https://pixabay.com/photos/maze-graphic-render-labyrinth-2264/

Subset Construction Algorithm

- To convert a non-deterministic FSR in to a deterministic FSR we use the **subset construction algorithm**
- It can be used on any non-deterministic finite state recogniser

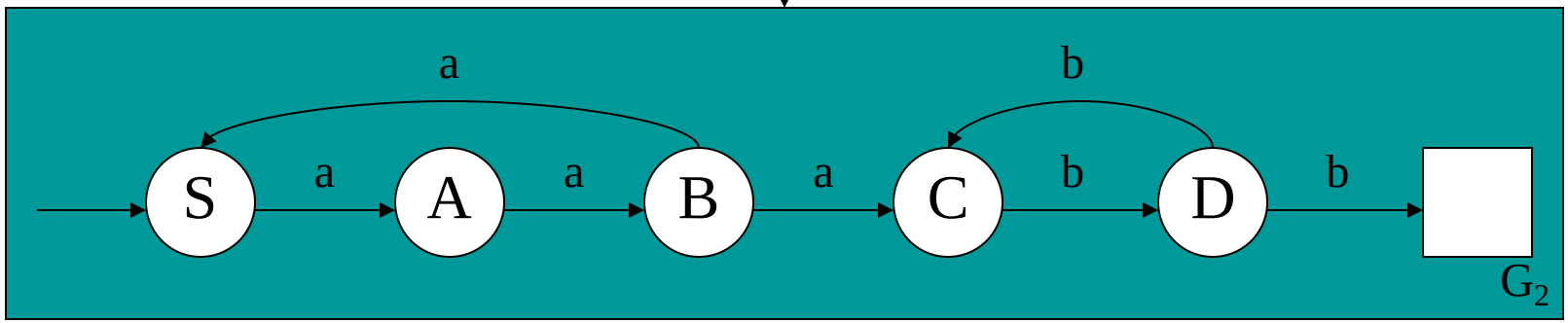
A Non-deterministic FSR for G_2

$S \rightarrow aA$ $\quad C \rightarrow bD$

$A \rightarrow aB$ $\quad D \rightarrow b \mid bC$

$B \rightarrow aS \mid aC$

G_2



Converting the Example

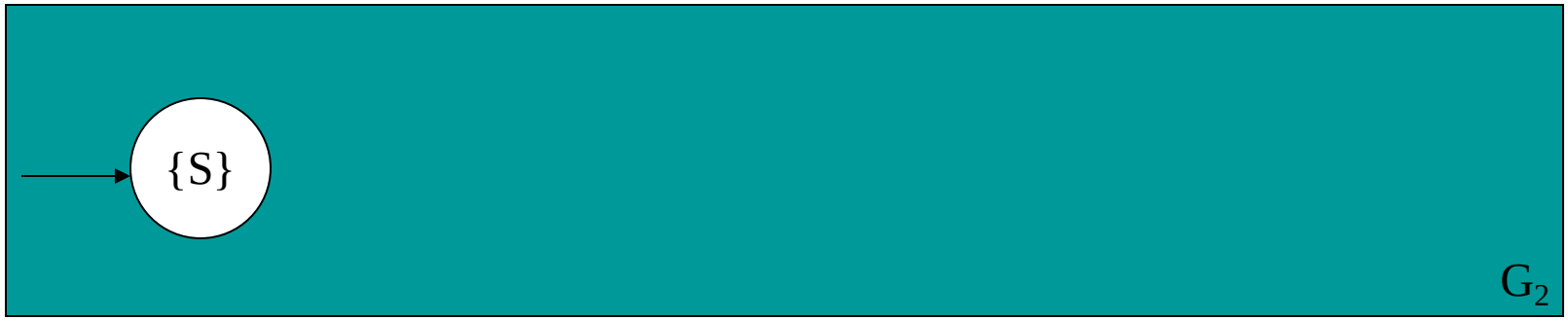
M – the non-deterministic FSR

M’ – the deterministic FSR

- The states in **M’** correspond to *sets* of states in **M**
  - Each state in **M’** might be made up of more than one state from **M**

Converting the Example

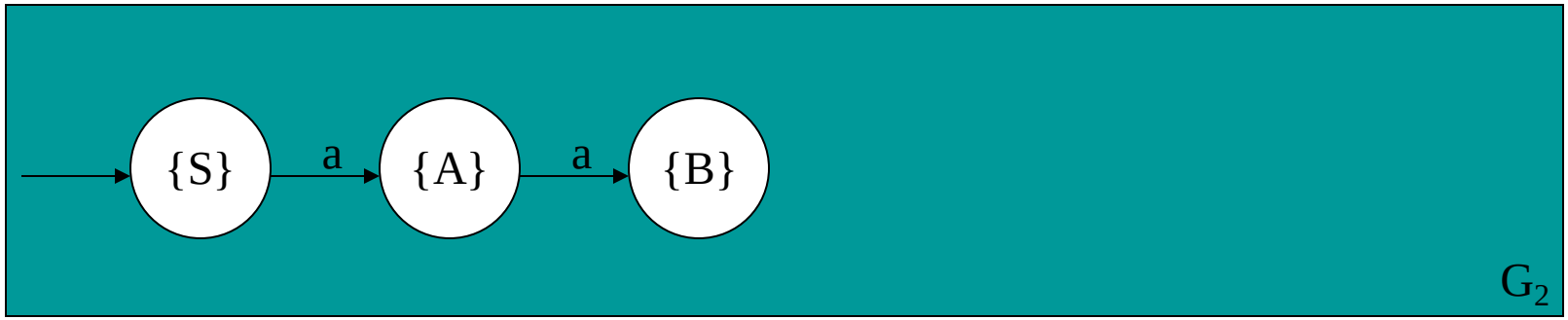
- Begin with a start state for **M'** labelled {S}
  - "the state in **M'** corresponds to the set of states consisting of just the state S in **M**"



