



Formal Languages and Automata

3

Theoretical Computer Science

Dipl.-Ing. Hubert Schölnast, BSc
September 20, 2021



Table of Contents

5	Type 1 grammars, languages and automata	3
5.1	Context-sensitive Grammars	3
5.1.1	Example for context-sensitive grammar	4
5.2	Linear Bounded Automaton (LBA)	5
5.2.1	Determinism	7
6	Type 0 grammars, languages and automata	7
6.1	Unrestricted Grammars	7
6.2	Turing Machine	8
6.2.1	Structure of a non-deterministic Turing machine (NTM)	9
6.2.2	Structure of a deterministic Turing machine (DTM)	9
6.2.3	Example	10
6.2.4	Church–Turing thesis	11

5 Type 1 grammars, languages and automata

Examples for languages that cannot be defined by context-free grammars and will not be recognized by pushdown-automata:

- $\mathcal{L}(G) = \{w \mid w = a^n b^n c^n, n \in \mathbb{N}\}$
- $\mathcal{L}(G) = \{w^* w^* \mid w \in \Sigma^*\}$
- $\mathcal{L}(G) = \{w \mid w = a^p, p \text{ is prime}\}$

When a context-free grammar creates a word, there always is only exactly 1 point where the word is growing. But creating a languages like $a^n b^n c^n$ or $w^* w^*$ is impossible, if there is only 1 point of growth.

A PDA has a memory, but it cannot multiply numbers, and therefore it has no idea what a prime number is. So, it will always be impossible for a PDA to recognize languages where the length of every word is a prime number.

So, when we want to have a more elaborate model of a computer, we need a more elaborate grammar.

5.1 Context-sensitive Grammars

Let us recall again the general definition of a grammar:

A grammar is a set with 4 elements:

$$G = \{V, \Sigma, P, S\}$$

- G = a grammar
- V = the alphabet of variables (auxiliary signs = non-terminal signs)
- Σ = the alphabet of final (terminal) signs
- P = the set of production rules
- S = the start sign (actually a start word that consists of exactly 1 sign)

V and Σ are disjoint: $V \cap \Sigma = \emptyset$

S is a word consisting of exactly one sign from V (in most books written as $S \in V$)

Each element of P is a rule, and each rule looks like this:

$$l \rightarrow r$$

with


$$l \in (V \cup \Sigma)^+$$

$$r \in (V \cup \Sigma)^*$$

Additional restrictions that define different Chomsky-types:

Type 3 (regular grammars):

$$l = A; A \in V \quad |l| = 1$$

$$r = \begin{cases} aB; a \in \Sigma, A \in V & |r| = 2 \\ \varepsilon & |r| = 0 \end{cases}$$


Type 2 (context-free grammars):

$$l = A; A \in V \quad |l| = 1$$

$$r \in (V \cup \Sigma)^* \quad |r| \geq 0$$


Type 1 (context-sensitive grammars):

$$l \in (V \cup \Sigma)^+ \quad |l| \geq 1$$

$$r \in (V \cup \Sigma)^* \quad |r| \geq |l|$$


Exception: You can have the rule $S \rightarrow \varepsilon$ if for all other rules $r \in ((V \setminus \{S\}) \cup \Sigma)^*$ which means "no S in the right side of any rule"

5.1.1 Example for context-sensitive grammar

- $G = \{V, \Sigma, P, S\}$
- $\Sigma = \{a, b, c\}$
- $V = \{A, B, C\}$
- $P = \{A \rightarrow aABC \mid aBC, CB \rightarrow BC, aB \rightarrow ab, bB \rightarrow bb, bC \rightarrow bc, cC \rightarrow cc\}$
- $S = A$

Production rules with numbers

1	$A \rightarrow aABC$
2	$A \rightarrow aBC$
3	$CB \rightarrow BC$
4	$aB \rightarrow ab$
5	$bB \rightarrow bb$
6	$bC \rightarrow bc$
7	$cC \rightarrow cc$

