Automata Theory (2A)

Young Won Lim
5/31/18

Please send corrections (or suggestions) to youngwlim@hotmail.com.

This document was produced by using LibreOffice and Octave.

Automata

The word **automata** (the plural of **automaton**) comes from the Greek word **αὐτόματα**, which means "**self-acting**".

https://en.wikipedia.org/wiki/Automata_theory

Automata Informal description (1) – Inputs

An automaton runs when it is given some sequence of **inputs** in discrete (individual) time steps or steps.

An automaton processes one input picked from a set of **symbols** or **letters**, which is called an **alphabet**.

The symbols received by the automaton as **input** at any step are a finite sequence of symbols called **words**.

https://en.wikipedia.org/wiki/Automata_theory

Automata informal description (2) – States

An automaton has a finite set of **states**.

At each moment during a run of the automaton, the automaton is in one of its states.

When the automaton receives new **input** it moves to another **state** (or **transitions**) based on a **function** that takes the **current state** and **input symbol** as parameters.

This function is called the **transition function**.

https://en.wikipedia.org/wiki/Automata_theory

Automata informal description (3) – Stop

The automaton reads the symbols of the **input word** one after another and transitions from **state** to state according to the **transition function** until the word is read completely.

Once the input word has been read, the automaton is said to have stopped.

The state at which the automaton **stops** is called the **final state**.

https://en.wikipedia.org/wiki/Automata_theory

Automata informal description (4) – Accept / Reject

Depending on the **final state**, it's said that the automaton either **accepts** or **rejects** an **input word**.

There is a subset of states of the automaton, which is defined as the set of **accepting states**.

If the **final state** is an **accepting state**, then the automaton **accepts** the **word**.

Otherwise, the **word** is **rejected**.

https://en.wikipedia.org/wiki/Automata_theory

Automata informal description (5) – Language

The set of **all the words accepted** by an automaton is called the "**language** of that automaton".

Any **subset** of the **language** of an automaton is a language **recognized** by that automaton.

https://en.wikipedia.org/wiki/Automata_theory

Automata informal description (6) – Decision on inputs

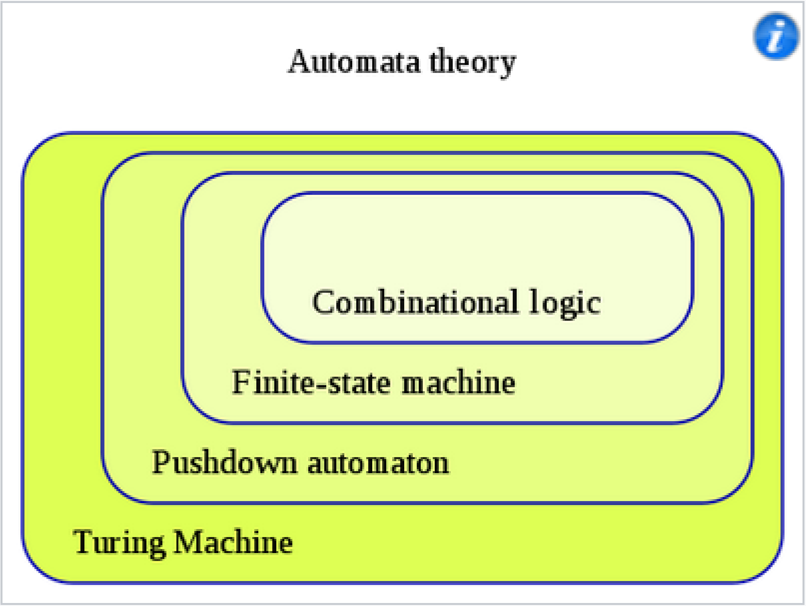
an **automaton** is a mathematical object
that takes a word as **input**
and **decides** whether to **accept** it or **reject** it.

Since all computational problems are
reducible into the **accept/reject question** on **inputs**,
(all problem instances can be represented
in a finite length of symbols),
automata theory plays a crucial role
in computational theory.

https://en.wikipedia.org/wiki/Automata_theory

Class of Automata

- Combinational Logic
- Finite State Machine (FSM)
- Pushdown Automaton (PDA)
- Turing Machine



Classes of automata
(Clicking on each layer will take you to an article on that subject)

https://en.wikipedia.org/wiki/Automata_theory

Class of Automata

Finite State Machine (FSM)	Regular Language
Pushdown Automaton (PDA)	Context-Free Language
Turing Machine	Recursively Enumerable Language
Automaton	Formal Languages

https://en.wikipedia.org/wiki/Automata_theory

Definition of Finite State Automata

A deterministic finite automaton is represented formally
by a 5-tuple $<Q, \Sigma, \delta, q_0, F>$, where:

Q is a finite set of **states**.
Σ is a finite set of **symbols**, called the **alphabet** of the automaton.
δ is the **transition function**, that is, $\delta: Q \times \Sigma \rightarrow Q$.
q_0 is the **start state**, that is, the state of the automaton
before any input has been processed, where $q_0 \in Q$.
F is a set of **states** of Q (i.e. $F \subseteq Q$) called **accept states**.

https://en.wikipedia.org/wiki/Automata_theory

Pushdown Automaton

a type of automaton that employs a **stack**.

The term "pushdown" refers to the fact that
the stack can be regarded as being "pushed down"
like a tray dispenser at a cafeteria,
since the operations never work on elements
other than the **top element**.