Converting the Example

- From state S in **M** we go (deterministically) to state A using input symbol *a* and from state A we go to state B using symbol *a*

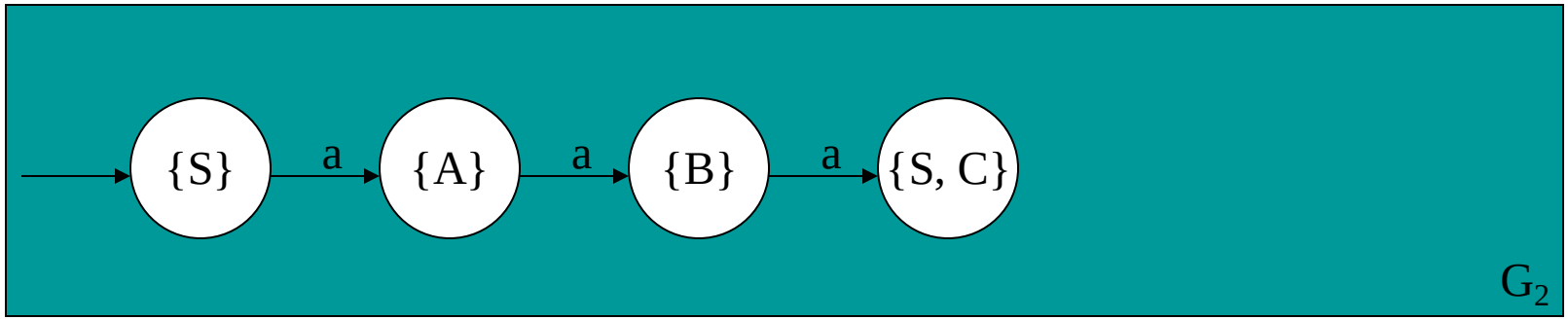
$$S \rightarrow aA,\ A \rightarrow aB$$



Converting the Example

- From state B in **M** using input symbol *a* we go to either state S or state C
- We create a new state labelled {S, C}

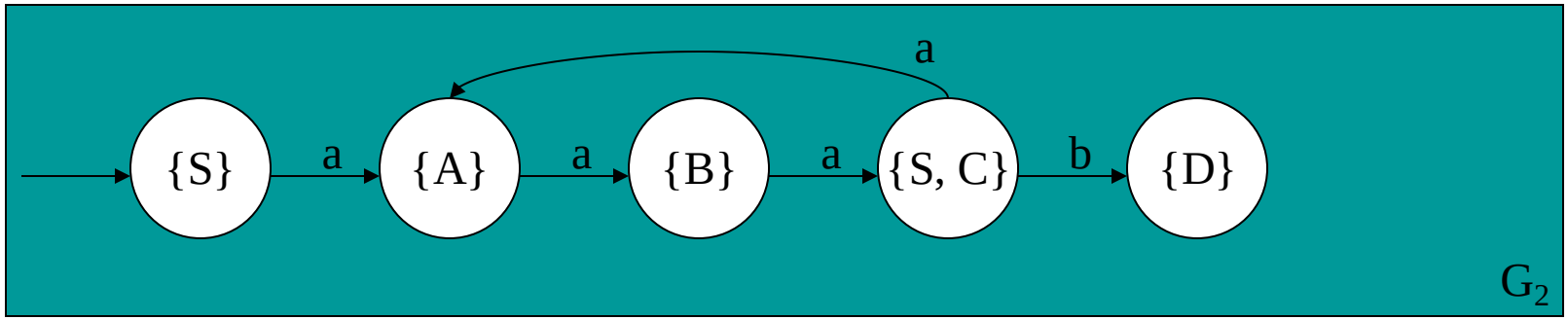
$$B \rightarrow aS \mid aC$$



Converting the Example

- We are now dealing with two states from **M**
- The productions are grouped together and treated as one set of rules

