2.2 Pushdown Automata

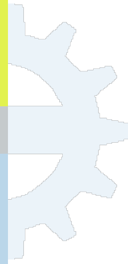
- Pushdown automata are like NFAs, but have an extra component: (an infinite) *stack*
- We can write a new symbol on the stack at the top by **`pushing`** it
- We can read and remove the top symbol from the stack by **`popping`** it
- In a pushdown automaton the transitions always also concern the stack
- The stack gives the automaton a ”memory” by which we can avoid some of the limitations that finite automata have

Definition 2.13

A pushdown automaton is a 6-tuple $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$, where

- Q is the finite set of states,
- Σ is the input alphabet,
- Γ is the stack alphabet,
- $q_0 \in Q$ is the start state,
- $F \subseteq Q$ is the set of accept states, and
- δ is the set-valued transition function:

$$\delta: Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon \to \mathcal{P}(Q \times \Gamma_\varepsilon)$$

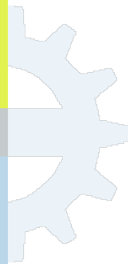


- In general pushdown automata are nondeterministic:

$$\delta(r, x, a) = \{ (r_1, b_1), ..., (r_k, b_k) \}$$

  - By reading the input symbol x and stack symbol a
  - The automaton may transfer from state r to one of the states $r_1, ..., r_k$, and
  - Simultaneously replace the top symbol of the stack by one of the symbols $b_1, ..., b_k$.

1. If $x = \varepsilon$, the automaton transfers without reading an input symbol;
2. If $a = \varepsilon$, the automaton does not read a stack symbol, but writes a new symbol at the top of the stack, leaving the old top symbol as is (**push**);
3. If $a \neq \varepsilon$ and $b_i = \varepsilon$, the top symbol of the stack is read and removed, but no new symbol is not written in its stead (**pop**)



A pushdown automaton $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$ accepts the string $w \in \Sigma^*$ if

- it can be written as $w = w_1 w_2 ... w_m$, where each $w_i \in \Sigma_\varepsilon$, and furthermore there exists
- a sequence of states $r_0, r_1, ..., r_m \in Q$ and
- strings $s_0, s_1, ..., s_m \in \Gamma^*$

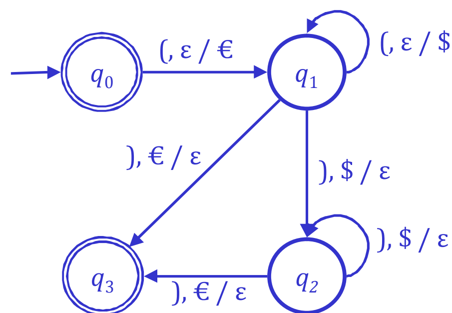
satisfying the following three conditions.

1. In the start M is in the start state and with an empty stack: $r_0 = q_0$ and $s_0 = \varepsilon$;
1. $(r_{i+1}, b) \in \delta(r_i, w_{i+1}, a)$ for all $i \in \{ 0, ..., m-1 \}$, where $s_i = at$ and $s_{i+1} = bt$ for some $a, b \in \Gamma_\varepsilon$ and $t \in \Gamma^*$;
3. $r_m \in F$.

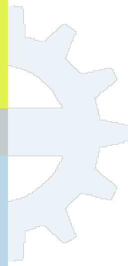
The language of balanced pairs of parentheses $\{ (^k)^k \mid k \geq 0 \}$ is a context-free language that is not a regular language. It can be recognized with a pushdown automaton $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$:

- $Q = \{q_0, q_1, q_2, q_3\}$,
- $\Sigma = \{ (,) \}$,
- $\Gamma = \{ \$, € \}$,
- q_0 is the start state,
- $F = \{ q_0, q_3 \}$ and
- δ is:

$\delta(q_0, (, \varepsilon) = \{ (q_1, €) \}$,
$\delta(q_1, (, \varepsilon) = \{ (q_1, \$) \}$,
$\delta(q_1,), \$) = \{ (q_2, \varepsilon) \}$,
$\delta(q_1,), €) = \{ (q_3, \varepsilon) \}$,
$\delta(q_2,), \$) = \{ (q_2, \varepsilon) \}$,
$\delta(q_2,), €) = \{ (q_3, \varepsilon) \}$,
$\delta(q, \sigma, \gamma) = \varnothing$ for other triples (q, σ, γ)



$\delta(q_0, (())), \varepsilon) \Rightarrow \delta(q_1, ())), €) \Rightarrow \delta(q_1, ())), \$€) \Rightarrow \delta(q_1,))), \$\$€) \Rightarrow \delta(q_2,)), \$€) \Rightarrow$
$\delta(q_2,), €) \Rightarrow \delta(q_3, \varepsilon, \varepsilon)$ $\quad q_3 \in F$, and hence $((())) \in L(M)$



Theorem 2.20 *A language is context free if and only if some pushdown automaton recognizes it*.