A **stack automaton**, by contrast, does allow
access to and operations on deeper elements.

https://en.wikipedia.org/wiki/Automata_theory

Deterministic Finite State Machine

A **deterministic finite state machine** or **acceptor** deterministic finite state machine is a quintuple $(\Sigma, S, s_0, \delta, F)$, where:

- Σ is the **input alphabet** (a finite, non-empty set of symbols).
- S is a finite, non-empty set of **states**.
- s_0 is an **initial state**, an element of S.
- δ is the **state-transition function**: $\delta : S \times \Sigma \rightarrow S$
- F is the set of **final states**, a (possibly empty) subset of S.

https://en.wikipedia.org/wiki/Automata_theory

Deterministic Pushdown Automaton

A **PDA** is formally defined as a 7-tuple:

$M = (Q, \Sigma, \Gamma, \delta, q_0, Z, F)$ where

Q is a finite set of **states**
Σ is a finite set which is called the **input alphabet**
Γ is a finite set which is called the **stack alphabet**
δ is a finite subset of $Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma \times Q \times \Gamma^*$, the **transition relation**.
$q_0 \in Q$ is the **start state**
$Z \in \Gamma$ is the **initial stack symbol**
$F \subseteq Q$ is the set of **accepting states**

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

Turing Machine

Turing machine as a 7-tuple $M = \langle Q, \Gamma, b, \Sigma, \delta, q_0, F \rangle$ where $\setminus$ set minus

Q is a finite, non-empty set of **states**;
Γ is a finite, non-empty set of **tape alphabet symbols**;
$b \in \Gamma$ is the **blank symbol**
$\Sigma \subseteq \Gamma \setminus \{ b \}$ is the set of **input symbols** in the initial tape contents;
$q_0 \in Q$ is the **initial state**;
$F \subseteq Q$ is the set of **final states** or **accepting states**.
$\delta : (Q \setminus F) \times \Gamma \rightarrow Q \times \Gamma \times \{ L , R \}$ is **transition function**,
where **L** is **left shift**, **R** is **right shift**.

The initial tape contents is said to be accepted by M if it eventually halts in a state from F .

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

Deterministic PDA (1) – transition relation

An element (p, a, A, q, α) ∈ δ is a **transition** of M.
It has the intended meaning that M, in **state** p ∈ Q,
on the **input** a ∈ Σ ∪ { ε } and
with A ∈ Γ as **topmost stack symbol**,
may read a, change the **state** to q, pop A,
replacing it by pushing α ∈ Γ^*.

The (Σ ∪ { ε }) component of the transition relation
is used to formalize that the PDA can
either read a letter from the input,
or proceed leaving the input untouched.

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

Deterministic PDA (2) – transition function

δ is the **transition function**,

mapping $Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma$

into finite subsets of $Q \times \Gamma^*$

$\delta(p, a, A) \rightarrow (q,\alpha)$

Here δ (p, a, A) contains all possible actions in **state** p with A on the **stack**, while reading a on the **input**.

One writes for example $\delta(p, a, A) = \{ (q, BA) \}$

precisely when $(q, BA) \in \{ (q, BA) \}$, $(q, BA) \in \delta(p, a, A)$

Because $((p, a, A), \{(q, BA)\}) \in \delta$.

Note that finite in this definition is essential.

$\delta(p, a, A) \rightarrow (q,\alpha)$

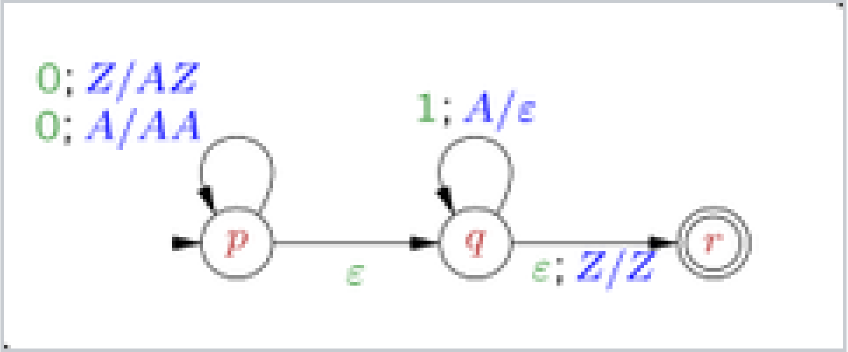
https://en.wikipedia.org/wiki/Pushdown_automaton

Deterministic PDA Example (1) – description

The following is the formal description of the PDA which recognizes the language $\{ 0^n 1^n \mid n \geq 0 \}$ by final state:

$M = (Q, \Sigma, \Gamma, \delta, q_0, Z, F)$, where

states:	$Q = \{p, q, r\}$
input alphabet:	$\Sigma = \{0, 1\}$
stack alphabet:	$\Gamma = \{A, Z\}$
start state:	$q_0 = p$
start stack symbol:	Z
accepting states:	$F = \{r\}$



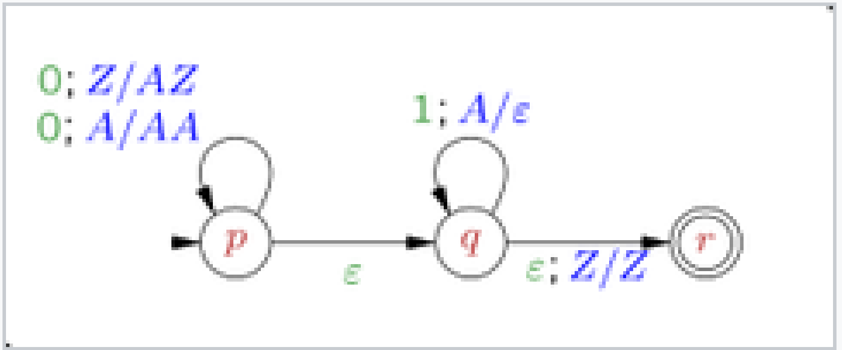
PDA for $\{0^n 1^n \mid n \geq 0\}$ (by final state)

