SCC.312 Languages and Compilation (Week 14)

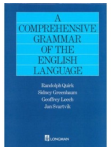
Paul Rayson

p.rayson@lancaster.ac.uk

Last Week

Reminder

- The “Repeat State” Theorem



A COMPREHENSIVE GRAMMAR OF THE ENGLISH LANGUAGE

Randolph Quirk
Sidney Greenbaum
Geoffrey Leech
Jan Svartvik

LONGMAN

- Context free Grammars

- Pushdown Recognisers

- Uses of Context Free Grammars
- Syntax, Semantics and Ambiguity

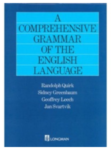
Chomsky Hierarchy

Type	Grammar	Machine	Other Equivalent
2	Context Free	Push Down Recogniser	
3	Regular	Finite State Recogniser	Regular Expressions

Non-deterministic and deterministic

This week’s Topics

- “*uvwxy*” Theorem



A COMPREHENSIVE GRAMMAR OF THE ENGLISH LANGUAGE

Randolph Quirk
Sidney Greenbaum
Geoffrey Leech
Jan Svartvik

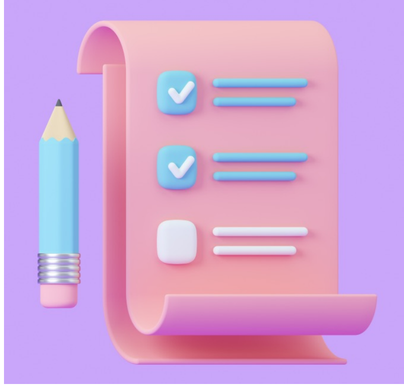
LONGMAN

- Context sensitive & unrestricted grammars

- The Turing Machine
  - Linear Bounded

- The Complete Chomsky Hierarchy

Learning Objectives



1. recognise context sensitive and unrestricted phrase structure grammars
2. design and understand the behaviour of Turing machines (TM)
3. understand the connection between Turing machines and unrestricted and context sensitive grammars
4. understand the order and relations of phrase structure grammars in the Chomsky Hierarchy

Context Free Grammar Reminder

Reminder

- The productions have the format
  - **X → RHS**
    - **X** is a single non-terminal,
    - **RHS** (right hand side) is any mixture of terminals and/or non-terminals, and can be empty
- They can be represented with a pushdown recogniser (PDR)

A Context Free Derivation Tree

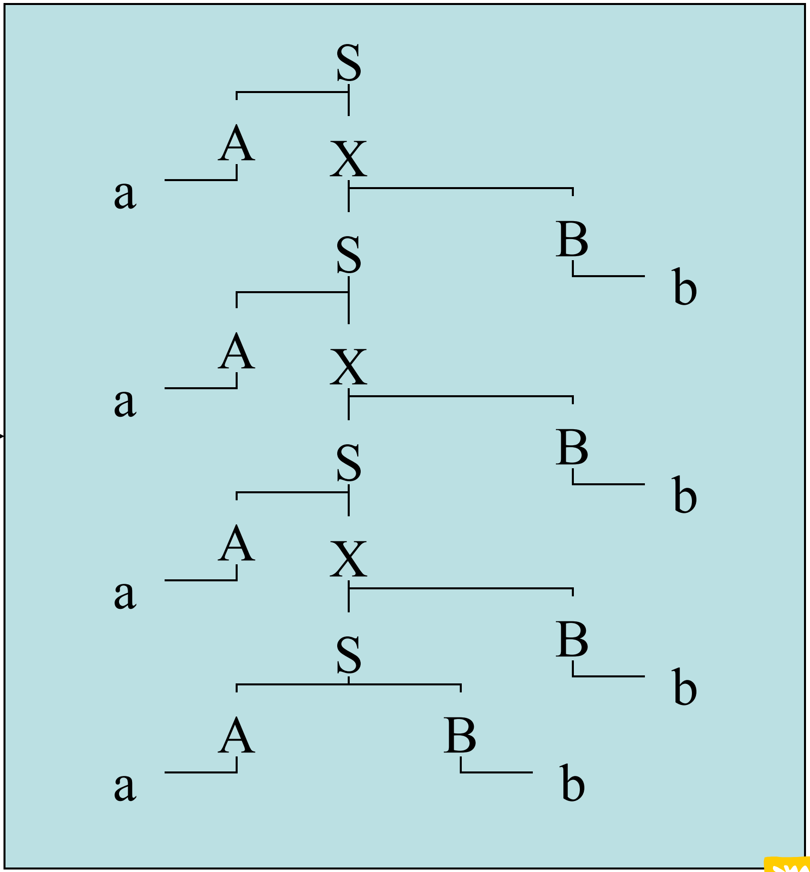
- The derivation of a sufficiently long string (e.g. aaaabbbb) must have at least one repeated non-terminal