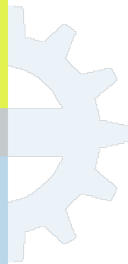
- A pushdown automaton M is deterministic if every configuration (r, x, a) has at most one possible successor (r', x', a'), for which

$$(r, x, a) \Rightarrow_M (r', x', a')$$

- Nondeterministic pushdown automata are strictly more powerful than deterministic ones. For example, the language

$$\{ ww^R \mid w \in \{ a, b \}^* \}$$

cannot be recognized using a deterministic pushdown automaton
- Deterministic context-free languages can be parsed more efficiently than general context-free languages



3. The Church-Turing Thesis

Church-Turing thesis *Any mechanically solvable problem can be solved using a Turing machine*

Many formulations of mechanical computation have an equal computing power with Turingin machines:

- RAM machines, simple random access computer models
- Programming languages (1950s)
- String rewriting systems (Post, 1936 and Markov, 1951)
- λ-calculus (Church, 1936)
- Recursively defined functions (Gödel and Kleene, 1936)

3.1 Turing Machines

- Alan Turing (1912-1954) 1935-36
- Turingin machine (TM) is similar to a finite automaton but with an unlimited and unrestricted memory (tape)
- TM can read and write the memory cells of the tape
- We can move to either direction in the tape
- The input to a TM is given in the beginning of the tape
- A TM has both an accepting final state, **accept**, and a rejecting one, **reject**
- The remainder of the tape is filled with blank characters (`□'), which cannot be a symbol in the input alphabet

- In one transition step a TM always reads one symbol and decides based on it and the state in which the machine finds itself
  - The new state,
  - The symbol that is written on the tape, and
  - The direction into which we move the tape head
- If the TM halts in the accepting final state, **accept**, the input string belongs to the language recognized by the TM
- If the TM halts in the rejecting final state, **reject**, or if it does not halt at all, then the input string does not belong to the to the language

Definition 3.3

A Turing machine is a 7-tuple
$M = (Q, \Sigma, \Gamma, \delta, q_0, \mathtt{accept}, \mathtt{reject})$, where

- Q is a finite set of states,
- Σ is the input alphabet, not containing the blank symbol, $\square \notin \Sigma$,
- Γ is the tape alphabet, where $\square \in \Gamma$ and $\Sigma \subseteq \Gamma$,
- $q_0 \in Q$ is the start state,
- $\mathtt{accept} \in Q$ is the accepting final state,
- $\mathtt{reject} \in Q$ is the rejecting final state, and
- $\delta: Q' \times \Gamma \to Q \times \Gamma \times \{L, R\}$ is the transition function, where

$Q' = Q \setminus \{\mathtt{accept}, \mathtt{reject}\}$

The value of transition function

$$\delta(q, a) = (q', b, \Delta)$$

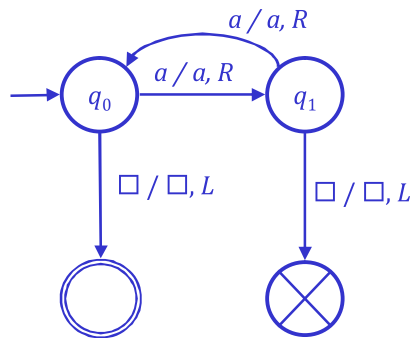
means that

- when the TM reads tape symbol a while in state q,
- it transfers to state q', writes symbol b into the tape cell that was just read, and moves the tape head one position to direction $\Delta \in \{L, R\}$

From the values of the transition function it is required that

1. You cannot move to the left of the leftmost cell in the tape. If that is ever tried, the tape head stays in the same place.
2. If $b = \square$, then $a = \square$ and $\Delta = L$.

