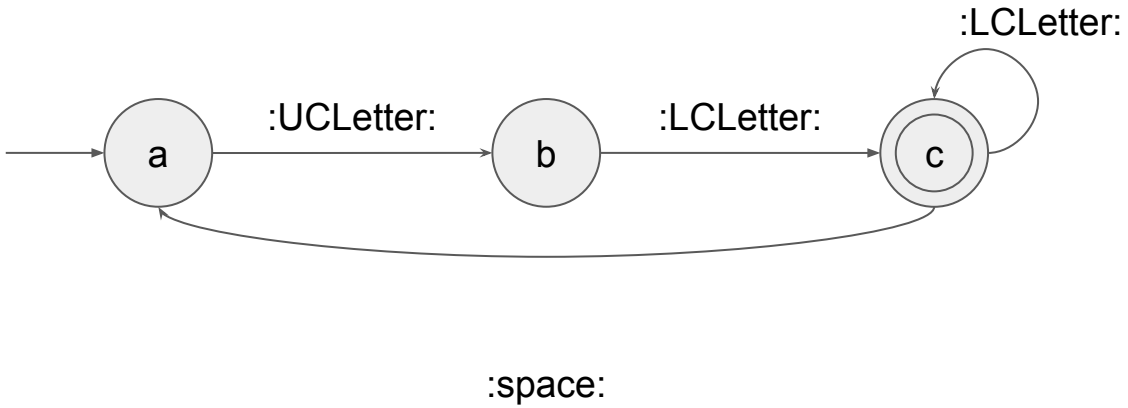
S → :UCLetter:L S
S → :UCLetter:L
L → :LCLetter:L
L → :LCLetter:



It’s exactly the same kind of recursion.

Another CFG

S → :UCLetter:L S
S → :UCLetter:L
L → :LCLetter:L
L → :LCLetter:



It's exactly the same kind of recursion.
But CFGs can do more...

Balanced parentheses

$$
\begin{aligned}
S &\rightarrow (S) \\
S &\rightarrow SS \\
S &\rightarrow ()
\end{aligned}
$$

Balanced parentheses

$$
\begin{aligned}
S &\rightarrow (S) \\
S &\rightarrow SS \\
S &\rightarrow ()
\end{aligned}
$$

Some derivations:

- $S \rightarrow ()$
- $S \rightarrow (S) \rightarrow (())$
- $S \rightarrow SS \rightarrow ()S \rightarrow ()(S) \rightarrow ()(SS) \rightarrow ()(()S) \rightarrow ()(()())$

Balanced parentheses

$$
\begin{aligned}
S &\rightarrow (S) \\
S &\rightarrow SS \\
S &\rightarrow ()
\end{aligned}
$$

- Some correct strings: (), (()), ((())), ()()(), ()(()), (())((()()())).
- Some incorrect strings: (,), ((), ()), (()(), (())),))((

Balanced parentheses

$$S \rightarrow (S)$$
$$S \rightarrow SS$$
$$S \rightarrow ()$$

- Why can we do balanced parentheses with a CFG and not with a DFA?
- Because the kind of recursion that we have in CFGs is more powerful: it has a memory!
- Specifically, when you “expand” a non-terminal, we remember what to do after we are done expanding.

Balanced parentheses

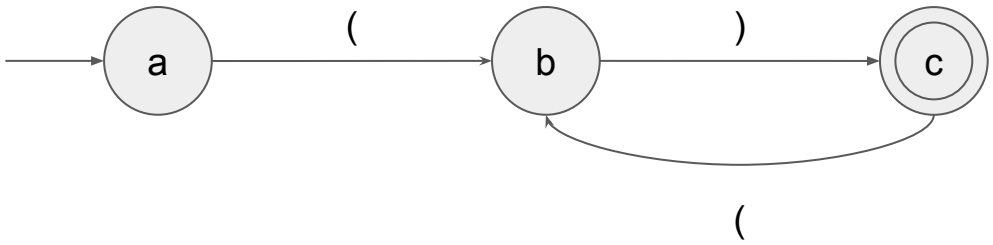
$$
\begin{aligned}
S &\rightarrow (S) \\
S &\rightarrow SS \\
S &\rightarrow ()
\end{aligned}
$$

Some derivations:

- $S \rightarrow ()$
- $S \rightarrow (S) \rightarrow (())$
- $S \rightarrow SS \rightarrow ()S \rightarrow ()(S) \rightarrow ()(SS) \rightarrow ()(()S) \rightarrow ()(()())$

