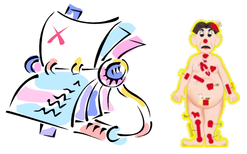
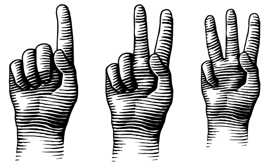
SCC.312 Languages and Compilation (Week 12)

Paul Rayson
p.rayson@lancaster.ac.uk

Last Week (week 11)

Reminder

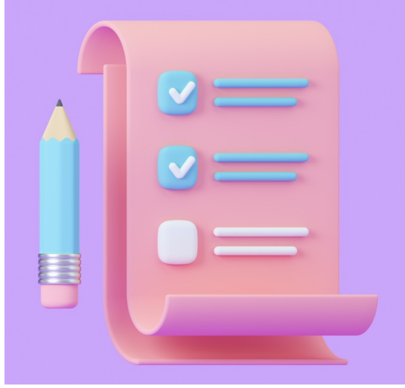
- Introduction to the elements and methods of defining formal languages
- Definitions of basic elements of languages
  - Strings, sentences, alphabets, operations
- Different ways of defining languages
  - Set definitions, decision programs, grammars
- Derivation and parsing
- Introduction to phrase structure grammars

This week's topics

- Regular Grammars
- Language of the grammar – L(G)
- Finite State Recognisers
  - Non-deterministic and deterministic
  - Subset construction algorithm
- Regular Expressions
- Equivalence

Learning objectives



By the end of week 12 you should be able to:

1. identify the format of rules in regular grammars
2. generate a finite state recogniser (FSR) that corresponds to a given regular grammar and vice-versa
3. determine whether a sentence is valid for a given regular grammar
4. translate a non-deterministic FSR into a deterministic FSR using the subset construction algorithm
5. understand the limitations of regular grammars in representing sets of strings
6. identify the format of regular expressions
7. understand the connections between regular grammars, finite state recognisers and regular expressions

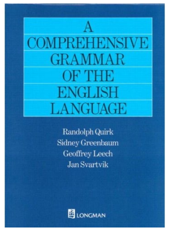
Phrase Structure Grammars

Reminder

- A set of *terminals* – the basic objects of the language
- A set of *non-terminals* – define the structure of valid sentences
- *start symbol* – starts all derivations
- A set of *production rules* – must be followed in order to generate valid strings

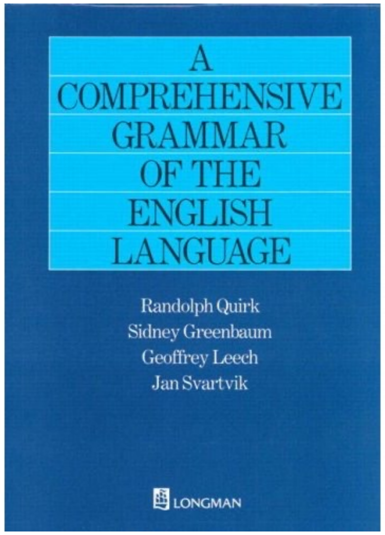
Classifying Grammars

- Throughout weeks 12, 13 and 14 we will be classifying different phrase structure grammars into 4 different types
  - Regular
  - Context free
  - Context sensitive
  - Unrestricted
- We also cover the abstract machines used to recognise them
- Today we will look at regular grammars



Regular grammars

Type 3



A COMPREHENSIVE GRAMMAR OF THE ENGLISH LANGUAGE

Randolph Quirk
Sidney Greenbaum
Geoffrey Leech
Jan Svartvik

LONGMAN

Type 3 - Regular Grammars

- **Regular grammars** generate regular languages
- This is the simplest form of grammar we will come across in this course
- All productions are one of two formats:
  - **NON-TERMINAL → terminal NON-TERMINAL**
  - **NON-TERMINAL → terminal**
- E.g.
  - **X → yZ** (X, Z non terminals, y is a terminal)
  - **X → t** (NB: t can be the empty string)

The integer/digit sequence grammar in BNF (week 11) was a regular grammar

A Simple Regular Grammar

$$\boxed{\begin{array}{l} S \rightarrow aA \\ A \rightarrow rB \\ B \rightarrow c \\ B \rightarrow m \end{array}}$$

- This grammar generates the strings *arc* and *arm*
- We can group productions together
  - $B \rightarrow c \mid m$
  - '|' means 'or'

Identifying a Regular Grammar

Is this a regular grammar?