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

Deterministic PDA Example (2) – instructions

The **transition relation** δ consists of the following six instructions:

(p, 0, Z, p, AZ)	0; Z/AZ, p→p
(p, 0, A, p, AA)	0; A/AA, p→p
(p, ϵ, Z, q, Z)	ϵ, Z/Z, p→q
(p, ϵ, A, q, A)	ϵ, A/A, p→q
(q, 1, A, q, ϵ)	1, A/ϵ, q→q
(q, ϵ, Z, r, Z)	ϵ, Z/Z, p→r



PDA for $\{0^n 1^n \mid n \geq 0\}$
(by final state)

the instruction (p, a, A, q, α) by an edge from state p to state q labelled by a ; A / α (read a; replace A by α).

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

Deterministic PDA Example (3) – instruction description

(p, 0, Z, p, AZ) , (p, 0, A, p, AA),	in state p any time the symbol 0 is read, one A is pushed onto the stack. Pushing symbol A on top of another A is formalized as replacing top A by AA (and similarly for pushing symbol A on top of a Z)
(p, ϵ, Z, q, Z), (p, ϵ, A, q, A),	at any moment the automaton may move from state p to state q.
(q, 1, A, q, ϵ),	in state q, for each symbol 1 read, one A is popped.
(q, ϵ, Z, r, Z).	the machine may move from state q to accepting state r only when the stack consists of a single Z.

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

Deterministic PDA Computation (1) – ID

to formalize the **semantics** of the pushdown automaton

a description of the current situation is introduced.

Any 3-tuple $(p , w , \beta) \in Q \times \Sigma^* \times \Gamma^*$ is called

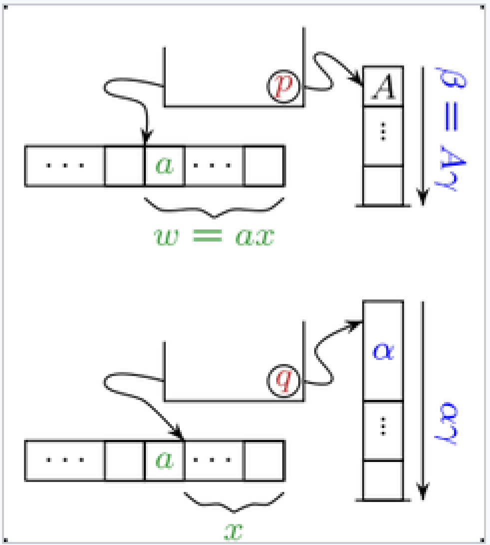
an **instantaneous description** (ID) of

$M = (Q, \Sigma, \Gamma, \delta, q_0, Z, F)$ which includes

the current **state**,

the part of the **input** tape that has not been read, and

the contents of the **stack** (topmost symbol written first).



a step of the pushdown automaton

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

Deterministic PDA Computation (2) – step-relation

The **transition relation** δ defines

the **step-relation** $\vdash_M$ on **instantaneous descriptions**.

For instruction (p, a, A, q, α) ∈ δ

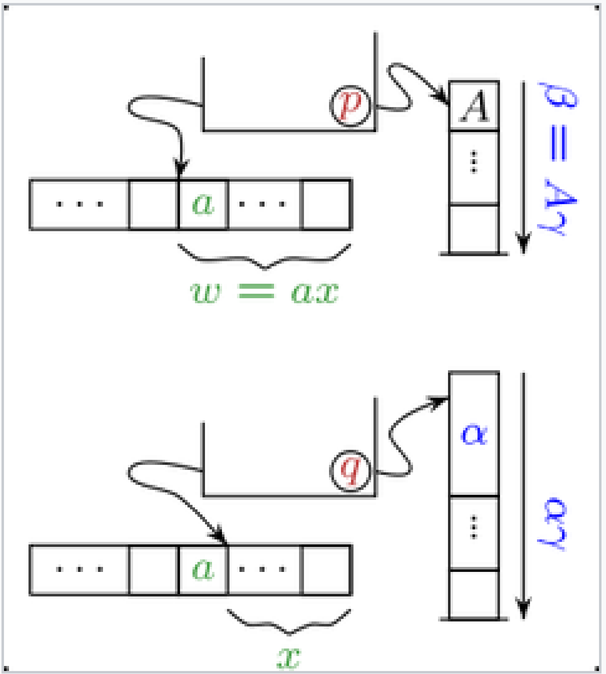
there exists a step (p , ax, Aγ) ⊢ M (q, x , αγ),

for every $x \in \Sigma^*$ and every $\gamma \in \Gamma^*$.

p, q : states

ax, x : inputs

Aγ, αγ : stack elementes



a step of the pushdown automaton

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

Deterministic PDA Computation (3) – non-deterministic

Nondeterministic :

in a given **instantaneous description** (p, w, β)

there may be several possible **steps**.

Any of these steps can be chosen in a computation.

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

Deterministic PDA Computation (4) – pop operation

With the above definition in each step always a single **symbol** (**top** of the **stack**) is popped, replacing it with as many symbols as necessary.