$S \to AX \mid AB$

$X \to SB$

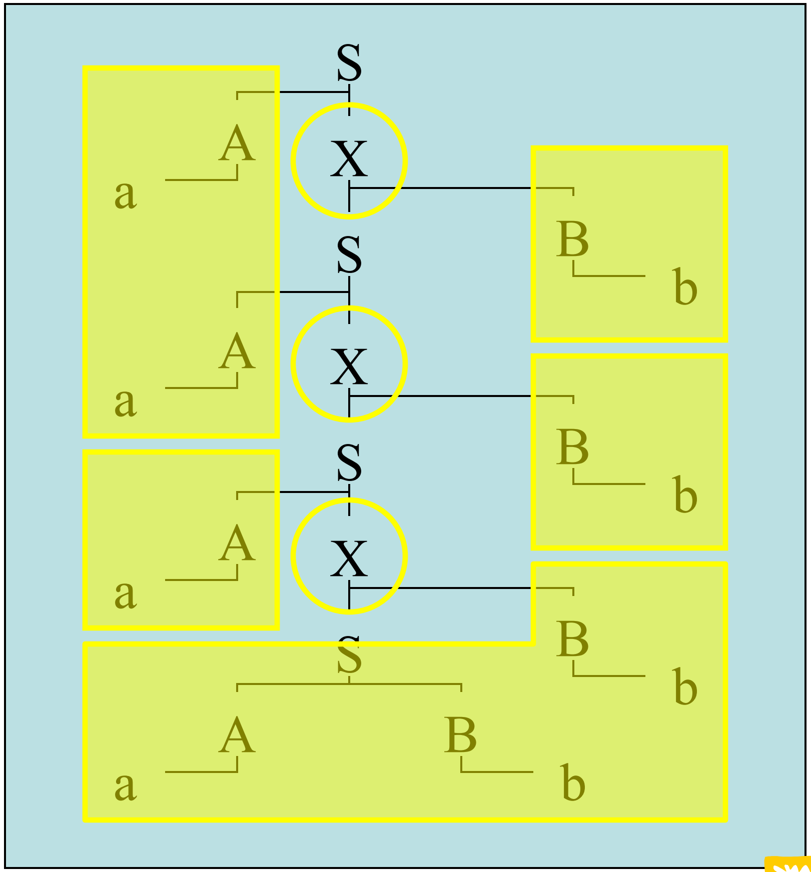
$A \to a$

$B \to b$



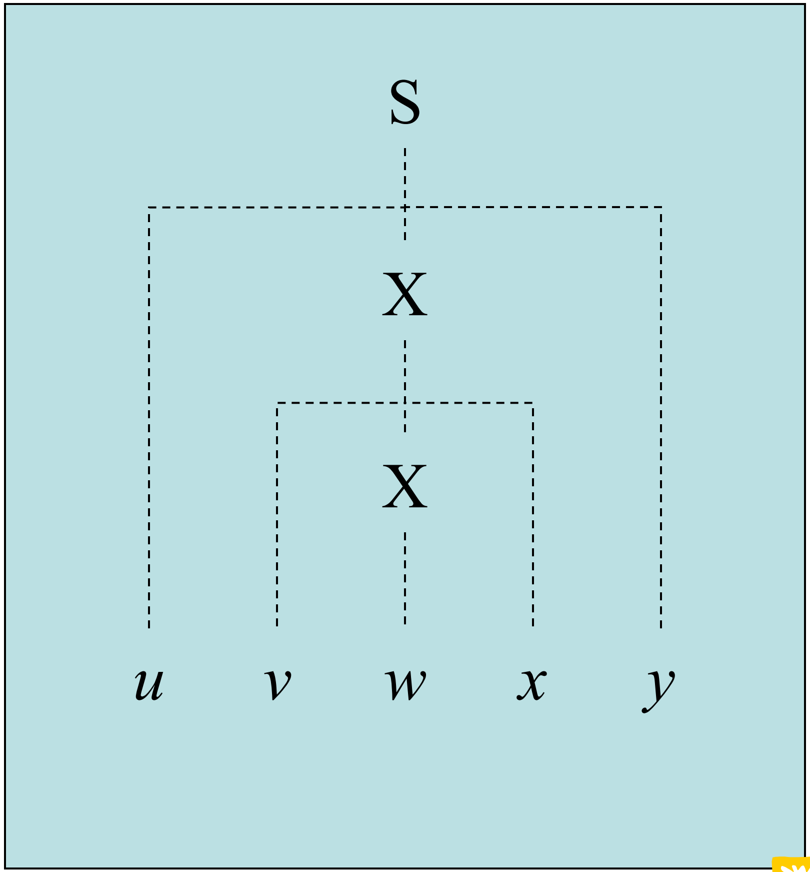
A Context Free Derivation Tree

- We select a repeated non-terminal on the main derivation path (in this case X)
- We can see that the tree is made up of 5 concatenated sub strings: *aa*, *a*, *abb*, *b*, *b*



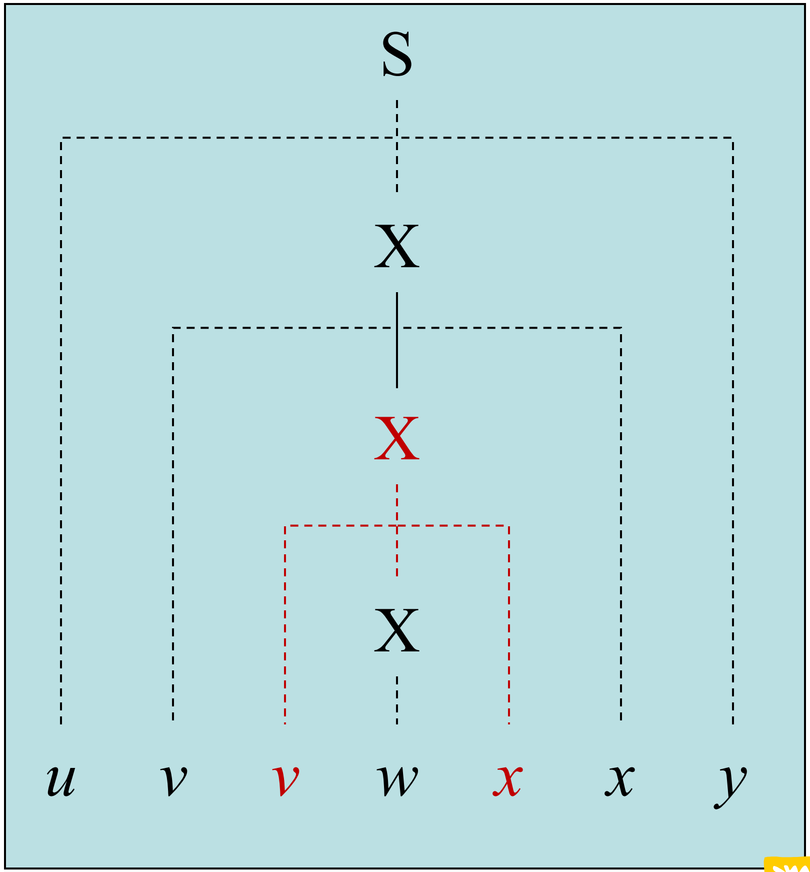
A Formal CF Derivation Tree

- The diagram shows the formal derivation of any context free grammar into the 5 concatenated sub strings: *uvwxy*



A Formal CF Derivation Tree

- If we repeat the non-terminal X then we repeat part of our string: uv^2wx^2y
- We can repeat this process indefinitely giving us uv^iwx^iy for all i ≥ 0



Homework: have a look at the "pumping lemma"

"*uvwxy*" Theorem

- The string that is accepted can be expressed as 5 concatenated sub strings: *uvwxy*
  - *u* and *y* are the strings accepted before and after the repeated states. *They may be empty (ε)*
  - *v* and *x* are the repeated strings *(non empty)*
  - *w* is the string accepted between *v* and *x* *(non empty)*
- uv^iwx^iy for all $i \geq 0$
- Applies to infinite context free languages

Remember: u, v, w, x, y are not symbols in the language, they represent substrings

v and x are non-empty but might occur with power 0 i.e. 0 times

Example

- "Applies to any infinite context free language"
- uv^iwx^iy for all i ≥ 0
- e.g. a^nbc^nde for n ≥ 1
  - u = ε
  - v = a
  - w = abc (remember w is non empty)
  - x = c
  - y = de

Using the “*uvwxy*” Theorem

- We can use the theorem to determine if a language is context free
- $\{a^n b^n c^n : n \geq 1\}$ looks like it might be
- The “*uvwxy*” Theorem says we need 5 concatenated sub strings for uv^iwx^iy
  - What is *v*? *ab*? *a*? *b*?
  - What is *x*? *bc*? *b*? *c*?
- $\{a^n b^n c^n : n \geq 1\}$ cannot be context free

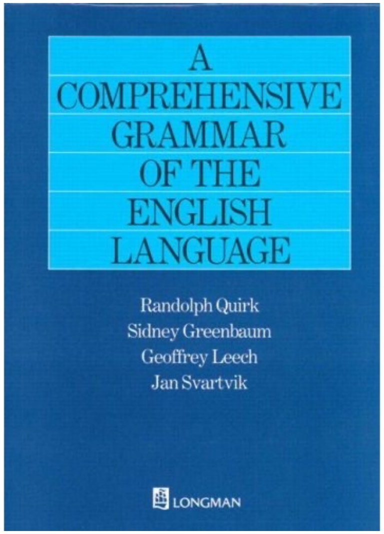
$a^n(bc)^n$
Try this one later, or in the lab. Is it
context free using this theorem? If
yes, can you make the PDR?