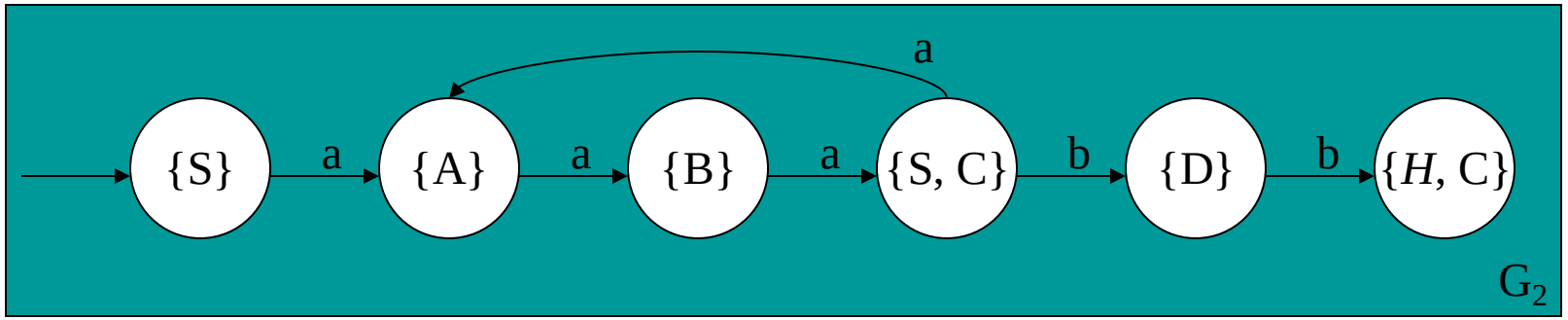
$$S \rightarrow aA,\ C \rightarrow bD \longrightarrow \{S, C\} \rightarrow aA \mid bD$$



Converting the Example

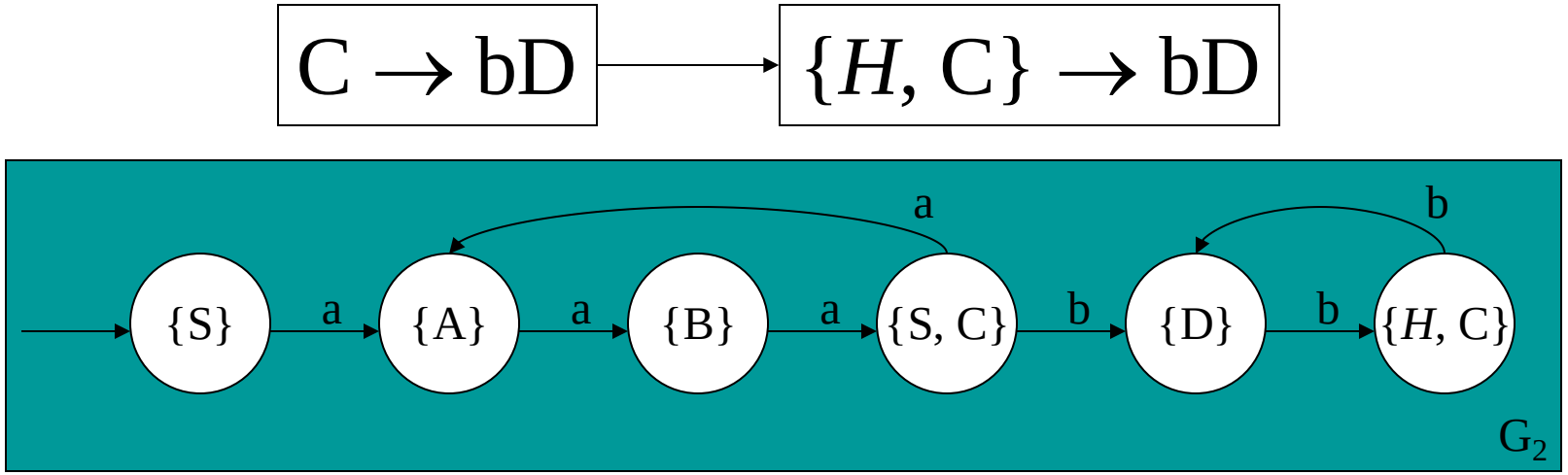
- From state D in **M** we go to the halt state, but we also go to state C using the input symbol *b*

$$D \rightarrow bH \mid bC$$



Converting the Example

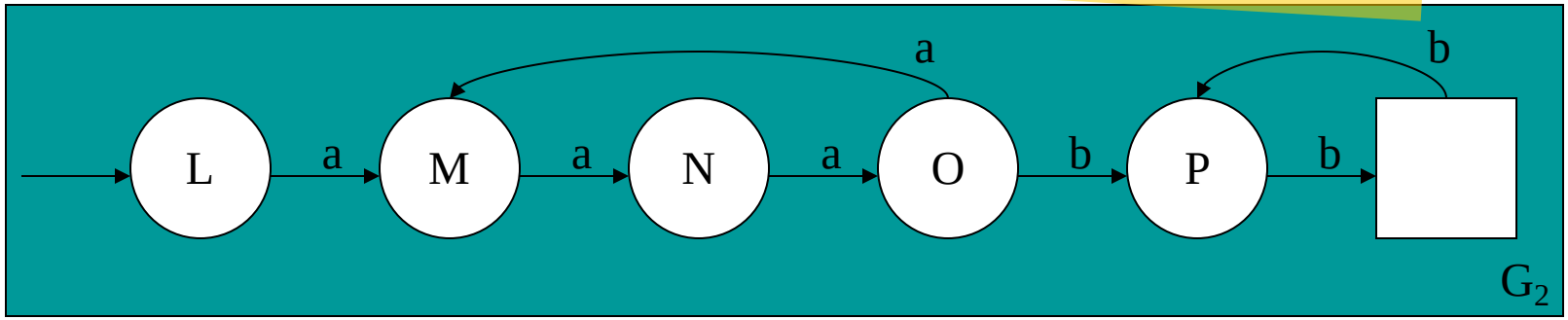
- Next, we deal with the state {*H*, C}



Converting the Example

- Finally, we mark the halt state and change the state names in **M'** as they do not match the rules

Things to double check! 1. Check your new machine is deterministic. 2. Does it accept/reject the same sentences as the original?



