Honors Class (Foundations of) Informatics

Technische Universiteit **Eindhoven** University of Technology

Tom Verhoeff

Department of Mathematics & Computer Science
Software Engineering & Technology

`www.win.tue.nl/~wstomv/edu/hci`

Automata and Grammars

?

Finite State Machine (FSM)

Takes a sequence of symbols from some alphabet as input.

Produces a sequence of symbols from some alphabet as output.

Has a finite set of (computational) states, with one *start state*.

Has a set of state transitions (computational steps):
Current state & input symbol determine output symbol & next state.

Can be given as

- a *table*, with a row per state and a column per input symbol, where each entry gives output and next state;
- a *labeled directed graph*, with a node per state, and an arrow between nodes labeled by input and output per transition.

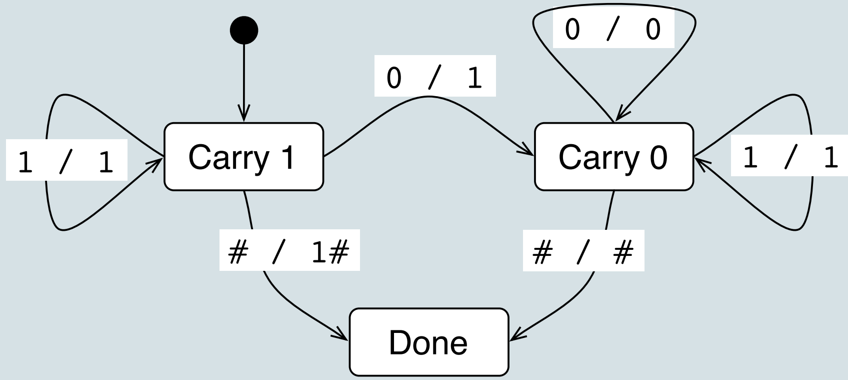
Also known as Deterministic Finite Automaton (DFA).

FSM to increment a binary number by one

Input alphabet = output alphabet = { 0, 1, # }

The number is input with its *least significant bit* first.

Output, Next State	Input		
State	0	1	#
A^*(Carry 1)	1, B	0, A	#, C
B (Carry 0)	0, B	1, B	1#, C
C (Done)	, C	, C	, C



$19 = 10011_2$ $\qquad$ $20 = 10100_2$

How to increment a *decimal* number by one? Multiply by 3?
How to check divisibility by 7? How to add two numbers?

*Start state

FSM has limited computational power

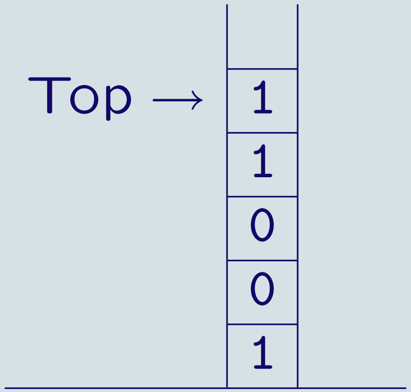
There exists no FSM

1. to increment an (arbitrary sized) number by one, when the number is offered with its *most significant digit* first;
2. to reverse the order of a sequence of symbols;
3. to multiply two (arbitrary sized) numbers;
4. to recognize a sequence of properly nested parentheses.

Reason: a FSM has finite storage.

Pushdown Automaton (PDA)

Like a FSM, but it also has an unbounded stack.



The stack stores a sequence of symbols from the stack's alphabet.

Besides the input symbol and current state, also the symbol at the top of the stack determines which transition the PDA takes.

Besides producing output and going to the next state, the action with a transition can also push a symbol onto the top of the stack, or pop the top symbol off the stack.

PDA has limited computational power

A PDA can do everything that a FSM can do, and even more:

- reverse a sequence of symbols
- recognize a sequence of properly nested parentheses

There exists no PDA

- to multiply two numbers
- etc.