Beyond Context Free Languages

- The following languages are also non-context free
  - $\{a^i b^j c^{i*j} : i, j \geq 1\}$
  - {xx : x is any string of *a*'s and or *b*'s}
  - there are infinitely more such languages
- Just as context free grammars are more powerful than regular ones, there are more powerful grammars than context free

Context Sensitive

Type 1



A COMPREHENSIVE GRAMMAR OF THE ENGLISH LANGUAGE

Randolph Quirk
Sidney Greenbaum
Geoffrey Leech
Jan Svartvik

LONGMAN

Type 1 – Context Sensitive

- The next grammar is context sensitive (CS)
- The productions have the format
  - **LHS** → **RHS**
    - **LHS** is any mixture of terminals and/or non-terminal, but there must be at least one symbol (| **LHS** | ≥ 1)
    - **RHS** is any mixture of terminals and/or non-terminals, but **cannot** be the empty string (ε)
    - | **LHS** | ≤ | **RHS** |
      (**RHS** is at least as long as **LHS**, if not longer)

A Context Sensitive Grammar

- $\{a^i b^i c^i : i \geq 1\}$ can be represented as the following context sensitive grammar (G_{10})

$$
\begin{aligned}
S &\rightarrow aSYZ \mid aYZ \\
ZY &\rightarrow YZ \\
aY &\rightarrow ab \\
bY &\rightarrow bb \\
bZ &\rightarrow bc \\
cZ &\rightarrow cc
\end{aligned}
$$

G_{10}

A CS Derivation for G_{10}

- CS strings grow as they are derived:

$S \rightarrow aSYZ \mid aYZ$

$ZY \rightarrow YZ$

$aY \rightarrow ab$

$bY \rightarrow bb$

$bZ \rightarrow bc$

$cZ \rightarrow cc$

G_{10}

1. **S**
2. a**S**YZ
3. aa**S**YZYZ
4. aaaY**ZY**ZYZ
5. aaaYYZ**ZY**Z
6. aaaYY**ZY**ZZ
7. aa**aY**YYZZZ
8. aaa**bY**YZZZ
9. aaab**bY**ZZZ
10. aaabb**bZ**ZZ
11. aaabbb**cZ**Z
12. aaabbbc**cZ**
13. aaabbbccc

An Alternative CS Definition

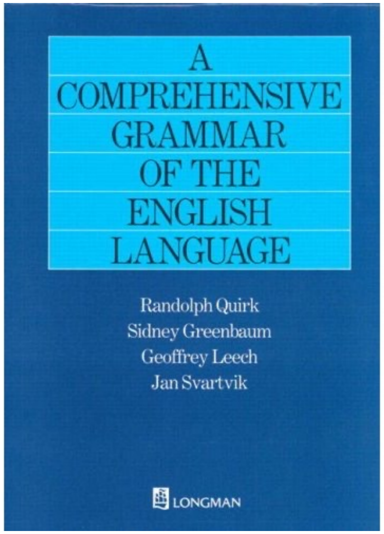
- All productions are of the form:
  - $\alpha_1 X \alpha_2 \rightarrow \alpha_1 \beta \alpha_2$
- You can write the non-terminal, X, as the non-empty string β, but only in the context of string α_1 on the left and α_2 on the right
  - Either α_1 or α_2 could be empty (ε)
    - If both are empty it's a context free rule
  - β cannot be empty (ε)

Beyond Context Sensitive

- The two CS definitions are equivalent. One can be turned into the other
- It can be shown that there are languages that cannot be represented by context sensitive grammars
  - Most reasonable languages are context sensitive
  - Non-CS ones are “a bit weird”

Unrestricted Grammars

Type 0



A COMPREHENSIVE GRAMMAR OF THE ENGLISH LANGUAGE

Randolph Quirk
Sidney Greenbaum
Geoffrey Leech
Jan Svartvik

LONGMAN

My clipart logos are not the official symbols!!

Type 0 – Unrestricted Grammars

- The next type is unrestricted grammars
- The productions have the format
  - **LHS** → **RHS**
    - **LHS** is any mixture of terminals and/or non-terminals, but there must be at least one symbol (| **LHS** | ≥ 1)
    - **RHS** is any mixture of terminals and non-terminals, and **can** be the empty string (ε)
    - There is no length restriction on **RHS**

These last two conditions differ from type 1 grammars

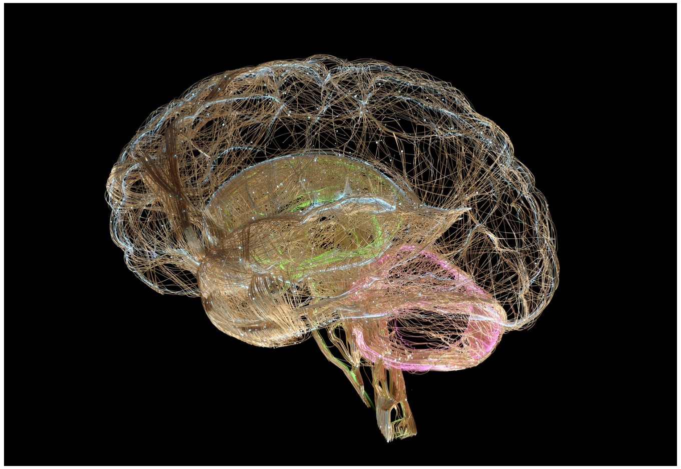
An Unrestricted Grammar G_{11}

- $\{a^i b^j c^{i * j} : i, j \geq 1\}$ can be represented as the following unrestricted grammar (G_{11})

$S \rightarrow AS \mid AB$	$Nc \rightarrow Mcc$
$B \rightarrow BB \mid C$	$XMBB \rightarrow BXNB$
$AB \rightarrow HXNB$	$XBMc \rightarrow Bc$
$NB \rightarrow BN$	$AH \rightarrow HA$
$BM \rightarrow MB$	$H \rightarrow a$
$NC \rightarrow Mc$	$B \rightarrow b$ G_{11}

Parsing a CS Language

- What sort of machine could we use to parse a context sensitive (or unrestricted) language?
- Obviously finite state recognisers and pushdown recognisers are not powerful enough



The Turing Machine

The Turing Machine

- The Turing Machine (TM) successively reads symbols from an input string, called the **tape**, and changes between states, like a finite state recogniser
- It can write onto the tape and read it back
  - Potentially, it has an infinite amount of storage like the push down recogniser
  - It has unrestricted access

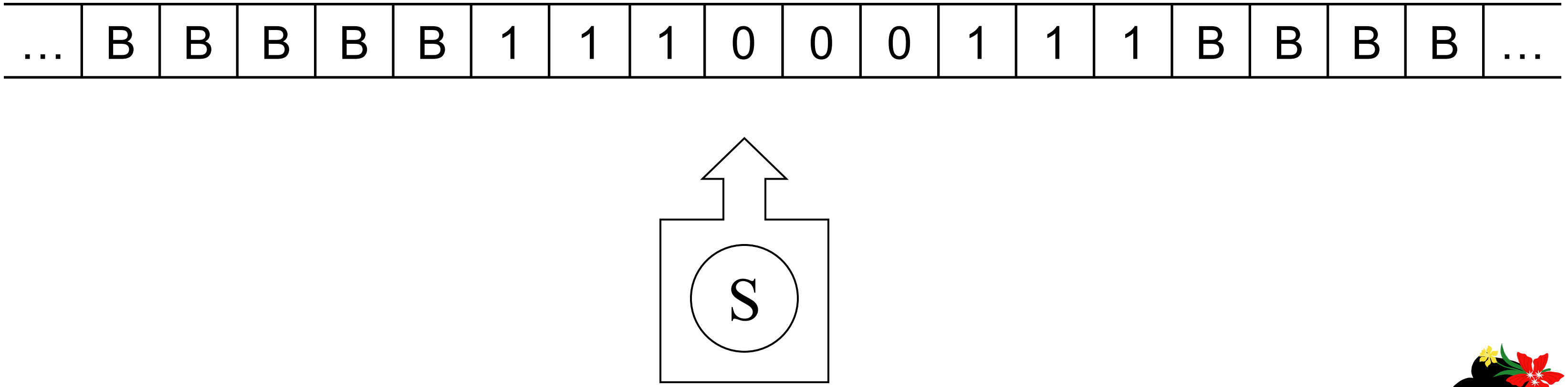
The Turing Machine Tape

- The tape is infinite in both directions and divided into squares
- Each square contains a symbol from the alphabet of the language
  - Most contain the **blank symbol**, B (or blank square in the book: Parkes, p135)