$S \rightarrow aSYZ \mid aYZ$

$ZY \rightarrow YZ$

$aY \rightarrow ab$

$bY \rightarrow bb$

$bZ \rightarrow bc$

$cZ \rightarrow cc$

Not a Regular Grammar

Identifying a Regular Grammar

Are either of these two grammars regular?

$$\boxed{\begin{aligned} S &\rightarrow aB \mid bA \mid \varepsilon \\ A &\rightarrow aS \mid bAA \\ B &\rightarrow bS \mid aBB \end{aligned}} \longleftarrow \boxed{\text{Not a Regular Grammar}}$$

$$\boxed{\begin{aligned} S &\rightarrow aS \mid aB \mid bC \\ B &\rightarrow bC \\ C &\rightarrow cC \mid c \end{aligned}} \longleftarrow \boxed{\text{A Regular Grammar}}$$

Language of the Grammar

- the set containing each and every string of terminal symbols (***terminal string***) that can be generated by a grammar is called the ***language generated by the grammar***
- If the grammar is G, then the language generated by G is denoted by **L(G)**

L(G) Examples

- The language generated by these grammars can be written as set definitions

$$\boxed{\begin{array}{l} S \to aS \mid aB \mid bC \\ B \to bC \\ C \to cC \mid c \end{array}}$$

$$\downarrow$$

$$\boxed{\{a^i b c^j : i \geq 0,\ j \geq 1\}}$$

$$\boxed{\begin{array}{l} S \to aA \\ A \to aB \\ B \to aS \mid aC \\ C \to bD \\ D \to b \mid bC \end{array}}$$

$$\downarrow$$

$$\boxed{\{a^{3i} b^{2j} : i, j \geq 1\}}$$

Derivation and Parsing

Reminder

- Derivation – begin with the start symbol and work through the rules until we have a valid sentence
- Parsing – begin with the sentence and work back to the start symbol to determine if it is valid according to the grammar
- We are often more interested in parsing

Parsing with a Regular Grammar

- How do we show that *abbbac* is grammatical and *abbbbc* is not, according to the following grammar (G_1):

$$
\boxed{
\begin{array}{l}
S \rightarrow aX \mid bY \\
X \rightarrow bX \mid aZ \\
Y \rightarrow aY \mid bZ \\
Z \rightarrow c \qquad\qquad G_1
\end{array}
}
$$

Diagram – Non-terminals

- We can think of the non-terminals as **states**
  - state of being about to write the non-terminal
- These are usually represented as circles

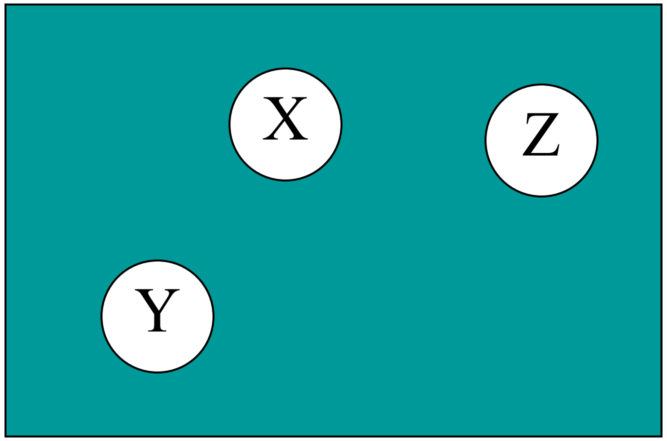


Diagram – Start Symbol

- The start symbol is the state of not having looked at the string at all yet
- Shown as a circle with an unmarked arrow