Reason: a PDA has infinite storage with very limited access

Turing Machine (TM)

Like a FSM, but it also has a two-way infinite read-write tape.

The tape is divided into cells that each hold one symbol.

...	#	1	0	0	1	1	...
						↑	

It has a read/write head that is positioned at a specific tape cell.

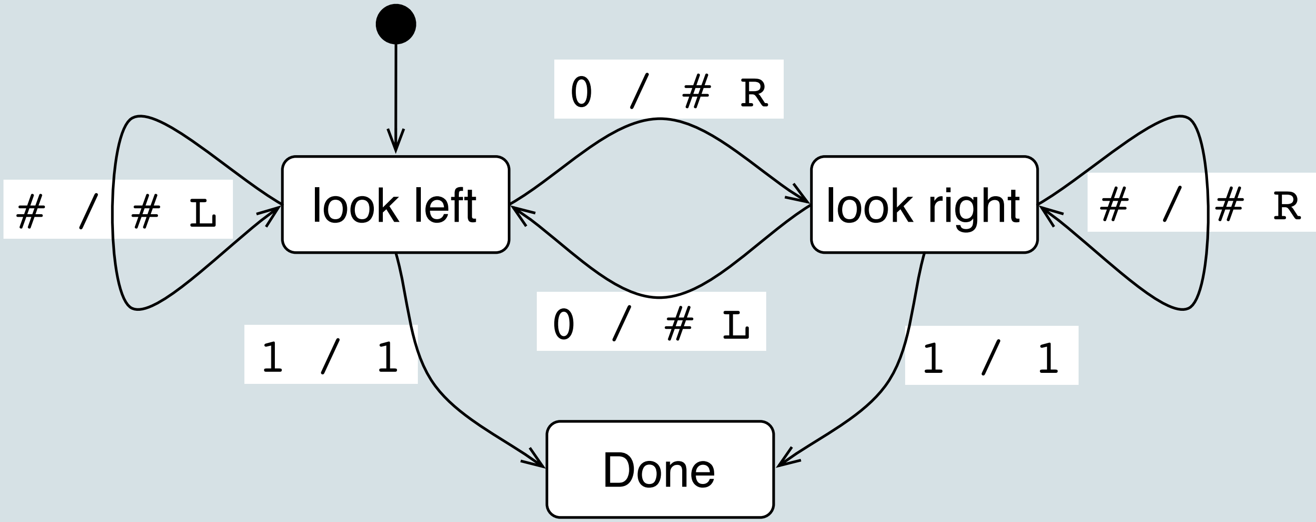
The current state and tape symbol under the R/W head determines the next state.

As action, it also can write a symbol at the head position and move the head one cell left/right.

Turing Machine that finds a 1 among 0s

Transition X / Y Z means

> 'when reading X, overwrite by Y and move the R/W head in direction Z' (R = right; L = left).



Formal Grammar to Generate a Formal Language

- Set of terminal symbols
- Set of non-terminal symbols

  One non-terminal symbol is designated as start symbol.
- Set of production rules, each of the form

$$\text{sequence of symbols} \rightarrow \text{sequence of symbols}$$

Grammar *generates* a set of sequences of symbols (formal language):

1. Start with the sequence consisting of just the start symbol.
2. Repeatedly replace subsequence t by u for production rule $t \rightarrow u$.
3. Terminate when the sequence contains terminal symbols only.

Formal Grammar: Example 1

Terminals: $\{ a, b \}$. Non-terminals: $\{ S, T \}$, start symbol S

Production rules:

1. $S \rightarrow aT$

2. $T \rightarrow b$

3. $T \rightarrow bS$

Production example:

$$\underline{S} \xrightarrow{1} \overline{a\underline{T}} \xrightarrow{3} a\overline{b\underline{S}} \xrightarrow{1} ab\overline{a\underline{T}} \xrightarrow{3} aba\overline{b\underline{S}} \xrightarrow{1} abab\overline{a\underline{T}} \xrightarrow{2} ababa\overline{b}$$