Subset Construction Algorithm

- Step 1: Mark the start state
- Step 2: For each rule (or set of rules) group together all the non-terminals (including the halt state) that are preceded by the same terminal. The new set of non-terminals forms a new state in the FSR.
- Step 3: Repeat step 2 until complete

Other methods (e.g. using tables) are available. Homework! Google “NFA to DFA conversion” to find out more.

A Deterministic FSR for G_3

$S \rightarrow aS \mid aB \mid bC$

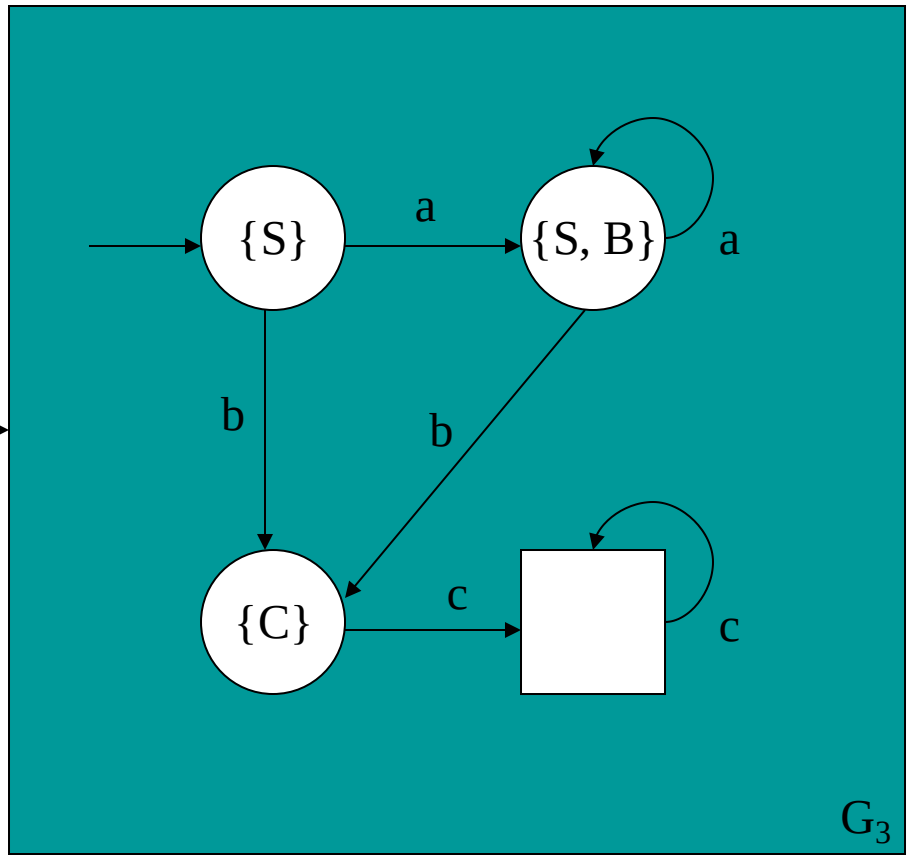
$B \rightarrow bC$

$C \rightarrow cC \mid c$ G_3

$\{S, B\} \rightarrow$

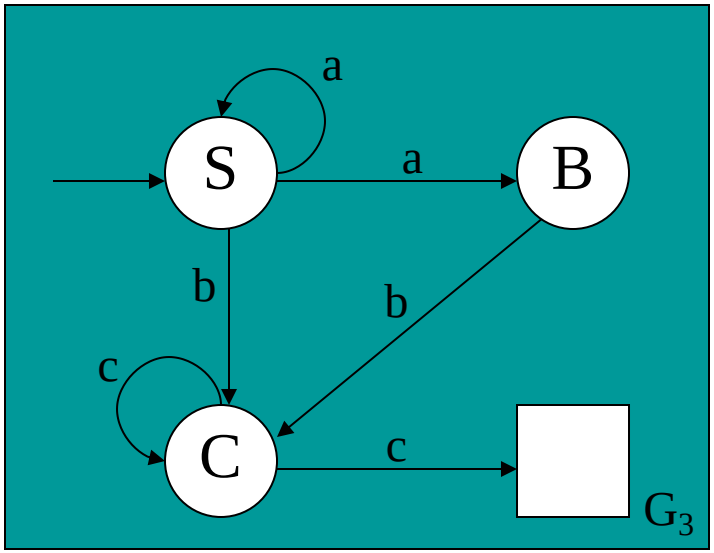
$aS \mid aB \mid bC$

$\{H, C\} \rightarrow cC \mid cH$

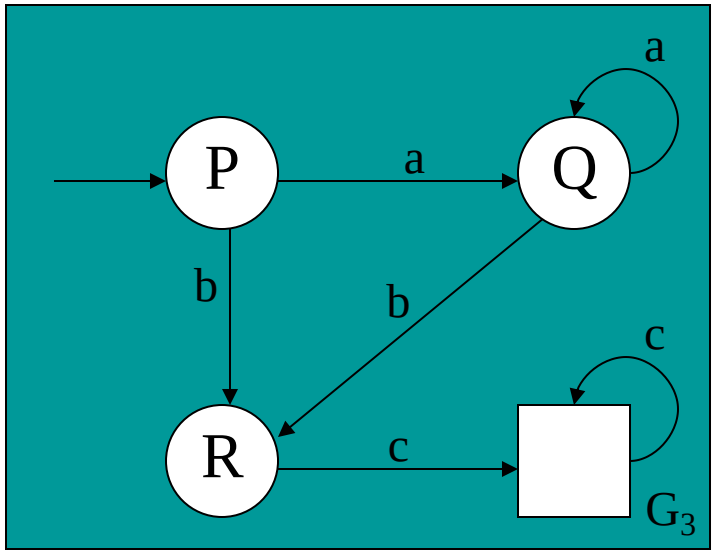


Equivalence

- Consider these two finite state recognisers
- They are **equivalent** to each other



Non-deterministic



Deterministic

Equivalence

- The two machines agree on whether or not a string is grammatical.
- The grammars do not give the same structure to the string so they are only **weakly** equivalent
- Non-deterministic finite state recognisers are not more powerful than deterministic finite state recognisers

Equivalence

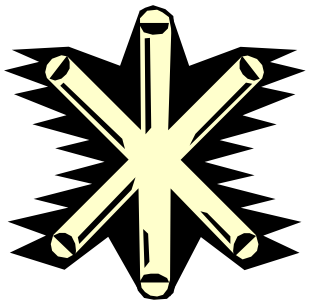