As a result no step is defined when the stack is empty.

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

Deterministic PDA Computation (5) – initial description

Computations of the pushdown automaton are sequences of **steps**.

The computation starts in the **initial state** q_0 with the **initial stack symbol** Z on the stack, and a string w on the **input tape**, thus with **initial description** (q_0, w, Z).

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

Deterministic PDA Computation (6) – acceptance modes

There are two modes of **accepting**.

either accepts by **final state**,

which means after reading its input the automaton reaches an **accepting state** (in F)

uses the **internal memory** (**state**)

or it accepts by **empty stack** (ε),

which means after reading its input the automaton empties its stack.

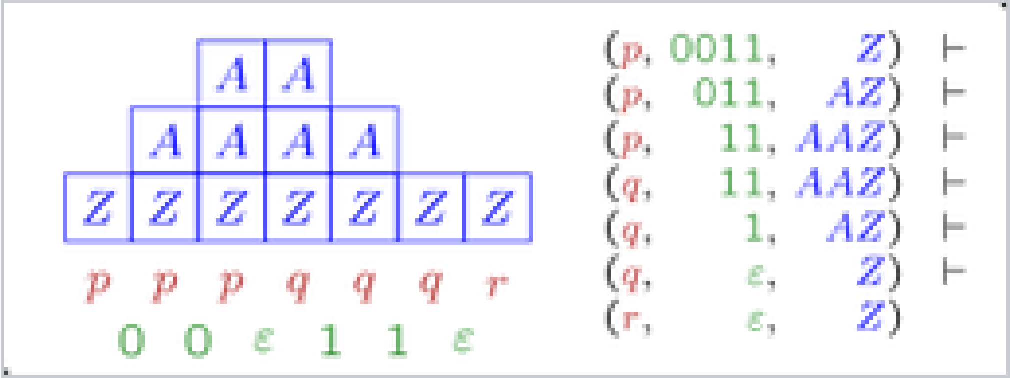
uses the **external memory** (**stack**).

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

Computation Example (1)

The following illustrates
how the above PDA computes
on different input strings.

The subscript M from the step symbol
⊢ is here omitted.



accepting computation for 0011

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

Computation Example (2)

input string = 0011.

There are various computations, depending on the moment the move from state p to state q is made.

Only one of these is accepting.

(p , 0011 , Z) ⊢ (p, ϵ, Z, q, Z),

(q , 0011 , Z) ⊢ (q, ϵ, Z, r, Z).

(r , 0011 , Z)

1. (p, 0, Z, p, AZ)
2. (p, 0, A, p, AA)
3. (p, ϵ, Z, q, Z)
4. (p, ϵ, A, q, A)
5. (q, 1, A, q, ϵ)
6. (q, ϵ, Z, r, Z)

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

Computation Example (3)

The final state is accepting, but the input is not accepted this way as it has not been read.

(p , 0011 , Z) ⊢
(p , 011 , A Z) ⊢
(q , 011 , A Z)

(p, 0, Z, p, AZ)
(q, 1, A, q, ϵ)

No further steps possible.

1. (p, 0, Z, p, AZ)
2. (p, 0, A, p, AA)
3. (p, ϵ, Z, q, Z)
4. (p, ϵ, A, q, A)
5. (q, 1, A, q, ϵ)
6. (q, ϵ, Z, r, Z)

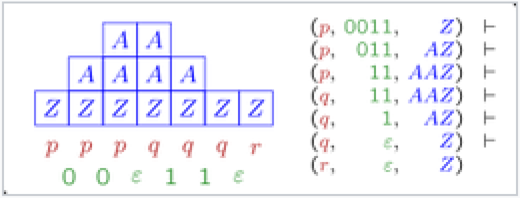
https://en.wikipedia.org/wiki/Pushdown_automaton

Computation Example (4)

(p , 0011 , Z) $\vdash$	(p, 0, A, p, AA)
(p , 011 , AZ) $\vdash$	(p, 0, A, p, AA)
(p , 11 , AAZ) $\vdash$	(p, ϵ, A, q, A)
(q , 11 , AAZ) $\vdash$	(q, 1, A, q, ϵ)
(q , 1 , AZ) $\vdash$	(q, 1, A, q, ϵ)
(q , ϵ , Z) $\vdash$	(q, ϵ, Z, r, Z)
(r , ϵ , Z)	

1. (p, 0, Z, p, AZ)
2. (p, 0, A, p, AA)
3. (p, ϵ, Z, q, Z)
4. (p, ϵ, A, q, A)
5. (q, 1, A, q, ϵ)
6. (q, ϵ, Z, r, Z)

Accepting computation: ends in accepting state, while complete input has been read.



accepting computation for 0011

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

Computation Example (5)

Input string = 00111. Again there are various computations. None of these is accepting.

(p , 00111 , Z) ⊢   (p, ϵ, Z, q, Z)
(q , 00111 , Z) ⊢   (q, ϵ, Z, r, Z)
(r , 00111 , Z)

The final state is accepting,
but the input is not accepted
this way as it has not been read.

1. (p, 0, Z, p, AZ)
2. (p, 0, A, p, AA)
3. (p, ϵ, Z, q, Z)
4. (p, ϵ, A, q, A)
5. (q, 1, A, q, ϵ)
6. (q, ϵ, Z, r, Z)

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

Computation Example (6)

(p , 00111 , Z) ⊢
(p , 0111 , A Z) ⊢
(q , 0111 , A Z)

(p, 0, Z, p, AZ)
(p, ϵ, A, q, A)

No further steps possible.