…	B	B	B	B	B	1	1	1	0	0	0	1	1	1	B	B	B	B	…

The Turing Machine R/W Head

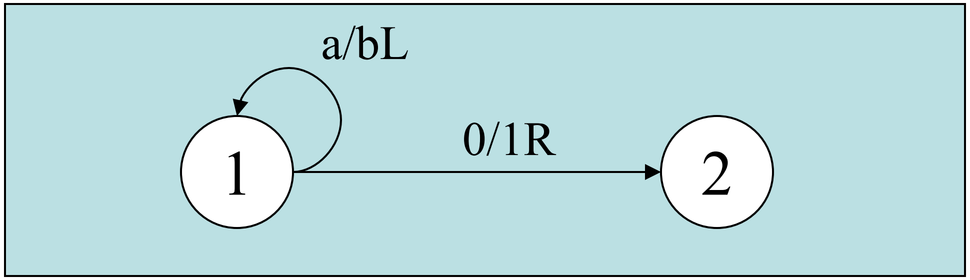
- The Turing machine has a **read-write head**
- This can look at a single square of the tape at any one time
  - – Also shows which state it is currently in



…	B	B	B	B	B	1	1	1	0	0	0	1	1	1	B	B	B	B	…

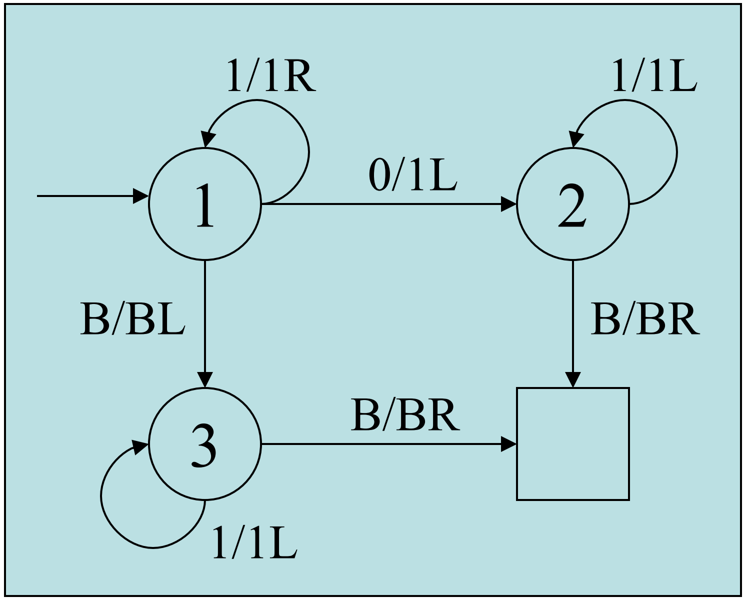
The Turing Machine Instructions

- The behaviour of the machine can be specified in a similar way to FSRs
- The arc is labelled with:
  - The symbol read from the input tape square
  - The new symbol to write to the square
  - The direction to move the r/w head (Left or Right)

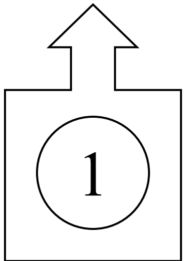


An Example Turing Machine

- Start by putting the Turing machine in a particular state
- The r/w head starts at a place on the tape
  - e.g. leftmost non-B symbol

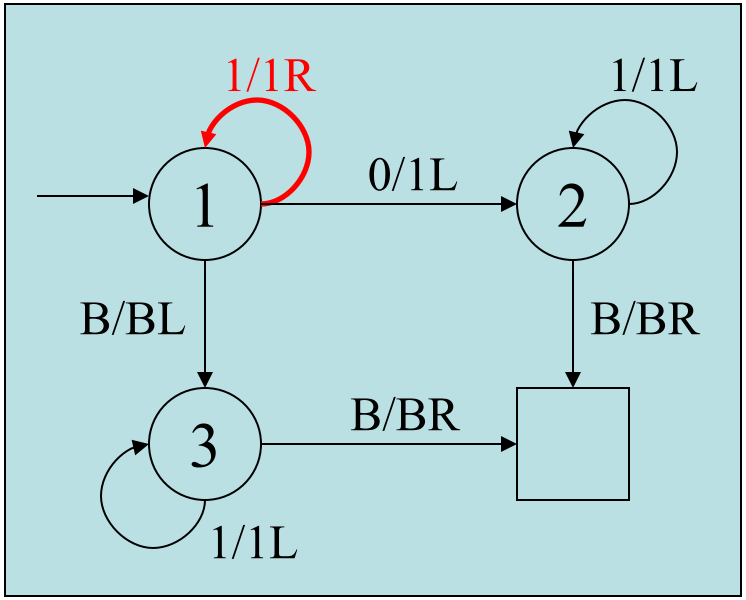


…	B	B	B	B	B	1	1	1	0	0	0	1	1	1	B	B	B	B	…

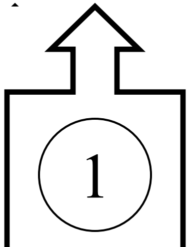


The Turing Machine in Action

- For the first 3 moves we stay in state 1
  - read and write a 1 to the tape
  - move the r/w head to the right

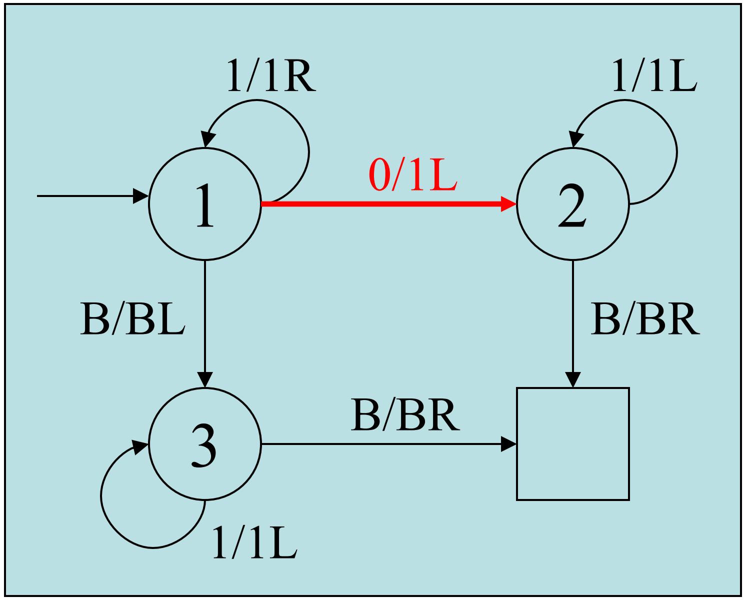


…	B	B	B	B	B	1	1	1	0	0	0	1	1	1	B	B	B	B	…

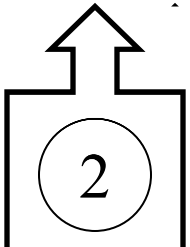


The Turing Machine in Action

- For the 4th move the machine reads a 0 on the tape and replaces it with 1
- It moves the r/w head to the left