- For every regular grammar there is an equivalent deterministic finite state recogniser
- A finite state recogniser can be converted into an equivalent regular grammar
  - The class of regular grammars is equivalent to the class of deterministic finite state machines.

Equivalence

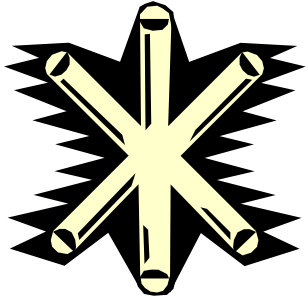
- It is also possible to find a *minimal* finite state recogniser, which has the least number of states.
- Showing the equivalence of two machines or grammars cannot always be done for the more powerful grammars and machines, which we will see later in the module.

Regular Expressions

Type 3

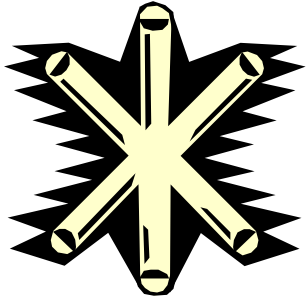


Regular Expressions



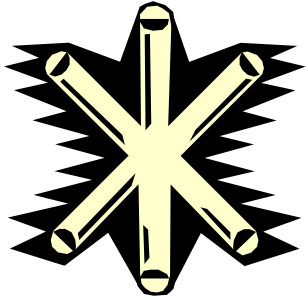
- Regular grammars can also be represented as **regular expressions**
- Regular expressions are a set of rules that describe a generalised string
- They are used for pattern matching and substitution which makes them very useful for text manipulation
- More compact than a regular grammar

Regular Expressions



- There are no production rules
- We will only cover a small part of regular expressions here
- We will also allow ‘or’ and parentheses
- For a more detailed description see Linux environments, Java, Perl, other scripting languages and some text editors