1. (p, 0, Z, p, AZ)
2. (p, 0, A, p, AA)
3. (p, ϵ, Z, q, Z)
4. (p, ϵ, A, q, A)
5. (q, 1, A, q, ϵ)
6. (q, ϵ, Z, r, Z)

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

Computation Example (7)

(p , 00111 , Z) ⊢
(p , 0111 , A Z) ⊢
(p , 111 , A A Z) ⊢
(q , 111 , A A Z) ⊢
(q , 11 , A Z) ⊢
(q , 1 , Z) ⊢
(r , 1 , Z)

(p, 0, Z, p, AZ)
(p, 0, Z, p, AZ)
(p, ϵ, A, q, A)
(q, 1, A, q, ϵ)
(q, 1, A, q, ϵ)
(q, ϵ, Z, r, Z)

1. (p, 0, Z, p, AZ)
2. (p, 0, A, p, AA)
3. (p, ϵ, Z, q, Z)
4. (p, ϵ, A, q, A)
5. (q, 1, A, q, ϵ)
6. (q, ϵ, Z, r, Z)

The final state is accepting, but the input is not accepted this way as it has not been (completely) read.

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

PDA and Context Free Language (1)

Every **context-free grammar** can be transformed
into an equivalent **nondeterministic pushdown automaton**.

The derivation process of the grammar
is simulated in a **leftmost way**

Where the grammar rewrites a **nonterminal**,
the **PDA** takes the **topmost nonterminal** from its **stack**
and replaces it by the **right-hand part**
of a grammatical rule (expand).

Where the grammar generates a **terminal** symbol,
the **PDA** reads a symbol from **input**
when it is the **topmost symbol** on the **stack** (match).

In a sense the **stack** of the **PDA** contains
the unprocessed data of the grammar,
corresponding to a pre-order traversal of a derivation tree.

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

PDA and Context Free Language (1)

Every **context-free grammar** can be transformed
into an equivalent **nondeterministic pushdown automaton**.

The derivation process of the grammar
is simulated in a **leftmost way**

In a sense the **stack** of the **PDA** contains
the unprocessed data of the grammar,
corresponding to a pre-order traversal of a derivation tree.

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

PDA and Context Free Language (2)

The derivation process of the grammar

is simulated in a **leftmost way**

Where the grammar rewrites a **nonterminal**,

the **PDA** takes the **topmost nonterminal** from its **stack**

and replaces it by the **right-hand part**

of a grammatical rule (**expand**).

Where the grammar generates a **terminal** symbol,

the **PDA** reads a symbol from input

when it is the **topmost symbol** on the **stack** (**match**).

I

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

Computation Example (3)

Technically, given a context-free grammar, the PDA has a single state, 1, and its transition relation is constructed as follows.

(1 , ε , A , 1 , α) for each rule A → α (expand)

(1 , a , a , 1 , ε) for each terminal symbol a (match)

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

PDA and Context Free Language (2)

Technically, given a context-free grammar,
the PDA has a single **state**, 1,
and its **transition relation** is constructed as follows.

(1 , ε , A , 1 , α) for each rule A → α (expand)

(1 , a , a , 1 , ε) for each terminal symbol a (match)

The PDA **accepts** by **empty stack**.

Its **initial stack symbol** is the grammar's **start symbol**.

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

Turing Machine

The Turing machine mathematically models a machine that mechanically operates on a tape.

On this tape are symbols, which the machine can read and write, one at a time, using a tape head.

Operation is fully determined by a finite set of elementary instructions such as

> "in state 42, if the symbol seen is 0, write a 1;
> if the symbol seen is 1, change into state 17;
> in state 17, if the symbol seen is 0,
> write a 1 and change to state 6;" etc.

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

Turing Machine – Tape

A **tape** divided into **cells**, one next to the other.
Each cell contains a **symbol** from some finite **alphabet**.
The alphabet contains a **special blank** symbol
(here written as '0') and one or more other symbols.

The tape is assumed to be arbitrarily extendable
to the left and to the right, i.e.,
the Turing machine is always supplied with
as much tape as it needs for its computation.

Cells that have not been written before are assumed
to be filled with the **blank symbol**.

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

Turing Machine – Head, State Register

A **head** that can read and write symbols on the tape and move the tape left and right one (and only one) cell at a time.
In some models the head moves and the tape is stationary.

A **state register** that stores the state of the Turing machine, one of finitely many.
Among these is the special **start state** with which the state register is initialized.
These states, writes Turing, replace the "state of mind" a person performing computations would ordinarily be in.

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

Turing Machine – Table of Instruction

A **finite table** of **instructions** that,
given the **state**(qi) the machine is currently in
and the **symbol**(aj) it is reading on the tape
(symbol currently under the head),
tells the machine to do the following in sequence
(for the 5-tuple models):

1. Either erase or write a symbol (replacing aj with aj1).

2. Move the head (which is described by dk and can have values: 'L' for one step left or 'R' for one step right or 'N' for staying in the same place).

3. Assume the same or a new state as prescribed (go to state qi1).

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

Turing Machine – unlimited amount

Note that every part of the machine
(i.e. its state, symbol-collections,
and used tape at any given time) and
its actions (such as printing, erasing and tape motion) is
finite, discrete and distinguishable;

it is the unlimited amount of tape and runtime
that gives it an unbounded amount of storage space.