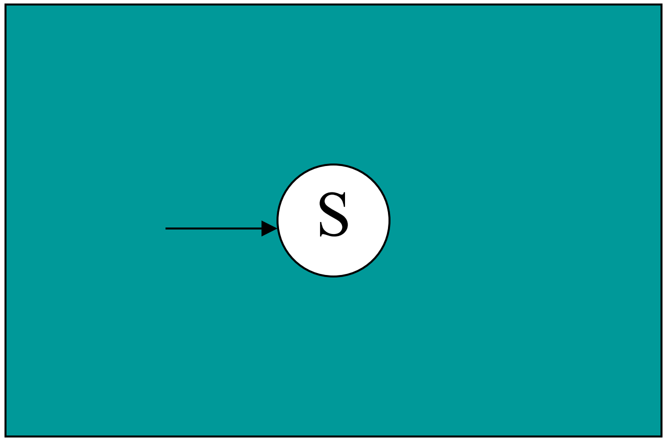


Diagram - Terminals

- Terminals are used to jump between states
- Marked on the diagram on the joining arrow – e.g. X $\rightarrow$ bX | aZ

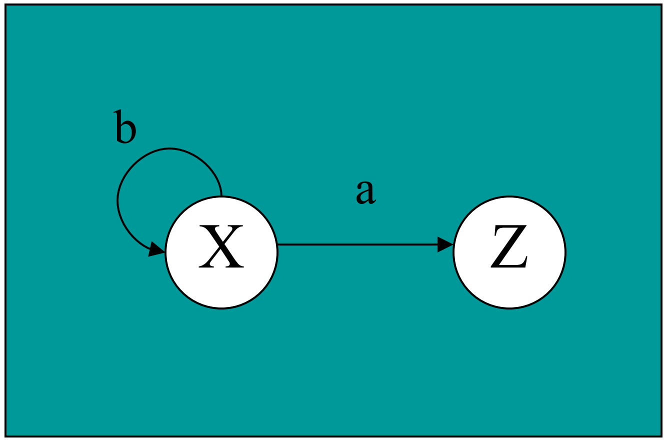


Diagram – Final State

- a single terminal shows we have come to the end of this part of the string
  - e.g $Z \rightarrow c$
- This is the **halt** or **final** state represented as a square