A Turing machine for recognizing $\{ a^{2k} \mid k \geq 0 \}$:

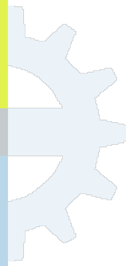


The configuration of a TM is denoted by $u\,q\,av$,
where

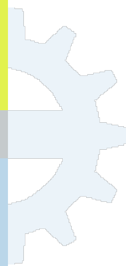
- $q \in Q$,
- $a \in \Gamma \cup \{ \varepsilon \}$
- $u, v \in \Gamma^*$

Intuitively

- The TM is in state q,
- The tape head is positioned at a cell containing symbol a,
- The contents of the tape from the left-most cell to the left of the tape head is u, and
- Its contents from the right of the tape head to the end of used space is v

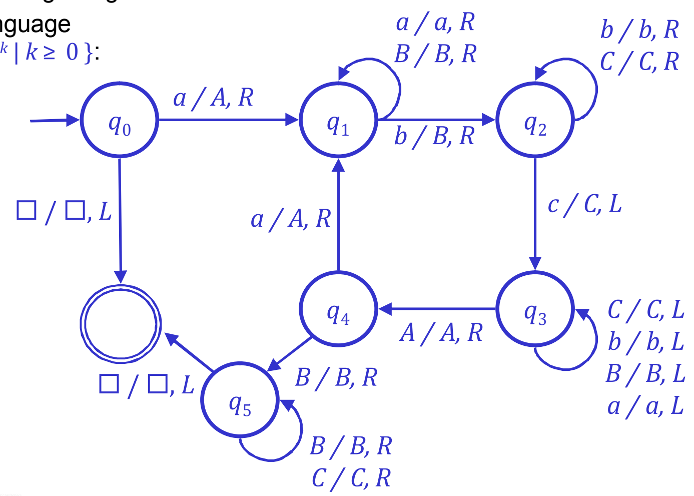


- The start configuration with input w is $q_0\, w$
- By $ua\, q_i\, bv \Rightarrow_M u\, q_j\, acv$ we denote that configuration $ua\, q_i\, bv$ *yields* configuration $u\, q_j\, acv$
- It is defined as follows
  - If $\delta(q_i, b) = (q_j, c, L)$, then $ua\, q_i\, bv \Rightarrow_M u\, q_j\, acv$,
  - if $\delta(q_i, b) = (q_j, c, R)$, then $ua\, q_i\, bv \Rightarrow_M uac\, q_j\, v$,
  - When the tape head is on the left-most cell, the configuration is $q_i\, bv$ and
    - transition $\delta(q_i, b) = (q_j, c, L)$ yields configuration $q_j\, cv$
    - transition $\delta(q_i, b) = (q_j, c, R)$ yields configuration $c\, q_j\, v$
  - When the tape head is on the right-most non-blank cell, the configuration is $ua\, q_i\, \square$ and
    - transition $\delta(q_i, \square)=(q_j, c, L)$ yields configuration $u\, q_j\, ac$
    - transition $\delta(q_i, \square) = (q_j, c, R)$ yields configuration $uac\, q_j\, \square$
    - only in the former situation can we have $c = \square$



- The value of the transition function is unspecified in final states **`accept`** and **`reject`**, and thus
  - configurations with **`accept`** or **`reject`** do not yield another configuration, instead the machine halts in them
- A Turing machine M accepts input w if a sequence of configurations $C_1, C_2, \ldots, C_k$ exists, where
  1. C_1 is the start configuration of M on input w,
  2. each C_i yields C_{i+1}, and
  3. C_k is an accepting configuration.
- The collection of strings that M accepts is the language of M, or the language recognized by M, denoted $L(M)$

A TM recognizing the language $\{a^k b^k c^k \mid k \geq 0\}$:



3.2 Variants of Turing Machines

The following generalizations of Turing machines do not change the collection of languages recognized

Multitrack machines

- The tape of a TM consists of k parallel tracks
- The TM still has only one tape head
- The machine reads and writes each track in each computation step (transition)
- We fill the contents of each track to be of equal length with a special empty symbol (#)
- The input is given at the left end of track 1

- A multitrack Turing machine is easy to simulate using a standard TM
- It is enough to have such a tape alphabet whose symbols can (if necessary) represent all possible k symbols on top of each other for a k-track TM
- For example:

$$\delta(q,(a_1,\ldots,a_k)) \rightarrow \hat{\delta}\left(q,\begin{bmatrix} a_1 \\ \vdots \\ a_k \end{bmatrix}\right)$$