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

Turing Machine – 4 tuple models

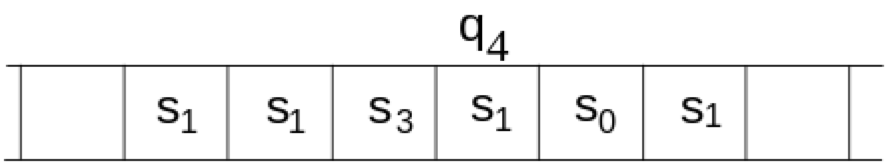
In the 4-tuple models,
erasing or writing a symbol (aj1) and
moving the head left or right (dk) are
specified as separate instructions.

Specifically, the table tells the machine to (ia) erase or write a symbol or (ib) move the head left or right, and then (ii) assume the same or a new state as prescribed, but not both actions (ia) and (ib) in the same instruction. In some models, if there is no entry in the table for the current combination of symbol and state then the machine will halt; other models require all entries to be filled.

Note that every part of the machine (i.e. its state, symbol-collections, and used tape at any given time) and its actions (such as printing, erasing and tape motion) is finite, discrete and distinguishable; it is the unlimited amount of tape and runtime that gives it an unbounded amount of storage space.

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

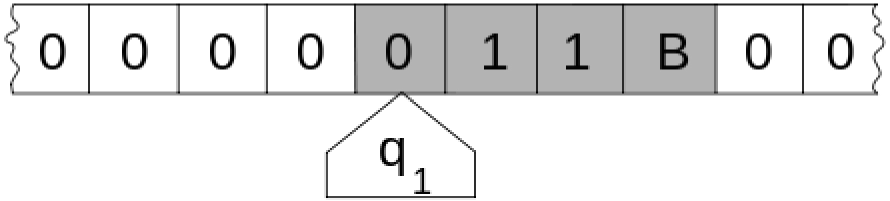
Turing Machine – head, instruction



The **head** is always over a particular square of the tape;
only a finite stretch of squares is shown.
The **instruction** to be performed (q4) is shown
over the scanned square.

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

Turing Machine – internal state, blank



Here, the **internal state** (q1) is shown inside the head, and the illustration describes the **tape** as being infinite and **pre-filled** with "**0**", the symbol serving as **blank**.

The system's full state (its complete configuration) consists of the **internal state**, any **non-blank symbols** on the tape (in this illustration "11B"), and the **position** of the head relative to those symbols including blanks, i.e. "011B".

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

Turing Machine

Turing machine as a 7-tuple $M = \langle Q, \Gamma, b, \Sigma, \delta, q_0, F \rangle$ where $\setminus$ set minus

Q is a finite, non-empty set of **states**;

Γ is a finite, non-empty set of tape **alphabet symbols**;

$b \in \Gamma$ is the **blank symbol** (the only symbol allowed to occur on the tape infinitely often at any step during the computation);

$\Sigma \subseteq \Gamma \setminus \{ b \}$ is the set of **input symbols**, that is, the set of symbols allowed to appear in the initial tape contents;

$q_0 \in Q$ is the **initial state**;

$F \subseteq Q$ is the set of **final states** or **accepting states**. The initial tape contents is said to be accepted by M if it eventually halts in a state from F .

$\delta : (Q \setminus F) \times \Gamma \to Q \times \Gamma \times \{ L , R \}$ is a partial function called the **transition function**, where **L** is **left shift**, **R** is **right shift**. (A relatively uncommon variant allows "**no shift**", say **N**, as a third element of the latter set.) If δ is not defined on the current state and the current tape symbol, then the machine halts;

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

3-State Busy Beaver

The 7-tuple for the 3-state busy beaver looks like this (see more about this busy beaver at Turing machine examples):

$Q = \{ A , B , C , HALT \}$	(states);
$\Gamma = \{ 0 , 1 \}$	(tape alphabet symbols);
$b = 0$	(blank symbol);
$\Sigma = \{ 1 \}$	(input symbols);
$q\ 0 = A$	(initial state);
$F = \{ HALT \}$	(final states);
$\delta =$ see state-table below	(transition function).

Initially all tape cells are marked with 0

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

3-State Busy Beaver

The 7-tuple for the 3-state busy beaver looks like this (see more about this busy beaver at Turing machine examples):

$Q = \{ A , B , C , HALT \}$	(states);
$\Gamma = \{ 0 , 1 \}$	(tape alphabet symbols);
$b = 0$	(blank symbol);
$\Sigma = \{ 1 \}$	(input symbols);
$q\ 0 = A$	(initial state);
$F = \{ HALT \}$	(final states);
$\delta =$ see state-table below	(transition function).