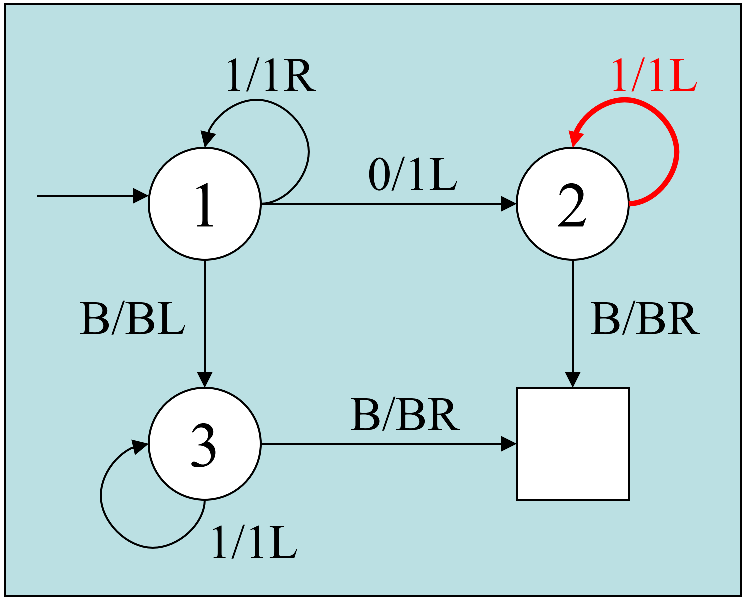


…	B	B	B	B	B	1	1	1	1	0	0	1	1	1	B	B	B	B	…

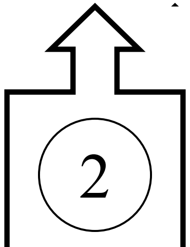


The Turing Machine in Action

- For the 5th move the machine reads a 1 on the tape and writes a 1 back to it
- The r/w head moves to the left each time

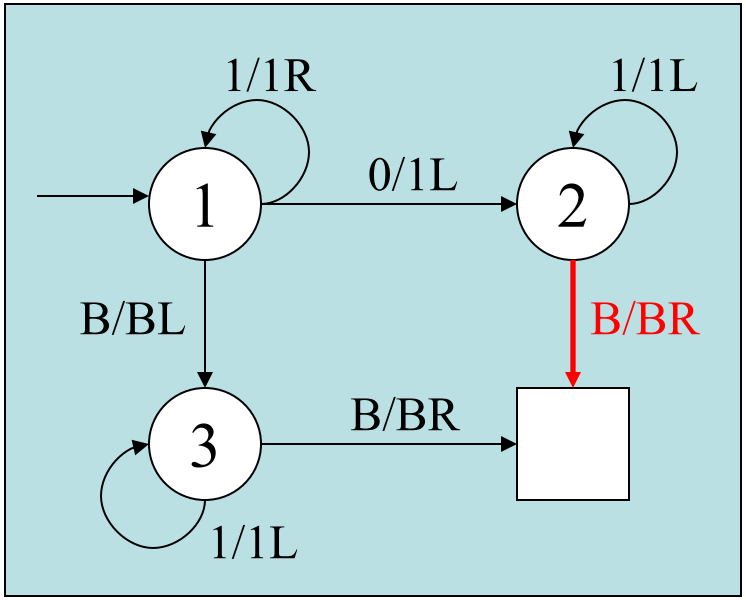


…	B	B	B	B	B	1	1	1	1	0	0	1	1	1	B	B	B	B	…

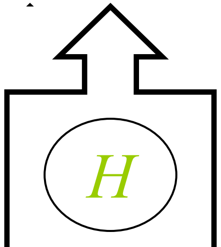


The Turing Machine in Action

- Finally, for the 6th move the machine reads a B on the tape, writes a B back to it and moves the r/w head to the right