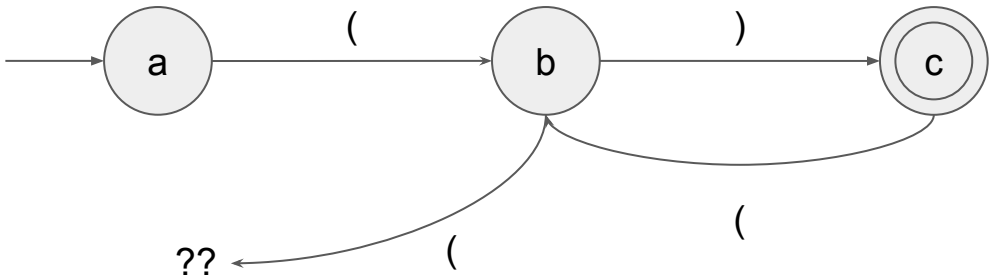
Balanced parentheses

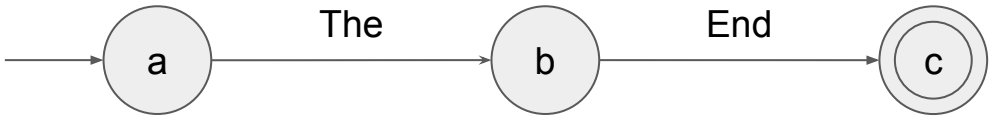
$$S \rightarrow (S)$$
$$S \rightarrow SS$$
$$S \rightarrow ()$$

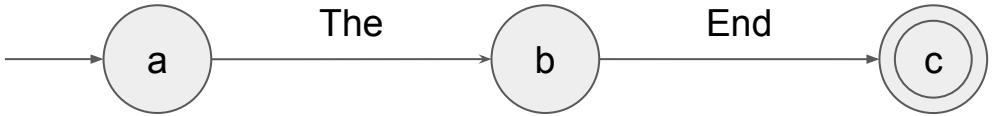


Balanced parentheses

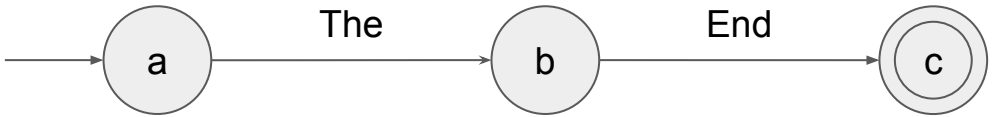
$$S \rightarrow (S)$$
$$S \rightarrow SS$$
$$S \rightarrow ()$$



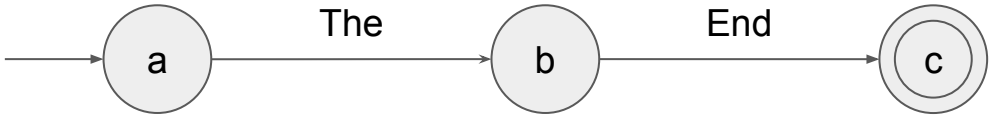

a
The
b
End
c



Wait a moment...



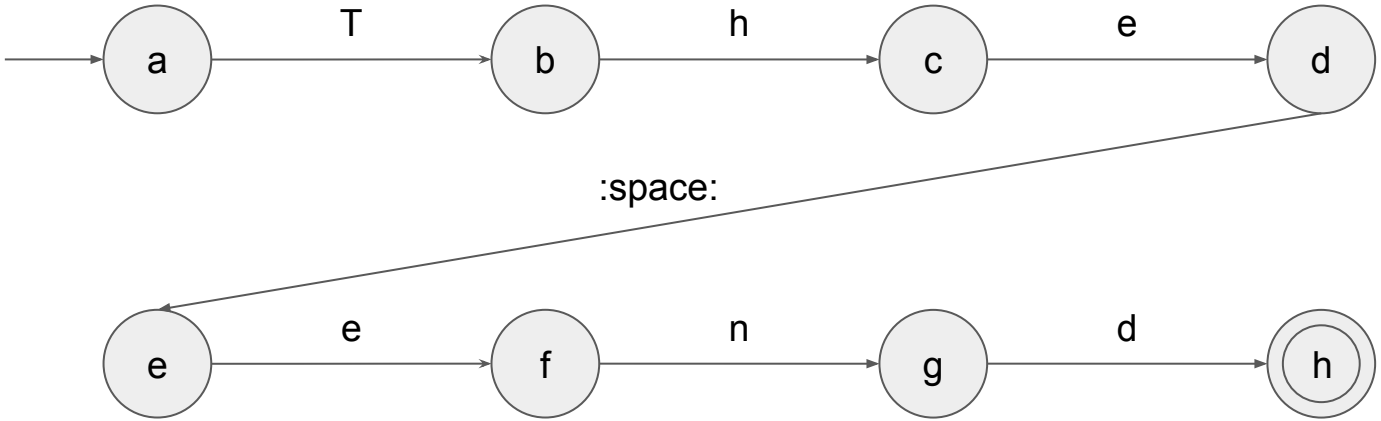
Wait a moment... is that a DFA?



Wait a moment... is that a DFA?

No, because transitions are labelled with single characters, not strings! (See many slides back.)

Don’t take these things too lightly, they can be tricky.



Questions?