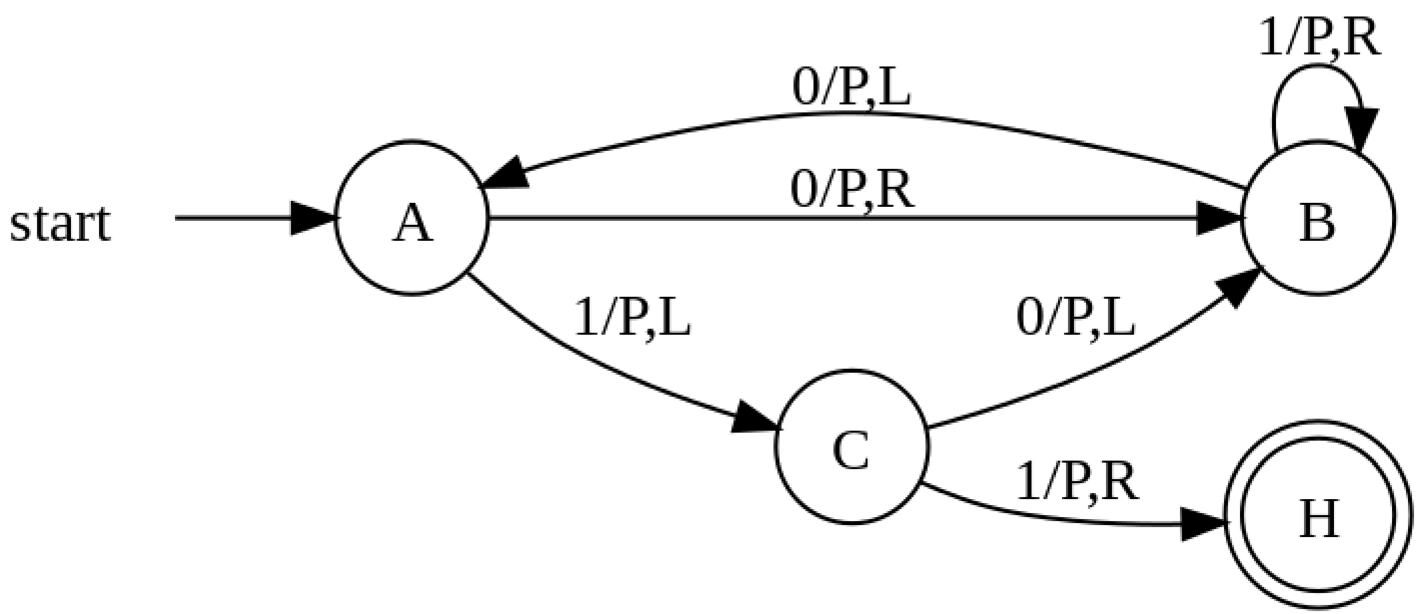
Initially all tape cells are marked with 0

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

3-State Busy Beaver

State table for 3 state, 2 symbol busy beaver

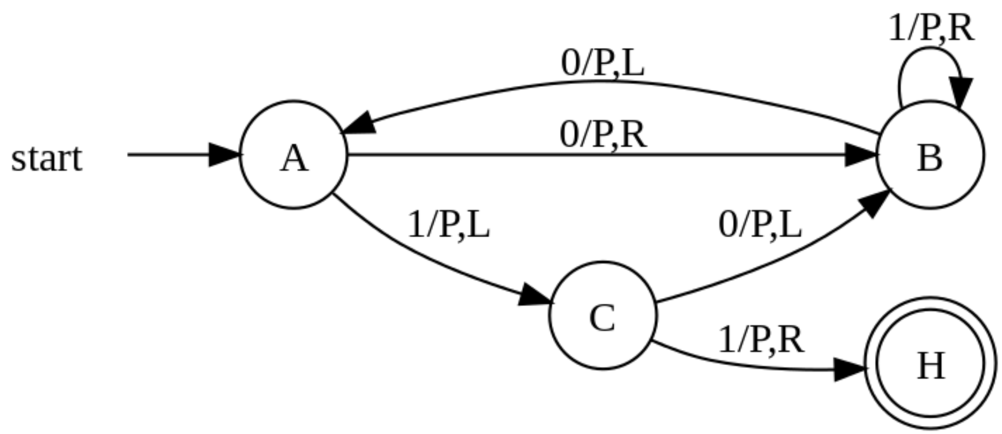
Tape symbol	Current state A			Current state B			Current state C		
	Write symbol	Move tape	Next state	Write symbol	Move tape	Next state	Write symbol	Move tape	Next state
0	1	R	**B**	1	L	**A**	1	L	**B**
1	1	L	**C**	1	R	**B**	1	R	**HALT**