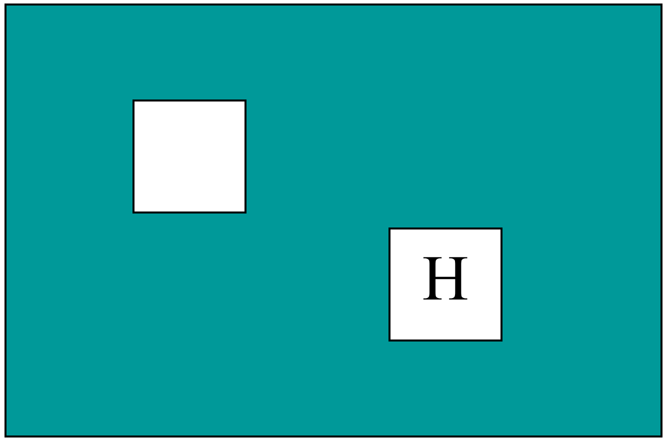


Diagram for a Regular Grammar

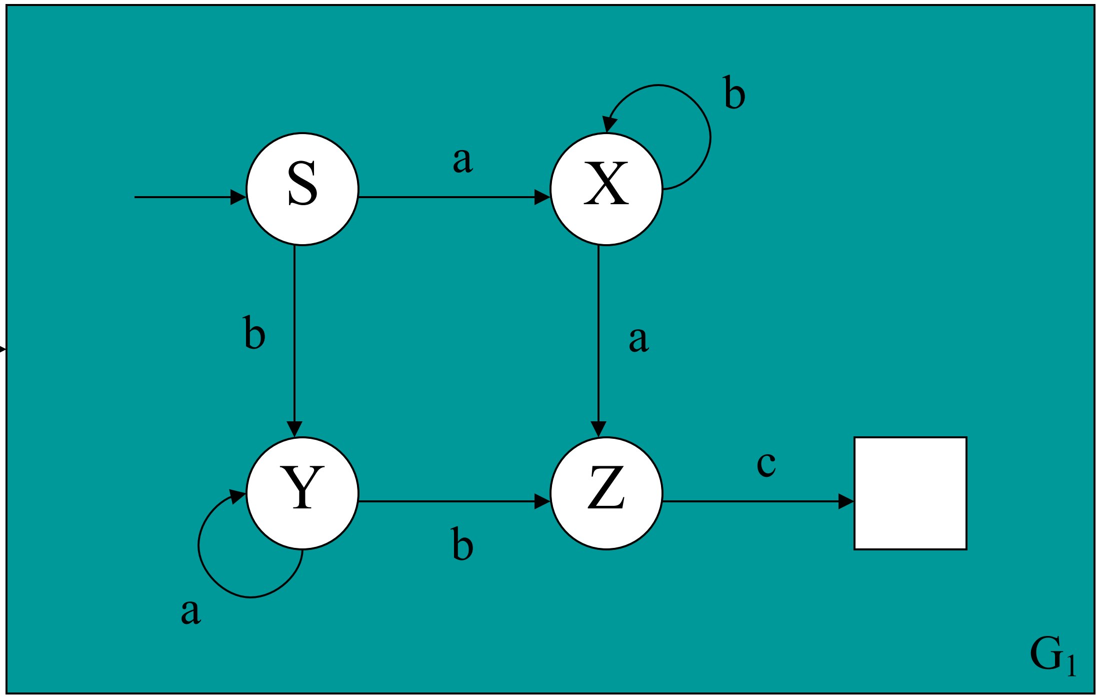
$S \rightarrow aX \mid bY$

$X \rightarrow bX \mid aZ$

$Y \rightarrow aY \mid bZ$

$Z \rightarrow c$

G_1

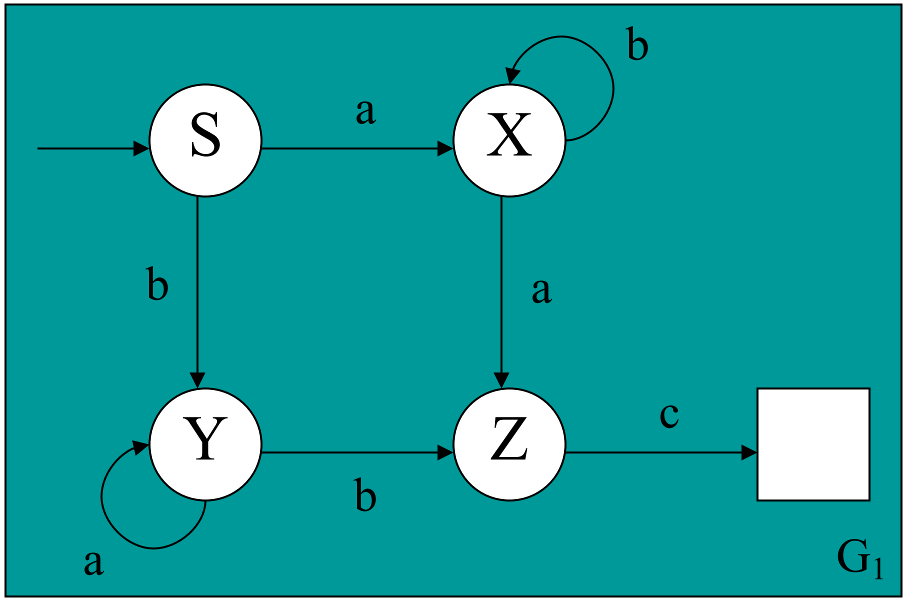


Parsing with the Diagram

- The string is only grammatical according to the grammar if we start at the start symbol and then end up in the halt state having read all the symbols.
- If we are ever in a state where there is no arc labelled with the next symbol from the string, then the string is not grammatical.

Parsing with the Diagram

- We can use the diagram of the grammar to parse the sentences *abbbac* and *abbbbc*



Finite State Recogniser

- The diagram we have just drawn represents a primitive computer called a **Finite State Recogniser** (FSR)
- It may also by called a Finite State Machine (FSM) or a Finite State Automaton (FSA)
- Called ‘finite state’ because there are only a finite number of states in the machine

Finite State Recogniser

Type 3

Finite State Recogniser

- Consists of:
  - A set of states
  - One state designated the start state
  - One state designated the finish state
  - A set of arcs from one state to another each labelled with a symbol from the input alphabet
- A finite state recogniser does exactly the same job as a regular grammar

Finite State Recogniser

- It can remember things about the past only by being in one state rather than another
- Can remember only a finite number of different things about the past,
  - for example, about the earlier parts of the string it is looking at
  - This means that counting is restricted