…	B	B	B	B	B	1	1	1	1	0	0	1	1	1	B	B	B	B	…

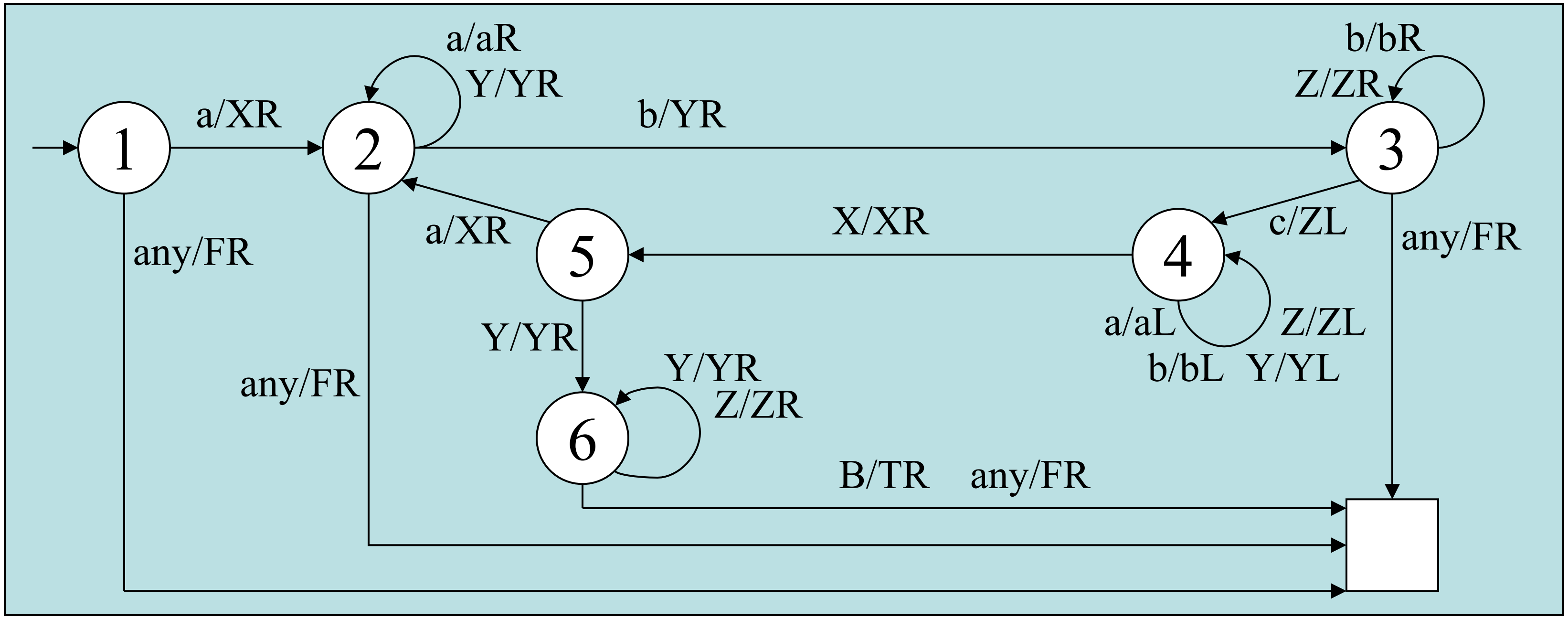


The Turing Machine

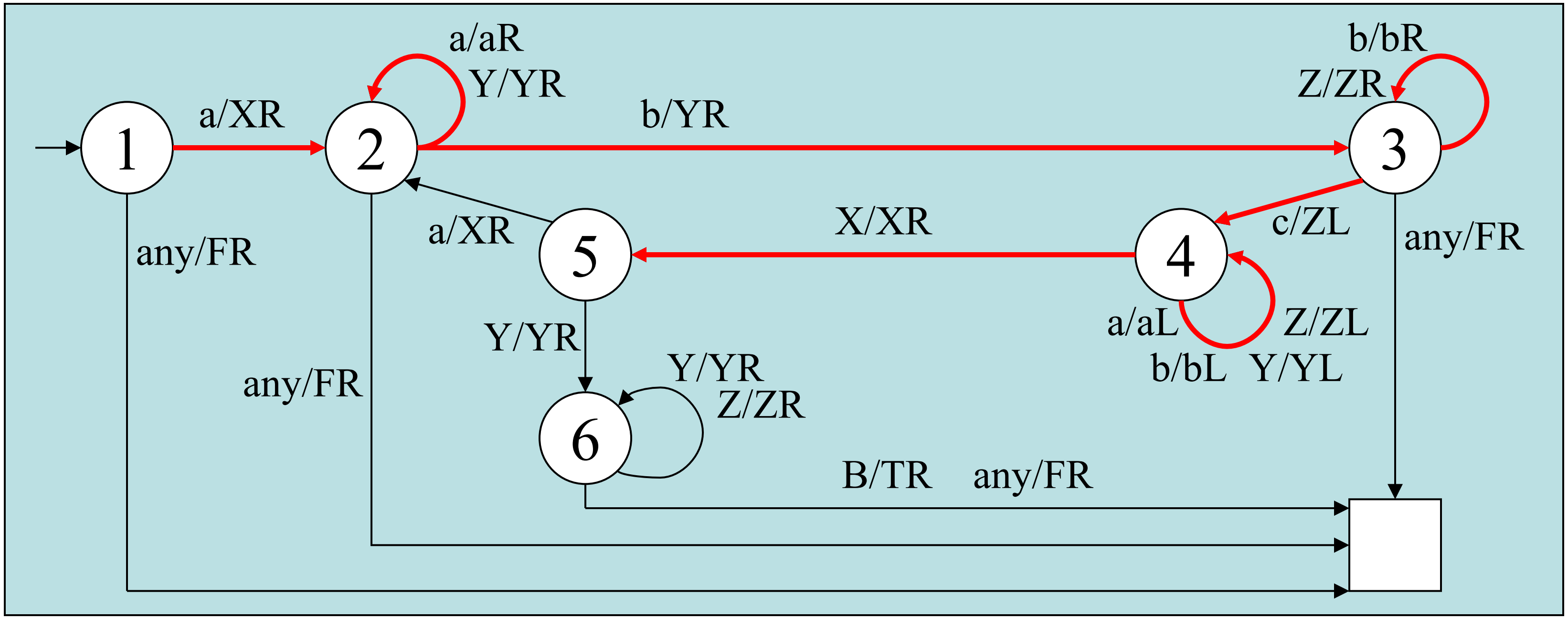
- The machine stops when it reaches the halt state (marked in the usual way)
- Our example machine has rewritten the left most 0 as a 1 and returned to the start
- If there is no left most 0 it keeps going right until it reads a B and then returns to the start
  - If this was not included the machine would move right indefinitely and never halt

A TM to Recognise $a^ib^ic^i$ (G$_{10}$)

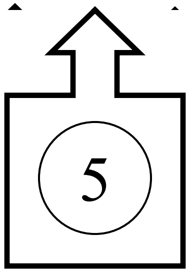
- This TM will recognise $\{a^ib^ic^i : i \geq 1\}$
  - (simplified) 'any' means 'any other symbol'



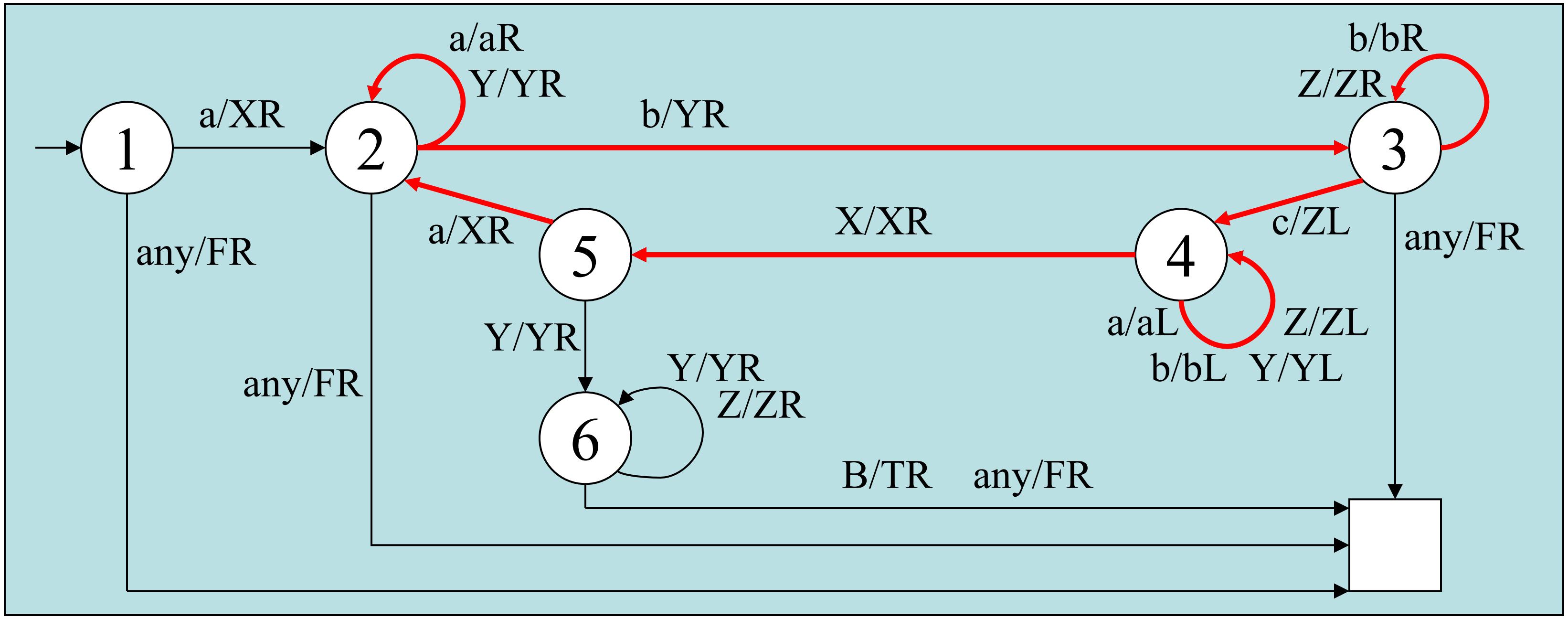
A TM to Recognise $a^ib^ic^i$ (G_{10})