- Doing the expansion of the alphabet systematically and changing all the symbols of the input

$$a \rightarrow \begin{bmatrix} a \\ \# \\ \vdots \\ \# \end{bmatrix}$$

into track 1, we can simulate the multitrack Turing machine with a standard one

Theorem *If a formal language can be recognized using a k-track Turing machine, then it can also be recognized with a standard Turing machine.*

Multitape Turing Machines

- Now a Turing machine may have k tapes, each with its own tape head
- The TM reads and writes each tape in each step of computation (transition)
- The tape heads move independent of each other
- The input is given at the left end of tape 1
- It is easy to simulate a multitape Turing machine using a multitrack one
- For each tape we have two tracks one of which corresponds to the tape and its contents and the other track holds a marker in the position of the tape head
- The marker is moved by simulating the movement of the tape head in the multitape TM

- By reading all tracks from left to right we find out all the symbols that are at the positions of the tape heads
- Then we know which symbols to write into the tapes and to which directions to move the tape heads
- We can implement/simulate the required changes in a right-to-left sweep over the tracks

Theorem 3.13 *Every multitape Turing machine has an equivalent single-tape (standard) Turing machine.*

Nondeterministic Turing Machines

- By giving the transition function of a Turing machine the potential of "prediction" we can define a nondeterministic Turing machine
- It is not a realistic model of mechanical computation, but it is useful in formal description of problems and to show their solvability
- Nondeterministic Turing machines have equivalent power in recognizing languages with the deterministic TMs.
- Nevertheless, they have a central role in computational complexity theory
- A nondeterministic TM accepts its input if some branch of the computation leads to the accept state

A nondeterministic Turing machine is a 7-tuple
$M = (Q, \Sigma, \Gamma, \delta, q_0, \texttt{accept}, \texttt{reject})$, where

$$\delta: Q' \times \Gamma \rightarrow \mathcal{P}(Q \times \Gamma \times \{L, R\}),$$

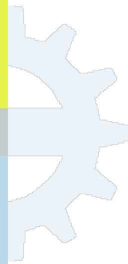
In which $Q' = Q \setminus \{\texttt{accept}, \texttt{reject}\}$

The value of the transition function

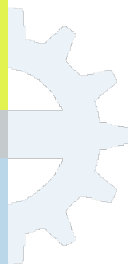
$$\delta(q, a) = \{ (q_1, b_1, \Delta_1), \ldots, (q_k, b_k, \Delta_k) \}$$

means that

- when reading symbol a while in state q
- the TM can choose the triple (q_i, b_i, Δ_i) as it pleases (whichever of them is best for its task)



- Recognizing nonnegative *composite numbers*: $n = pq$? ($p, q > 1$)
- If a number is not composite, then it is a prime number
- All known deterministic tests for composite numbers end up to go through a large number of potential factors in the worst case (basis for encryption)
- However, using a nondeterministic Turing machine "recognizing" composite numbers is easy
- The nondeterministic Turing machine does not give us an algorithm; it is only a computational description of composite numbers



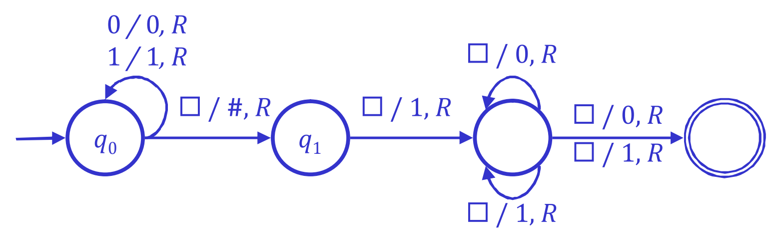
We need Turingin machines …

- CHECK-MULT recognizes the language
$$\{ n\#p\#q \mid n, p, q \in \{ 0, 1 \}^*, n = pq \}$$
- GEN-INT generates an arbitrary integer (>1) in binary to the end of the tape
- GO-START positions the tape head to the first memory cell of the tape

… to construct the TM

- TEST-COMPOSITE which recognizes the language
$$\{ n \in \{ 0, 1 \}^* \mid n \text{ is a composite number} \}$$

Turing Machine GEN-INT



Turing machine TEST-COMPOSITE