Let's try some runs

rule	word
	A $\notin \Sigma^*$
2 $A \rightarrow aBC$	↓
	aBC $\notin \Sigma^*$
4 $aB \rightarrow ab$	↓
	abC $\notin \Sigma^*$
6 $bC \rightarrow bc$	↓
	abc $\in \Sigma^*$

rule	word
	A $\notin \Sigma^*$
1 $A \rightarrow aABC$	↓
	aABC $\notin \Sigma^*$
2 $A \rightarrow aABC$	↓
	aaBCBC $\notin \Sigma^*$
3 $CB \rightarrow BC$	↓
	aaBBCC $\notin \Sigma^*$
4 $aB \rightarrow ab$	↓
	aabBCC $\notin \Sigma^*$
5 $bB \rightarrow bb$	↓
	aabbCC $\notin \Sigma^*$
6 $bC \rightarrow bc$	↓
	aabbcc $\notin \Sigma^*$
7 $cC \rightarrow cc$	↓
	aabbcc $\in \Sigma^*$

rule	word
	A $\notin \Sigma^*$
1 $A \rightarrow aABC$	↓
	aABC $\notin \Sigma^*$
1 $A \rightarrow aABC$	↓
	aaABCBC $\notin \Sigma^*$
2 $A \rightarrow aABC$	↓
	aaaBCBCBC $\notin \Sigma^*$
3 $CB \rightarrow BC$	↓
	aaaBCBBCC $\notin \Sigma^*$
3 $CB \rightarrow BC$	↓
	aaaBBBCCC $\notin \Sigma^*$
3 $CB \rightarrow BC$	↓
	aaaBBBCCC $\notin \Sigma^*$
4 $aB \rightarrow ab$	↓
	aaabBBCCC $\notin \Sigma^*$
5 $bB \rightarrow bb$	↓
	aaabbBBCCC $\notin \Sigma^*$
5 $bB \rightarrow bb$	↓
	aaabbbCCC $\notin \Sigma^*$
6 $bC \rightarrow bc$	↓
	aaabbbcCC $\notin \Sigma^*$
7 $cC \rightarrow cc$	
	aaabbbccC $\notin \Sigma^*$
7 $cC \rightarrow cc$	↓
	aaabbbccc $\in \Sigma^*$

$$\mathcal{L}(G) = \{abc, aabbcc, aaabbbccc, aaaabbbbcccc, aaaaabbbbbccccc, \dots\}$$

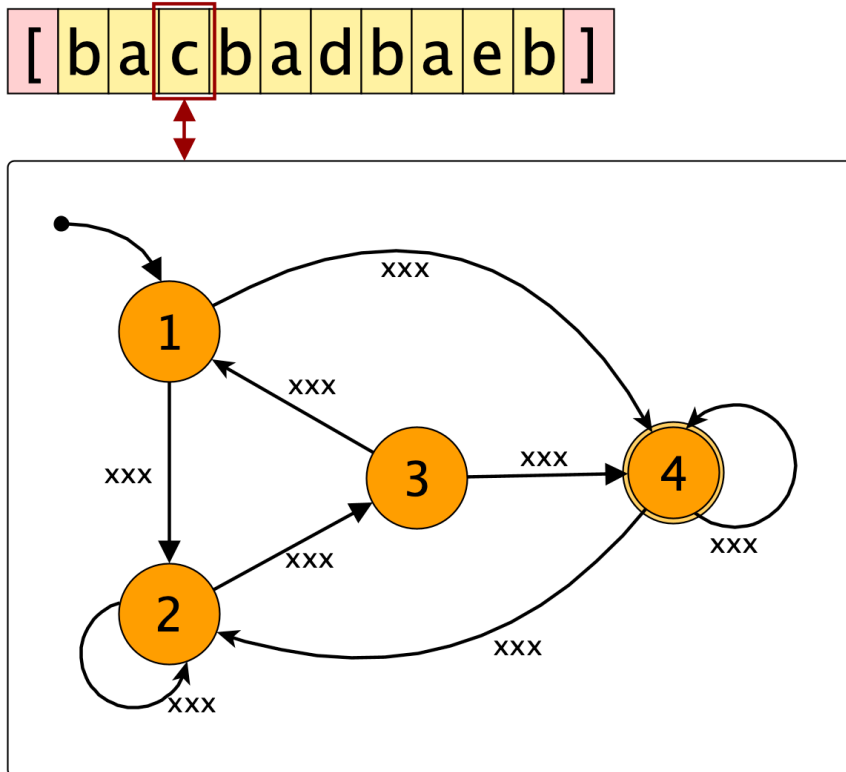
$$\mathcal{L}(G) = \{w \mid w = a^n b^n c^n\}$$