Generated language $= \{ (ab)^n \mid n \geq 1 \}$

Formal Grammar: Example 2

Terminals: $\{ a, b \}$. Non-terminals: $\{ S \}$, start symbol S

Production rules:

1. $S \rightarrow ab$

2. $S \rightarrow aSb$

Production example:

$$\underline{S} \xrightarrow{2} \overline{a\underline{S}b} \xrightarrow{2} a\overline{a\underline{S}b}b \xrightarrow{1} aa\overline{ab}bb$$

Generated language $= \{ a^n b^n \mid n \geq 1 \}$

Formal Grammar: Example 3

Terminals: $\{a, b, c\}$. Non-terminals: $\{S, T\}$, start symbol S

Production rules:

1. $S \longrightarrow aTSc$
2. $S \longrightarrow abc$
3. $Ta \longrightarrow aT$
4. $Tb \longrightarrow bb$

Production example:

$$\underline{S} \xrightarrow{1} \overline{aT\underline{S}c} \xrightarrow{2} a\underline{T\overline{a}}\overline{bc}c \xrightarrow{3} a\overline{a\underline{T}}\underline{b}cc \xrightarrow{4} aa\overline{bb}cc$$

Generated language $= \{ a^n b^n c^n \mid n \geq 1 \}$

Chomsky Hierarchy of Grammars

- Unrestricted Grammar (Type 0): no restrictions
- Context-Free Grammar (Type 2):

  left-hand side of each production rule consist of
  a *single non-terminal symbol*
- Regular Grammar (Type 3): like context-free grammar, but also

  right-hand side of each production rule is restricted:
  - either empty, or
  - single terminal symbol, or
  - single terminal symbol followed by single non-terminal symbol

Classification of Example Grammars

- Grammar 1 is regular
- Grammar 2 is context-free, and not regular
- Grammar 3 is unrestricted, and not context-free

Classification of Languages

A language is called ...	if it can be generated by a ...
Regular	Regular Grammar
Context-Free	Context-Free Grammar
Recursively Enumerable	Unrestricted Grammar

- Note: Different grammars can define the same language.
- Regular $\subset$ Context-Free $\subset$ Recursively Enumerable
- Context-free, non-regular grammar can generate regular language
- Unrestiected non-context-free grammar can generate context-free language

Classification of Example Languages

- $\{ (ab)^n \mid n \geq 1 \}$ is regular
- $\{ a^n b^n \mid n \geq 1 \}$ is context-free, but not regular
- $\{ a^n b^n c^n \mid n \geq 1 \}$ is recursively enumerable, but not context-free

Automata as Language Recognizers

By marking some states as accepting states an automaton can be turned into a recognizer:

> An automaton is said to accept a sequence when it terminates in an accepting state after processing that sequence as input.

An automaton is said to recognizes a language, if it accepts exactly the sequences of that language (no more, and no less).

- Finite State Machines recognize Regular Languages.
- Pushdown Automata recognize Context-Free Languages.
- Turing Machines recognize Recursively Enumerable Languages.

Summary

- There are many different *computational mechanisms*, such as
  - Finite State Machine (FSM)
  - Pushdown Automaton, with a single stack (PDA)
  - Turing Machine, with a two-way infinite read/write tape (TM)

  These also differ in *computational power*.

- A *formal language* is a set of sequences of symbols.
  A *formal grammar* defines (generates) a formal language:
  - Regular Grammar (recognizable by FSM)
  - Context-Free Grammar (recognizable by PDA)
  - Unrestricted Grammar (recognizable by TM)

References

http://en.wikipedia.org/wiki/Finite-state_machine

http://en.wikipedia.org/wiki/Pushdown_automaton

http://en.wikipedia.org/wiki/Turing_machine

http://en.wikipedia.org/wiki/Computability

http://en.wikipedia.org/wiki/Formal_language

http://en.wikipedia.org/wiki/Formal_grammar