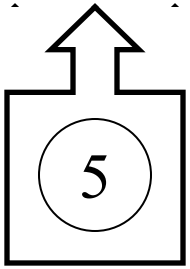
…	B	B	B	X	a	a	Y	b	b	Z	c	c	B	B	B	B	B	B	…



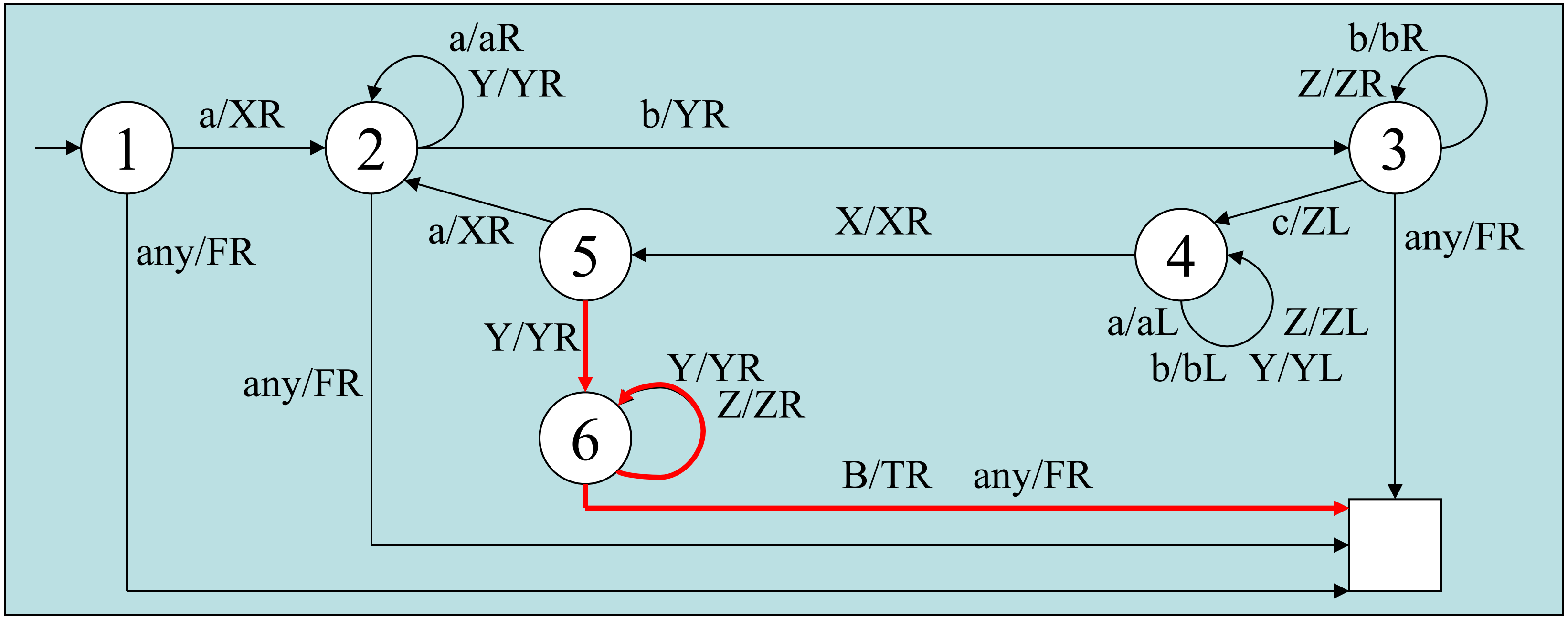
A TM to Recognise $a^i b^i c^i$ (G_{10})



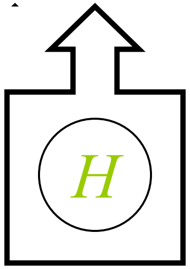
…	B	B	B	X	X	X	Y	Y	Y	Z	Z	Z	B	B	B	B	B	B	…



A TM to Recognise $a^ib^ic^i$ (G_{10})



…	B	B	B	X	X	X	Y	Y	Y	Z	Z	Z	T	B	B	B	B	B	…

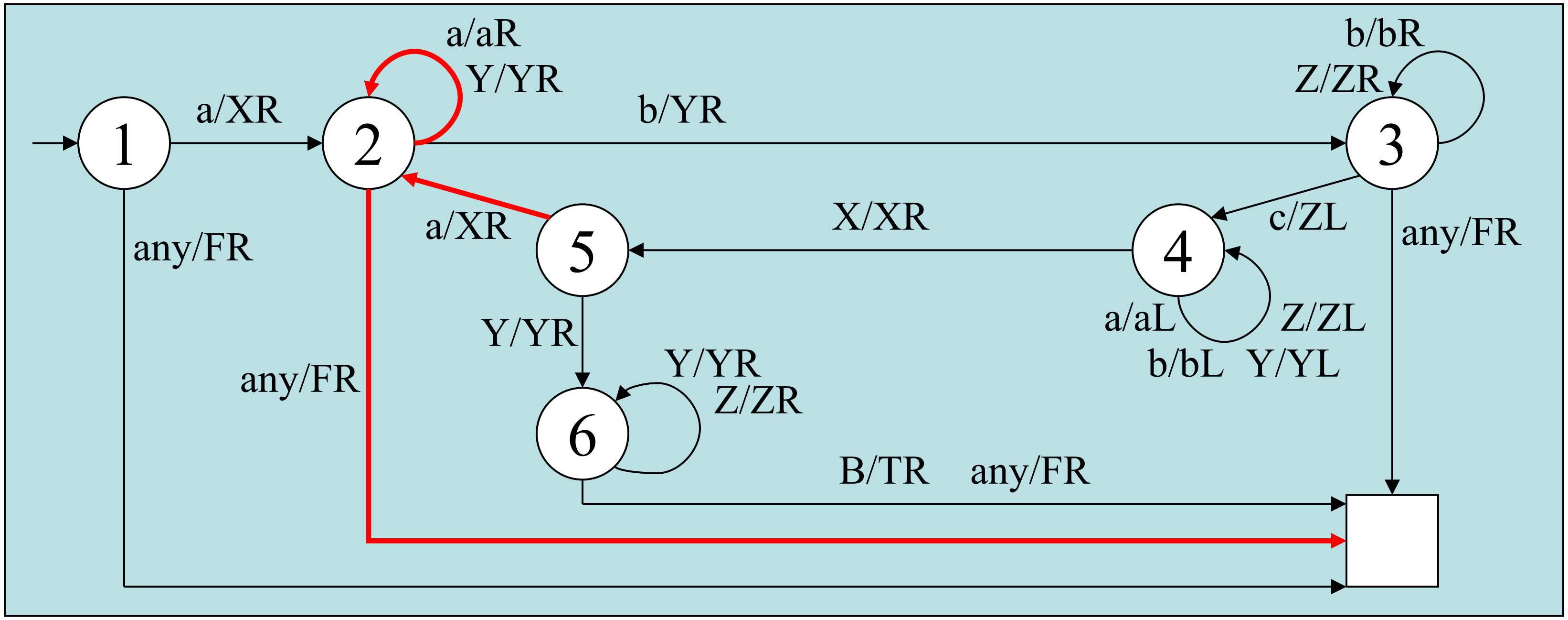


A TM to Recognise $a^i b^i c^i$ (G_{10})

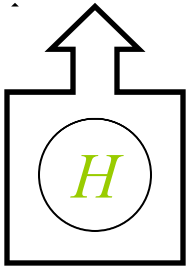
Success!

- For the valid string *aaabbbccc* the machine write T (for true) and halts
- Now lets see what happens when we try an invalid string: *aaabbccc*
- The machine will recognise the first two *a*’s, *b*’s and *c*’s as before…

A TM to Recognise $a^ib^ic^i$ (G_{10})



…	B	B	B	X	X	X	Y	Y	F	Z	c	B	B	B	B	B	B	B	…



A TM to Recognise $a^i b^i c^i$ (G_{10})

Failure!

- For invalid strings the machine writes an F (for false) and halts
  - The same occurs for other invalid strings
- It is possible to add a state so that the TM halts with the r/w head over the T or F

The Turing Machine

- It is typical for a TM to scan backwards and forwards along the tape for several iterations
- The machine remembers what to write by being in one state rather than another
  - The machine marks where it has got to, so that it can return there later
  - Often this is by changing symbols to X, Y, etc.