As you can see, this grammar produces exactly one of the languages that cannot be formed with context-free grammars. You need the context and you need to be able to change the context to form words from this language.

5.2 Linear Bounded Automaton (LBA)

The automaton that is associated with context-sensitive languages is called "linear bounded automaton". But because it is just a linear bounded version of a Turing machine, you also will find the term "linear bounded Turing machine". So, both terms mean the same.

A linear bounded automaton (LBA) looks very similar to the pushdown automaton (PDA), but there are some important differences:



First of all: A LBA has no stack and therefore no stack bottom sign. Instead this type of machine uses the tape not only to read the input, but it uses the tape also as it's memory, This means, this machine can write signs to the tape.

The head, which is now a read-write-head, can not only move to the right, it also can move to the left, or it can stay where it was.

The tape still has a fixed and limited length, like in PDA and FSM, but now, both ends are marked with special signs, similar to the stack bottom sign. If you want you can add these signs explicitly to a PDS or FSM too, but this will not change how these machines work. The LBA can read this signs, but this signs cannot be replaced by other signs. This signs also cannot be written to other positions, and the head cannot move beyond these signs.

The total number of characters on the tape between the two end marks can be either the length of the input word or the result of any linear function from that length.

$$l_{tape} = a \cdot l_{input} \quad a \geq 1$$

This means, that the usable "memory" of this type of machine is as big as you want, but it is still limited. This may sound like a contradiction, but it isn't.

Here is why: Let's say, you can use this automaton to solve a problem for an input that is $l_{input} = 10$ signs long and to solve this problem you need $l_{tape} = 40$ signs on the tape. So, $a = 4$. If your problem is of a kind, where you need a maximum number of 400 signs to solve the problem for an input length of 100 signs, then it can be solved with a linear bounded automaton, and then there is a context-sensitive grammar to describe the problem.

But when you need $l_{tape} > 1,000,000$ signs on the tape to solve a the problem for $l_{input} = 100$, then this problem belongs to a class that cannot be solved on a linear bounded automaton.

Since this type of automaton is a Turing machine with a limited tape, it makes more sense to talk about this type of automaton in the section about type 0 automata. But there still is something to say specifically about LBAs:

5.2.1 Determinism

The state machine in an LBA can be deterministic or nondeterministic.

For type 3 automata (finite state machines) you have seen an algorithm that allows to convert any NFSM to a DFSM that can recognize exactly the same language. And this means, that there is no difference between nondeterministic and deterministic regular languages.

For type 2 automata (pushdown automata) you have seen, that there are some languages produced by context-free grammars where it is impossible to recognize them with a deterministic machine (an example for these languages are palindromes). As a consequence, the set of deterministic context-free languages is a proper subset of the set of all context-free languages, which therefor also is often named "nondeterministic context-free languages".

So, how is this situation with type 1 automata (LBA)?

Well, this still is a subject of research where nobody has found an answer yet.