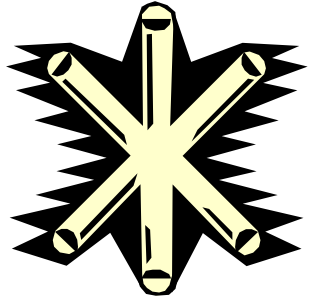
Regular Expressions



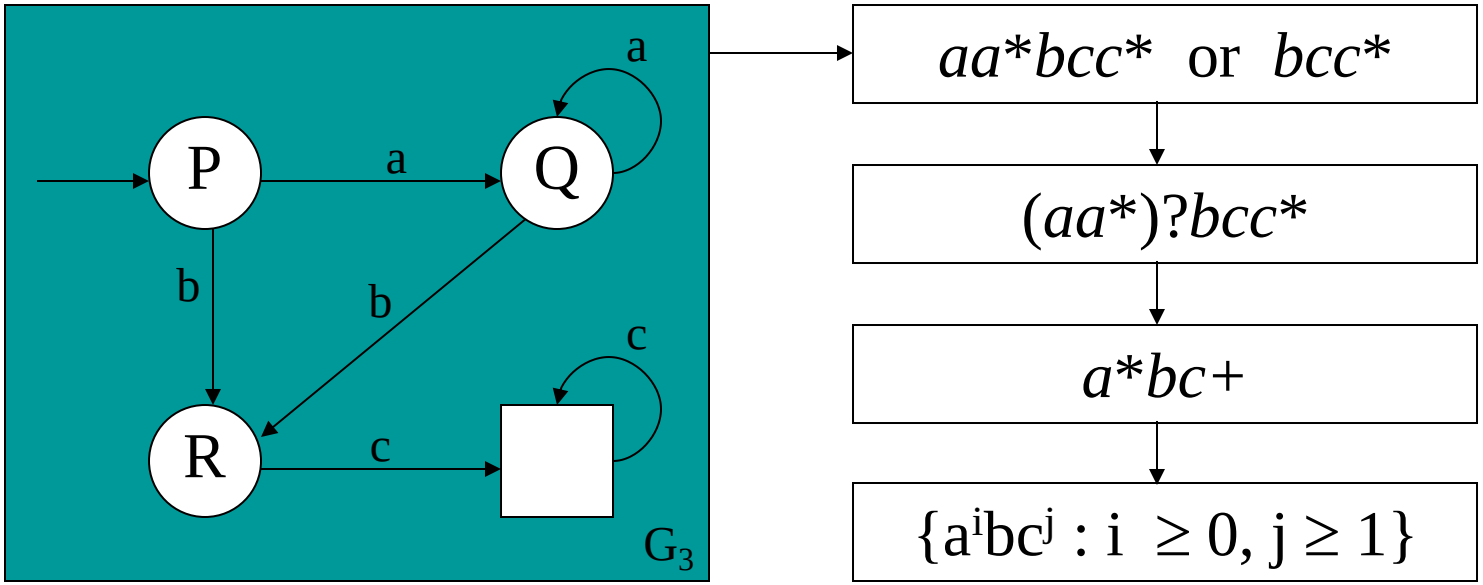
- Regular expressions are made up of
  - Individual symbols/characters (e.g. *a*, *b*, *c*, etc.)
  - Specific quantifiers (*, +, ?)

Quantifier	Description	Example	Matches	Not Match
*	Zero or more instances of the atom	*ab*c*	*ac, abc, abbbbc*	*abb*
+	One or more instances of the atom	*ab+c*	*abc, abbbbc*	*ac*
?	Zero or one instances of the atom	*ab?c*	*ac, abc*	*abbc*

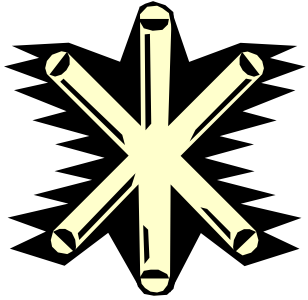
Regular Expressions



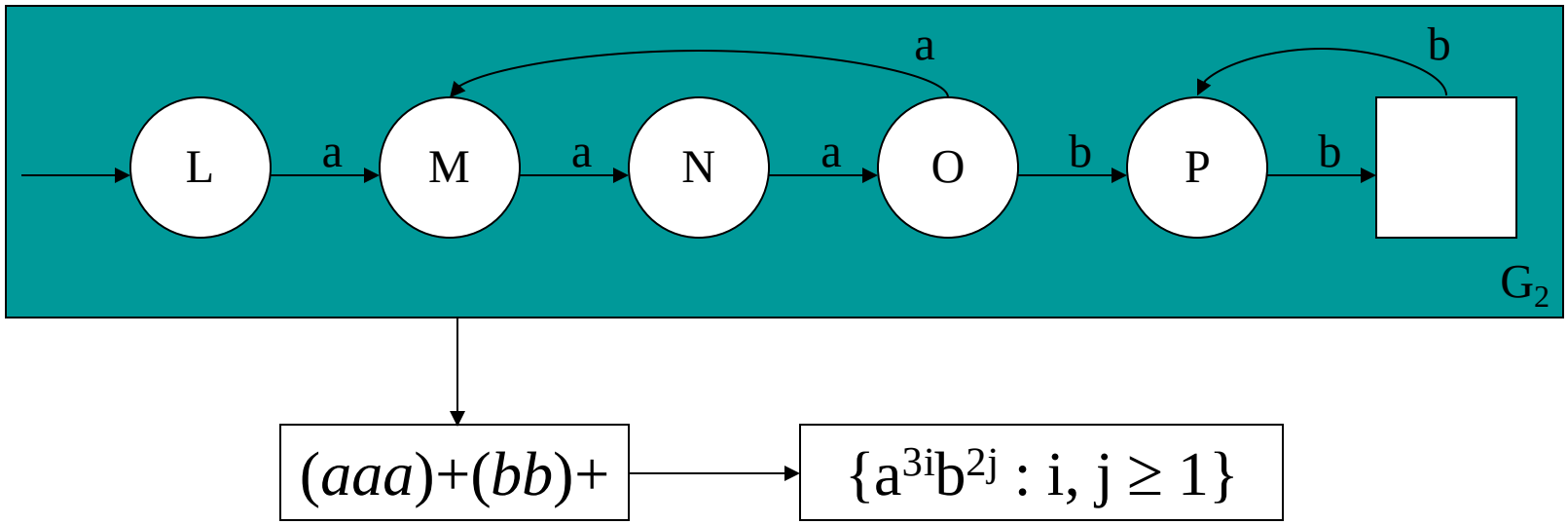
- The previous example can be represented as a regular expression



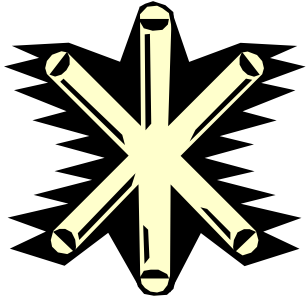
Regular Expressions



- With the following example we get…



Uses of Regular Expression



- In compilers, some parts of a programming language can be represented. For example:
  - *letter*(*letter* or *digit*)* – a variable name
  - *digit*+ or *digit*+.*digit*+ – typical integers and real numbers
- Used for identifiers, numbers, character strings, symbols (:= and !=), reserved words (if, class)
- Cannot be used for checking matching brackets (in the unlimited case)