A Finite State Recogniser for G_2

$S \rightarrow aA$

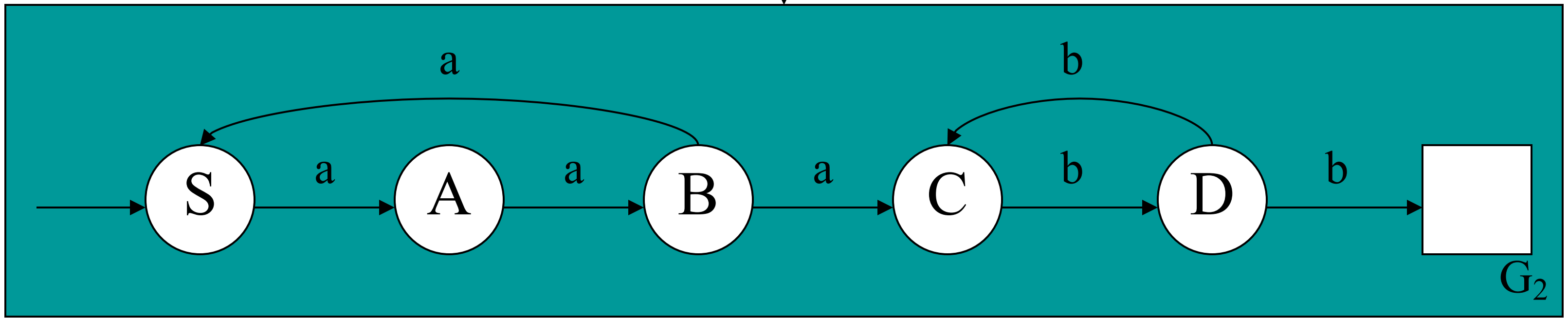
$A \rightarrow aB$

$B \rightarrow aS \mid aC$

$C \rightarrow bD$

$D \rightarrow b \mid bC$

G_2

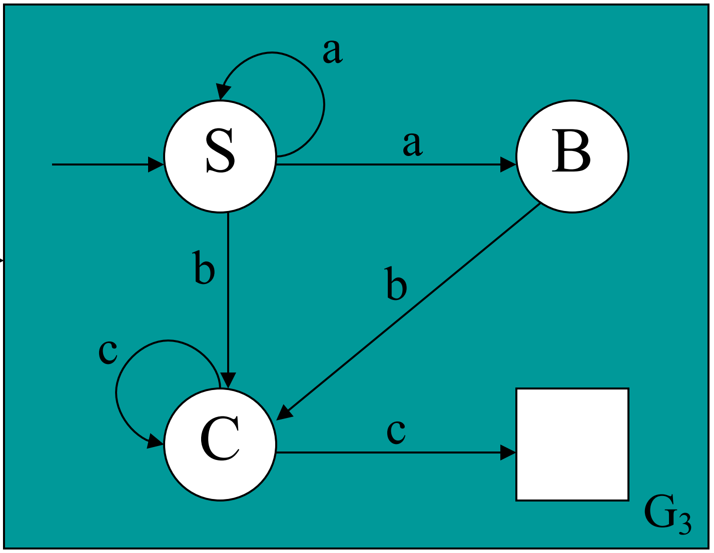


A Finite State Recogniser for G_3

- Every regular grammar has an **equivalent** finite state recogniser

$$S \rightarrow aS \mid aB \mid bC$$
$$B \rightarrow bC$$
$$C \rightarrow cC \mid c$$