- Until now, nobody found an algorithm that would make it possible to convert every nondeterministic context-sensitive grammar into a deterministic context-sensitive grammar. But only the existence of such an algorithm would prove, that deterministic and nondeterministic languages are equal.
- Until now, nobody found a context-sensitive language, that could not be recognized by a deterministic linear bounded automaton. But only the existence of such a language would prove, that deterministic and nondeterministic languages are different.

6 Type 0 grammars, languages and automata

6.1 Unrestricted Grammars

Unrestricted grammars are grammars where no additional constraints are added

Type 0 (unrestricted grammars):

$$l \in (V \cup \Sigma)^+ \quad |l| \geq 1$$

$$r \in (V \cup \Sigma)^* \quad |r| \geq 0$$



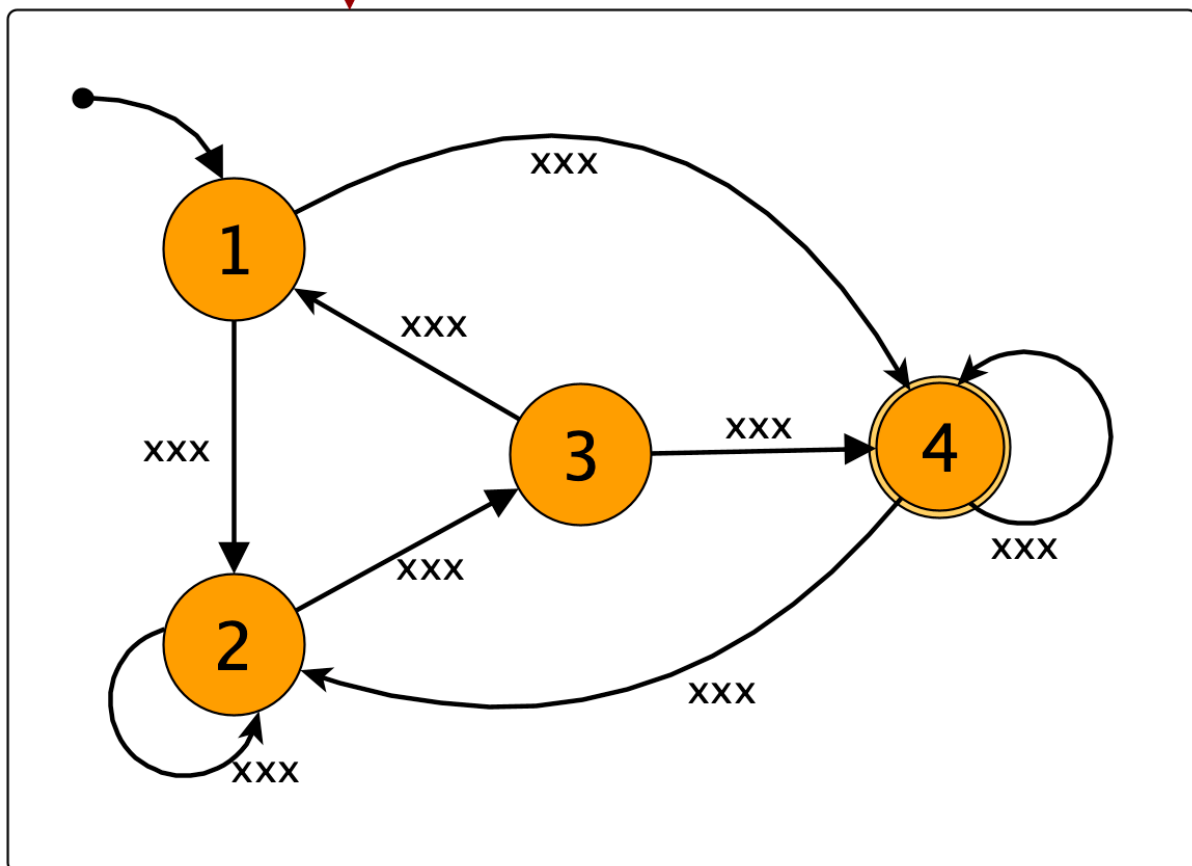
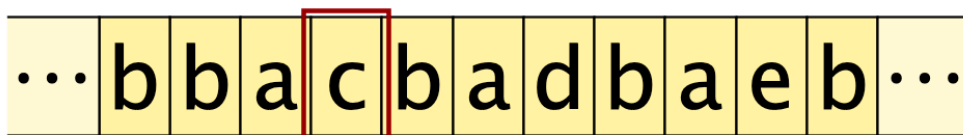
No exceptions: Anything is allowed.