Brief Overview of Compilers

- **Lexical analyser –** part of a compiler used to recognise the basic units (**tokens**)
  - Specified as a regular expression or grammar
- **Syntax analyser –** checks that the tokens are put together in the correct arrangement
  - Specified as a context-free grammar (week 13)
- Other parts handle language specification (e.g. symbol table, identifiers declared first)

One Last Example

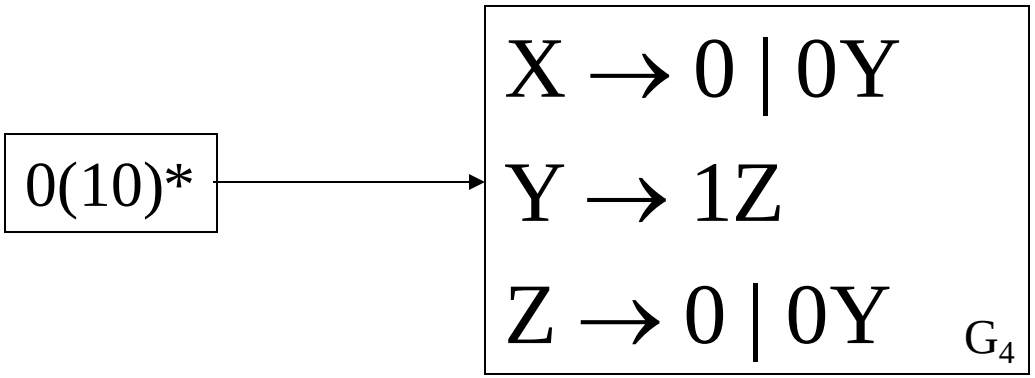
- We shall start with the regular expression:

$$0(10)^*$$

- This would give us any of the following:
  - 0, 010, 01010, 0101010, etc.

One Last Example

- We can convert the regular expression into a regular grammar (G_4):



One Last Example

- We can convert the regular grammar into a non-deterministic finite state recogniser