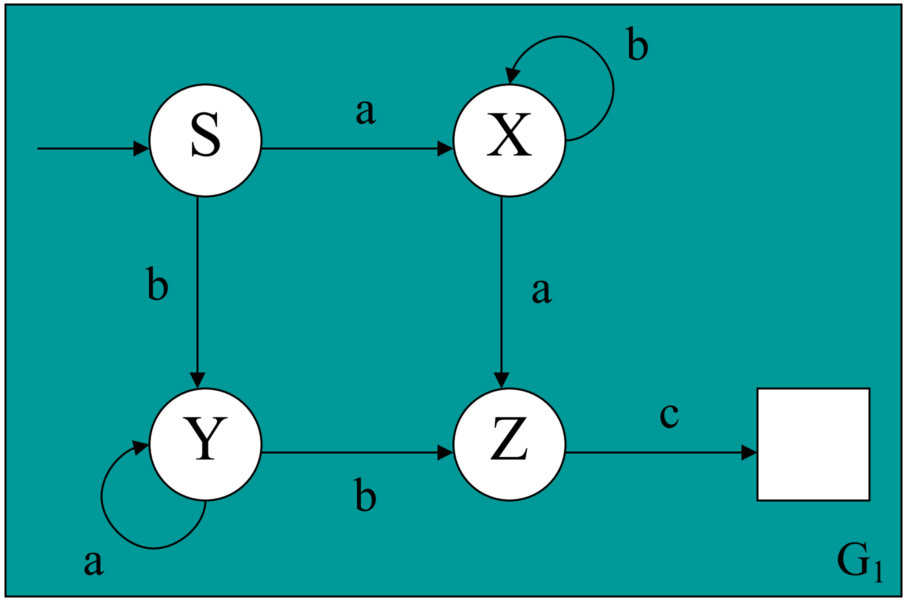
G_3



Other Representations

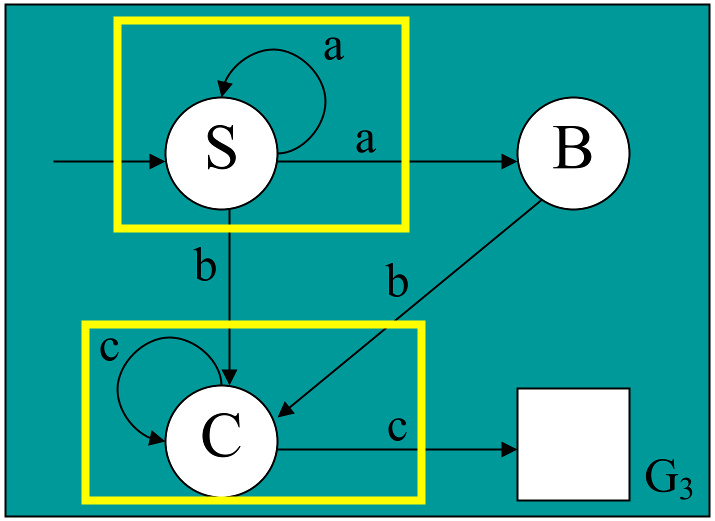
Remember Graphs in SCC120?

- Finite state recognisers can be represented as state transition tables as well as graphs

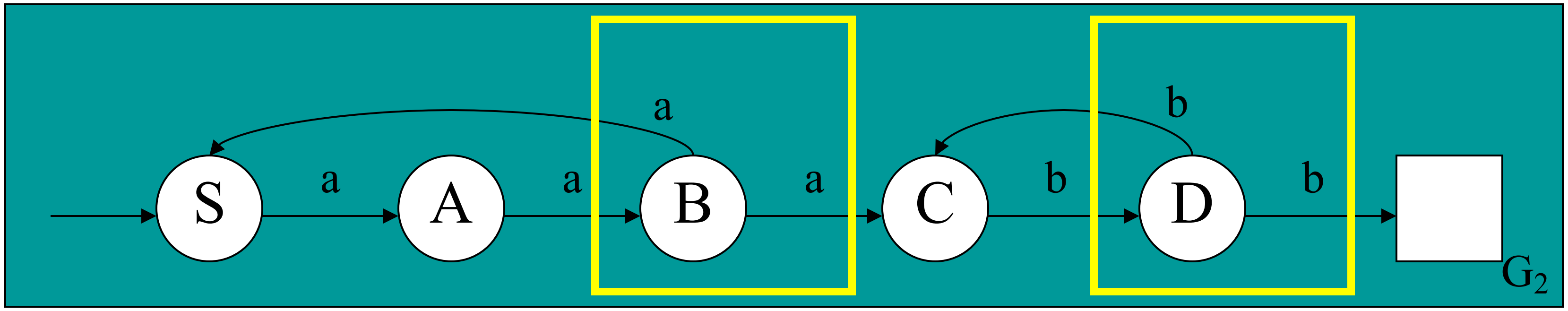


	a	**b**	**c**
S	X	Y	
X	Z	X	
Y	Y	Z	
Z			*F*

Non-Deterministic FSRs



- Both examples feature states in which a choice of transition must be made given a certain terminal
- Called **non-determinism**



Deterministic FSRs

- We do not want to use non-deterministic finite state recognisers
  - We have to try each alternative in turn
  - This could lead to further alternatives
  - Could be a lengthy procedure
- We need to find a **deterministic** FSR
  - These do not feature states where a choice of transition is needed

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