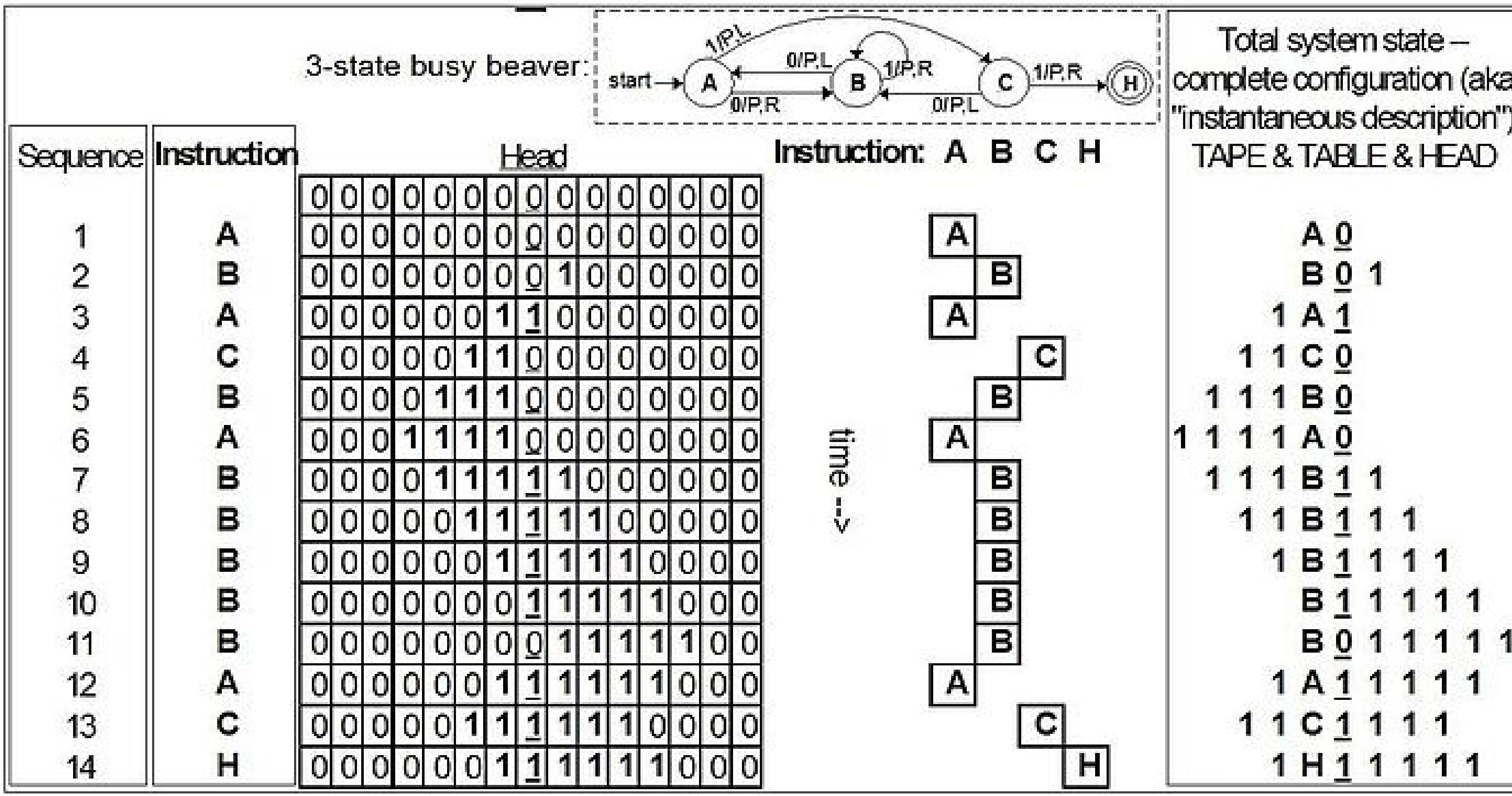
https://en.wikipedia.org/wiki/Turing_machine

3-State Busy Beaver

Each circle represents a "**state**" of the table
—an "**m-configuration**" or "**instruction**".
"**Direction**" of a state transition is shown by an **arrow**.
The **label** (e.g. 0/P,R) near the outgoing state
(at the "tail" of the arrow) specifies the **scanned symbol**
that causes a particular transition (e.g. 0) followed by a slash /,
followed by the subsequent "**behaviors**" of the machine,
e.g. "P Print" then move tape "R Right".



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



Sequence	Instruction								Head								Total system state
		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
1	A	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	A 0
2	B	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	B 0 1
3	A	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	1 A 1
4	C	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	1 1 C 0
5	B	0	0	0	0	1	1	1	0	0	0	0	0	0	0	0	1 1 1 B 0
6	A	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0	1 1 1 1 A 0
7	B	0	0	0	0	1	1	1	1	1	0	0	0	0	0	0	1 1 1 B 1 1
8	B	0	0	0	0	0	1	1	1	1	1	0	0	0	0	0	1 1 B 1 1 1
9	B	0	0	0	0	0	0	1	1	1	1	1	0	0	0	0	1 B 1 1 1 1
10	B	0	0	0	0	0	0	0	1	1	1	1	1	0	0	0	B 1 1 1 1 1
11	B	0	0	0	0	0	0	0	0	1	1	1	1	1	0	0	B 0 1 1 1 1 1
12	A	0	0	0	0	0	0	1	1	1	1	1	1	0	0	0	1 A 1 1 1 1 1
13	C	0	0	0	0	0	1	1	1	1	1	1	0	0	0	0	1 1 C 1 1 1 1
14	H	0	0	0	0	0	0	1	1	1	1	1	1	0	0	0	1 H 1 1 1 1 1

Progress of the computation (state-trajectory) of a 3-state busy beaver

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

n-State Busy Beaver

the design specifications:

1. The machine has **n** "**operational**" **states** plus a **Halt state**, where **n** is a positive integer, and one of the **n** states is distinguished as the **starting state**.
2. The machine uses a single two-way infinite (or unbounded) **tape**.
3. The **tape alphabet** is {0, 1}, with **0** serving as the **blank symbol**.
4. The machine's **transition function** takes two inputs:
   - the **current** non-Halt state,
   - the **symbol** in the current tape cell,

   and produces three outputs:
   - a **symbol** to write over the symbol in the current tape cell (it may be the same symbol as the symbol overwritten),
   - a **direction** to move (**left** or **right)**
   - a **state** to **transition** into (which may be the Halt state).

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

n-State Busy Beaver

"**Running**" the machine consists of starting in the **starting state**, with the current tape cell being any cell of a **blank** (all-0) tape, and then iterating the **transition function** until the **Halt** state is entered (if ever).

If, and only if, the machine eventually halts, then the number of 1s finally remaining on the tape is called the machine's score.

The n-state busy beaver (BB-n) game is a contest to find such an n-state Turing machine having the largest possible score
— the largest number of 1s on its tape after halting.
A machine that attains the largest possible score among all n-state Turing machines is called an n-state busy beaver, and a machine whose score is merely the highest so far attained (perhaps not the largest possible) is called a champion n-state machine.

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

References

[1] http://en.wikipedia.org/
[2]