There is not much to say about unrestricted grammars, and also in other textbooks unrestricted grammars are hardly mentioned, and you will not find explicit examples for proper unrestricted grammars, i.e. for type-0-grammars that are not type-1-grammars. This is because the magic that is in type 0 of Noam Chomsky's hierarchy is not in the grammar but in the automaton.

6.2 Turing Machine

This automaton is named after the British logician, mathematician, cryptanalyst, and computer scientist Alan Turing (1912-1954). He is one of the most important pioneers in computer science. During World War II, he was instrumental in deciphering German radio messages encrypted with the German Enigma rotor cipher machine, but also his contributions to theoretical biology also proved to be groundbreaking.

There are many variations of definitions of a Turing machine. The most common is this:



A Turing machine is a LBA where the tape has no limits. Every position on the tape has a left and a right neighbor, and the head can move in both directions as far as it wants. (The head moves just 1 position in every single step. But step by step it can go as far as it wants.)

6.2.1 Structure of a non-deterministic Turing machine (NTM)

A is a non-deterministic Turing machine (NTM), and every NTM is a set with exactly 7 elements:

$$A = \{Z, Z_0, E, \Sigma, \Gamma, \delta, \#\}$$

Z is the set of all states (the cardinality of Z is finite)

Z_0 is the initial state $Z_0 \in Z$

E is the set of all accepting states. $E \subseteq Z$

Σ is the input alphabet. Only signs from this alphabet can be used to write the input word

Γ is the work alphabet. The automaton can write signs from this alphabet on any position of the tape. Γ is a proper superset of Σ ($\Gamma \supset \Sigma$). As all alphabets also Γ (and therefore also its subset Σ) is a finite set.

δ is a relation: $\delta \subseteq (Z \times \Gamma) \times (Z \times \Gamma \times \{L, H, R\})$ It takes as "input" an element from the Cartesian product $Z \times \Sigma$ (a combination of state and sign) and produces as "output" zero, one or many combinations of a new state, a new sign and a direction (left, halt, right).

$\#$ is the blank sign. Before the machine starts to operate, the entire tape is filled with this blank sign, only the section where the input is written contains other signs. The blank sign is a member of Γ but not of Σ : $\# \in \Gamma, \# \notin \Sigma$

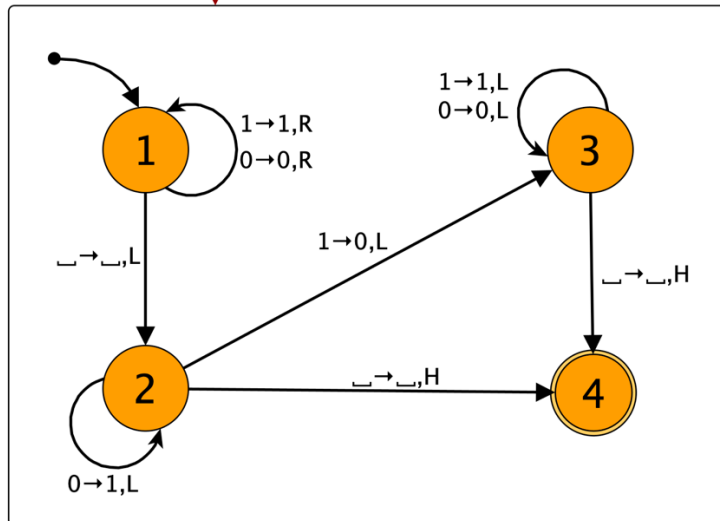
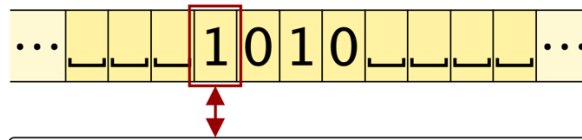
6.2.2 Structure of a deterministic Turing machine (DTM)