Linear Bounded Turing Machines

- The TM we have just used only requires the space on the tape taken by the input string
- For this reason it is called a **Linear Bounded Turing Machine**
  - It is restricted to the space required for the input string plus a fixed number of extra squares

Turing Machines and Grammars

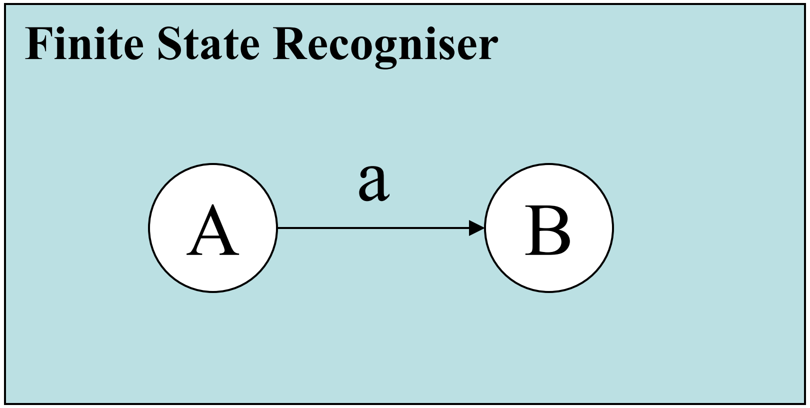
- A Linear Bounded TM is all we need for context sensitive grammars
  - In a CS grammar $|L| \leq |R|$ for $\mathbf{L} \rightarrow \mathbf{R}$
  - The derivation never gets shorter
  - Keep generating longer strings until we get the string we are trying to parse
- For unrestricted grammars we may require a TM with unbounded extra space on the tape

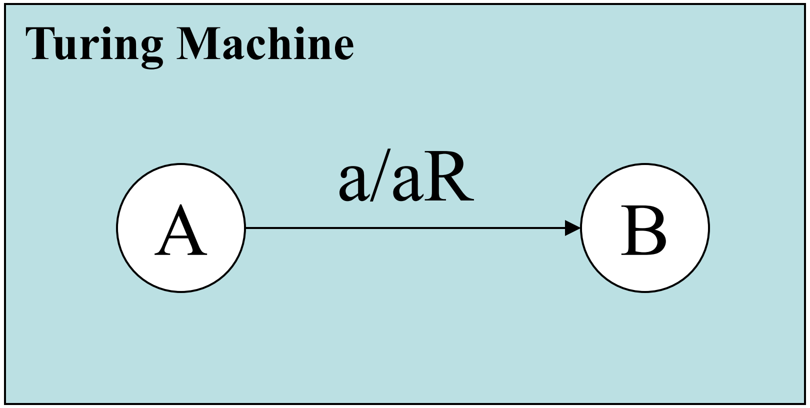
TMs as Language Processors

- TMs can be used as language recognisers
- If fact Turing machines can be used to recognise any phrase structure grammar

Turing Machines v. FSRs

- TMs are more powerful than FSRs
- A FSR is simply a TM that always moves its r/w head to the right replacing each symbol it reads by the same symbol



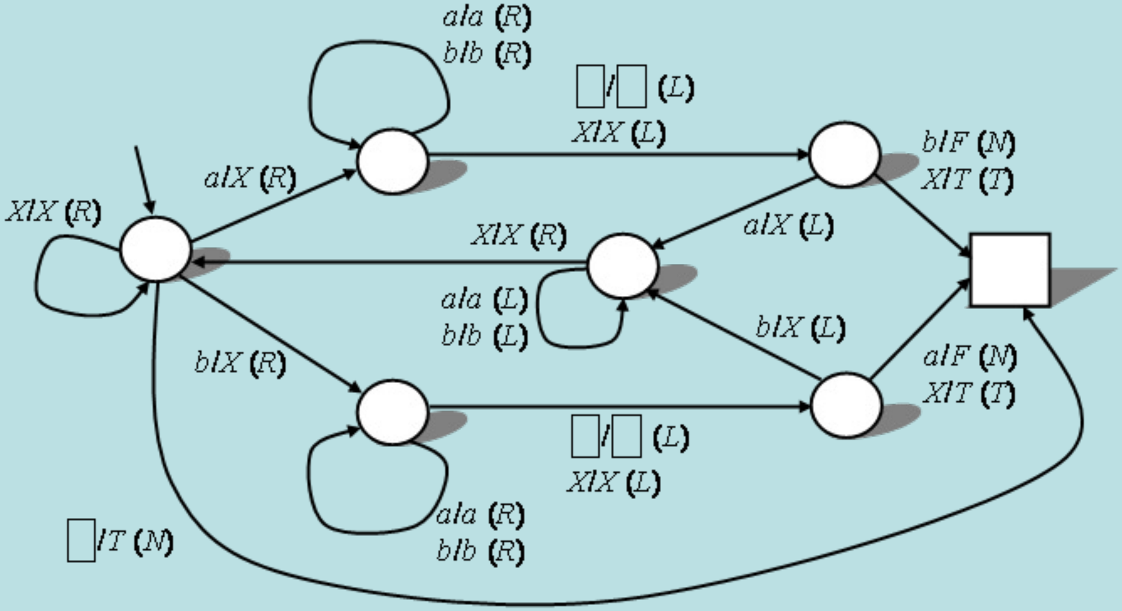


Turing Machines v. PDRs

- TMs are more powerful than PDRs, even non-deterministic PDRs

Example from Parkes book (B = blank square)

Deterministic TM for arbitrary palindrome strings of *a*’s and *b*’s



Prints T for valid strings, F for invalid strings

Lab exercise: try building this in JFlap

Different Types of TM

- Various alterations that can be made to a TM
  - A tape infinite in one direction only
  - Several tapes and r/w heads
  - A non-deterministic TM
- However, any non-standard TM can be modelled by a functionally equivalent standard TM
  - they do not give us more or less power
  - they are different in efficiency and complexity

Alternative Models

We won't be covering these in SCC.312

- Various other formal models were invented and shown to be exactly equivalent to TMs
  - Recursive functions, post systems (see Parkes' book)
- They never exceed Turing machines in power

Alan Turing

- Turing machines were invented by the British mathematician and early computer scientist Alan Turing in the 1930s
- He was investigating the properties of *algorithmic computation of effective procedures*
- He was not attempting to model a computer

- https://turingarchive.kings.cam.ac.uk/
- http://www.turing.org.uk/
- http://en.wikipedia.org/wiki/Alan_Turing
- https://turing100.acm.org/

The Chomsky Hierarchy

Type	Grammar	Machine	Other Equivalent
0	Unrestricted	Turing Machine	Recursive Functions, Post Systems, etc.
1	Context Sensitive	Linear Bounded Turing Machine	
2	Context Free	Push Down Recogniser	
3	Regular	Finite State Recogniser	Regular Expressions

Out of scope for us

Non-deterministic only/deterministic

The Chomsky Hierarchy

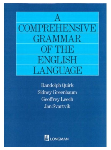
- Our diagram for the classification of grammars is called the Chomsky hierarchy
- Invented by the American linguist Noam Chomsky in the 1950s
- He used the terms type 0 to 3, but we use the more mnemonic terms (unrestricted to regular)
- http://www.chomsky.info/

The Chomsky Hierarchy

- The difference between the grammars is the restrictions we place on the form of the productions
- Each type of grammar is more powerful than the one below it
  - it can describe a wider range of languages (see the $a^n b^n$, $a^n b^n c^n$ languages)
- But each type of grammar is more difficult to use and less efficient to parse than the one below it

Summary

- Not all languages are context free, as can be shown by the "*uvwxy*" Theorem



A COMPREHENSIVE GRAMMAR OF THE ENGLISH LANGUAGE

Randolph Quirk
Sidney Greenbaum
Geoffrey Leech
Jan Svartvik

LONGMAN

- Context sensitive grammars are slightly more restricted than unrestricted grammars
- Introduction to Turing machines and Linear Bounded Turing machines
  - Their power as language processors
- The complete Chomsky Hierarchy

https://pixabay.com/photos/nature-landscape-clouds-trees-4627138/