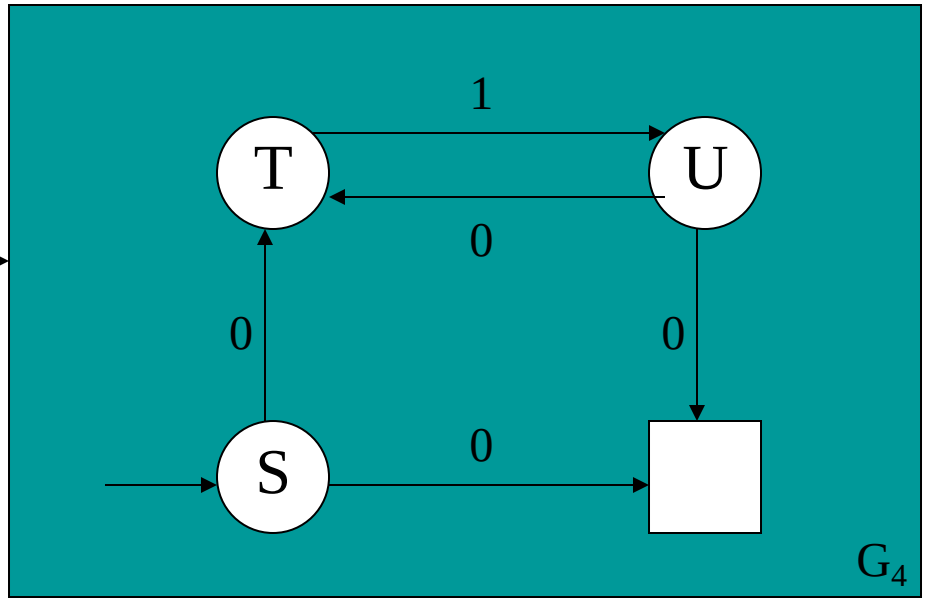
S

T

U

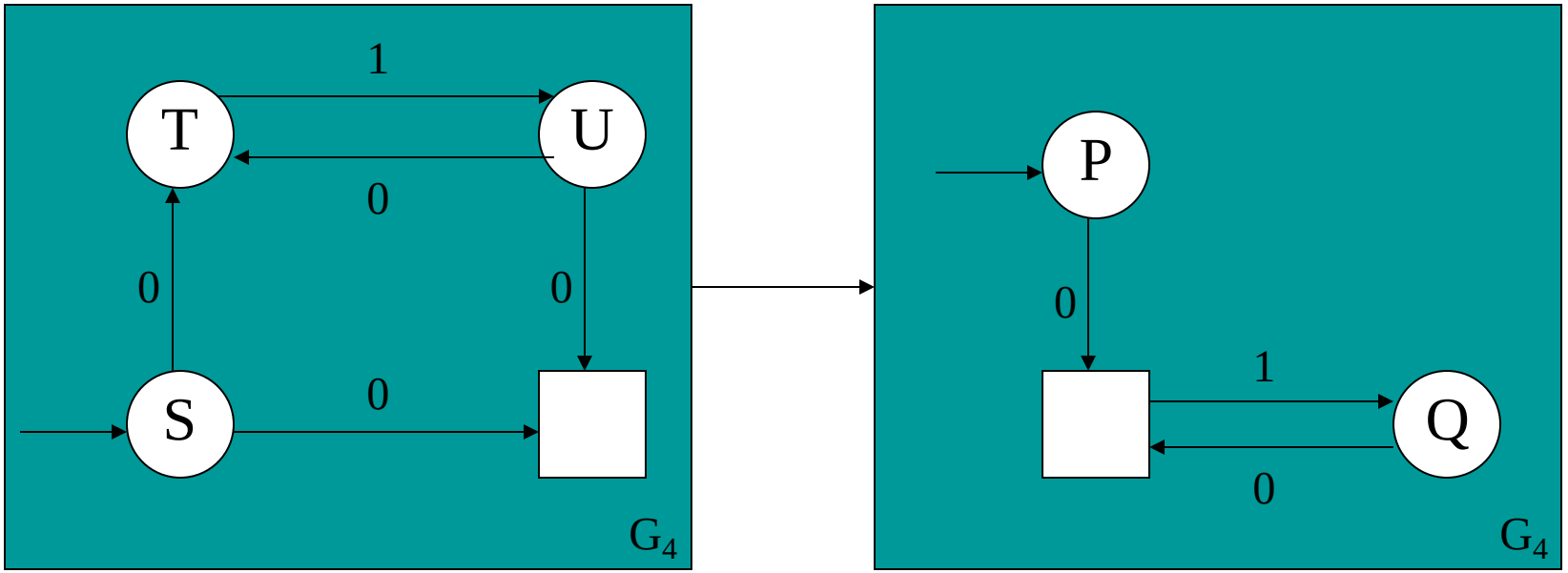
$$X \rightarrow 0 \mid 0Y$$
$$Y \rightarrow 1Z$$
$$Z \rightarrow 0 \mid 0Y$$

G_4



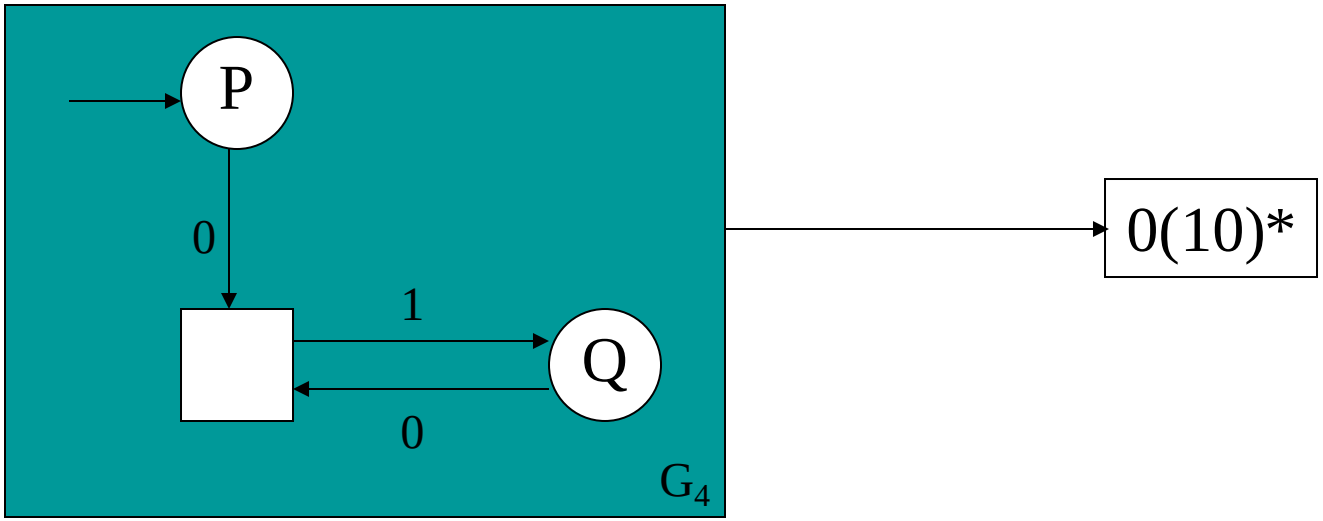
One Last Example

- We can convert a non-deterministic FSR into a deterministic FSR



One Last Example

- We can convert the deterministic FSR back into the regular expression
- All 3 notations are equivalent



Chomsky Hierarchy

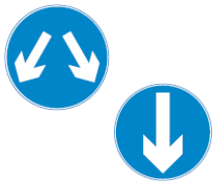
Type	Grammar	Machine	Other Equivalent
3	Regular	Finite State Recogniser	Regular Expressions

Summary and conclusions

- A regular grammar has rules only of the form $X \rightarrow yZ$ or $X \rightarrow y$

- For every regular grammar there is a finite state recogniser that accepts strings derived from that grammar

- Any non-deterministic FSR can be replaced by an equivalent deterministic FSR



- Are also equivalent to regular expressions

https://pixabay.com/photos/cat-domestic-cat-sleep-pet-animal-4189697/