As before, but

δ is a function: $\delta: (Z \times \Gamma) \rightarrow (Z \times \Gamma \times \{L, H, R\})$ It takes as "input" an element from the Cartesian product $Z \times \Sigma$ (a combination of state and sign) and produces as "output" a new state, a new sign and a direction (left, halt, right).

6.2.3 Example

- $TM = \{Z, Z_0, E, \Sigma, \Gamma, \delta, \#\}$
- $Z = \{\textcircled{1}, \textcircled{2}, \textcircled{3}, \textcircled{4}\}$
- $Z_0 = \textcircled{1}$
- $E = \{\textcircled{4}\}$
- $\Sigma = \{0,1\}$
- $\Gamma = \{0,1, _ \}$
- $\delta = \left\{ \begin{array}{l} (\textcircled{1}, 0) \rightarrow (\textcircled{1}, 0, R), \\ (\textcircled{1}, 1) \rightarrow (\textcircled{1}, 1, R), \\ (\textcircled{1}, _) \rightarrow (\textcircled{2}, _, L), \\ (\textcircled{2}, 0) \rightarrow (\textcircled{2}, 1, L), \\ (\textcircled{2}, 1) \rightarrow (\textcircled{3}, 0, L), \\ (\textcircled{2}, _) \rightarrow (\textcircled{4}, _, H), \\ (\textcircled{3}, 0) \rightarrow (\textcircled{3}, 0, L), \\ (\textcircled{3}, 1) \rightarrow (\textcircled{3}, 1, L), \\ (\textcircled{3}, _) \rightarrow (\textcircled{4}, _, H), \end{array} \right.$
- $\# = _$



The same automaton, just in a different syntax:

```

init: ①
accept: ④

①,0
①,0,>

①,1
①,1,>

①,_
②,<,

②,0
②,1,<

②,1
③,0,<

②,_
④,<,-

③,0
③,0,<

③,1
③,1,<

③,_
④,<,-
    
```

Go to <https://turingmachinesimulator.com>, copy the blue code from here, paste it into the simulator, compile it, enter 1010 as input, press play and see what happens.

About the programming language of the simulator:

- When this simulator compiles the program, it finds out what Z , Σ and Γ are, so you don't have to explicitly specify them. You just have to specify Z_0 (`init`), E (`accept`) and δ (the rest of the program).
- # is always a blank in the tape but an underline character (`_`) in the program.
- The simulator uses different names for the directions in which the head can move:
 - R is `>`
 - L is `<`
 - H is `-`

Each element of the function is written in two lines in the program. The first line is for what we have written in the first pair of brackets, this is the condition under which this rule can be applied. The second line tells the machine what to do when this condition is true, this is what we have written in the second bracket.

This Turing machine accepts all binary numbers and rejects any string that is not a binary number. This means, that this Turing machine accepts a regular language that can be written with this regular expression:

$$(0|1)^*$$

This is not really exciting. Also the grammar is just a boring regular grammar.

But did you notice what happened in the tape? Enter different binary strings and compare it with the string that stands in the tape at the end.

This is what Turing machines are about: They can compute. A Turing machine is a computer and δ is its program. And it's not just a simple programmable pocket calculator. A Turing machine is an universal computer:

6.2.4 Church–Turing thesis

Named after Alan Turing and Alonzo Church (1903-1995), American mathematician, logician and philosopher and PhD-supervisor of Alan Turing.

The class of Turing-computable functions coincides with the class of intuitively computable functions.

In other words:

Everything that can be computed can be computed on a Turing machine.