Formal Grammars for Generation

Emmanouil Karystinaios - Silvan David Peter

JOHANNES KEPLER UNIVERSITY LINZ

Kunstuniversität Linz
University of Arts Linz

Institute of Computational Perception

Organization

Introduction:

1. Formal Grammars
2. Derivations
3. Context Free Grammars

Examples

4. Linguistics
5. Music

Generative Theory of Tonal Music:

6. Harmonic Analysis
7. Rhythm
8. Variations

Practical Implementations

9. Prolog & Haskell
10. Python

kunst universität linz www.ufg.at

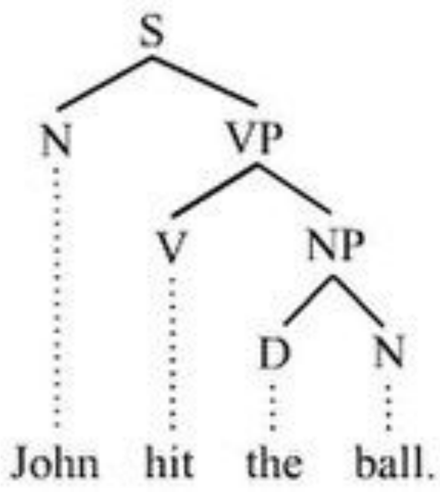
Formal Grammar

A **Formal Grammar** describes how to form strings from a language's alphabet that are valid according to the language's syntax. It is defined by a quadruple G = (T, N, P, S)

- T a vocabulary of Terminal symbols
- N a vocabulary of non-terminal symbols disjoint from T
- P a set of production rules
- S a set of starting symbols

Grammar Definition through Examples

- **S** - Sentence
- **D** - Determiner
- **N** - Noun
- **V** - Verb
- **NP** - Noun Phrase
- **VP** - Verb Phrase



Constituency-based parse tree

Informal Definition

A **formal grammar** is a set of rules for rewriting strings, along with a "start symbol" from which rewriting starts. Therefore, a grammar is usually thought of as a language generator.

Given a set of production rules and some symbols we can create a "language".

Given a grammar we can decide if a string is valid.

Example in computer languages:
- decide if a line of code x is valid in language y.
- mathematical formulation: can y generate x?

Terminology

- **Alphabet** = a set of things, for example characters.
- **Grammar** = a set of rules/restrictions on *alphabets*.
- **Word** = is made from elements from the *alphabet*.
- **Language** = a set containing *words*.

Turing Machine

A Turing machine is a theoretical machine that manipulates symbols on a tape strip, based on a table of rules. Even though the Turing machine is simple, it can be tailored to replicate the logic associated with any computer algorithm.

It is also particularly useful for describing the CPU instructions within a computer.

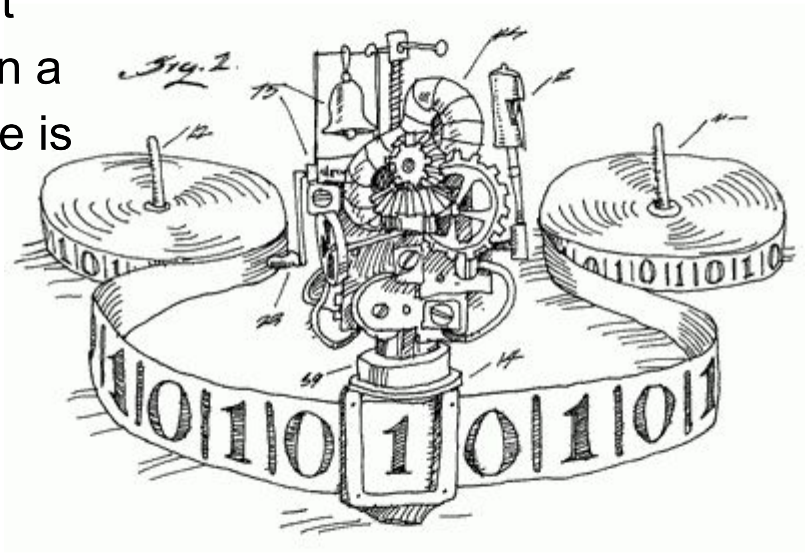


Image from:
https://www.qwizbowl.com/post/qwiz5-quizbowl-essentials-turing-machine

Chomsky Hierarchy

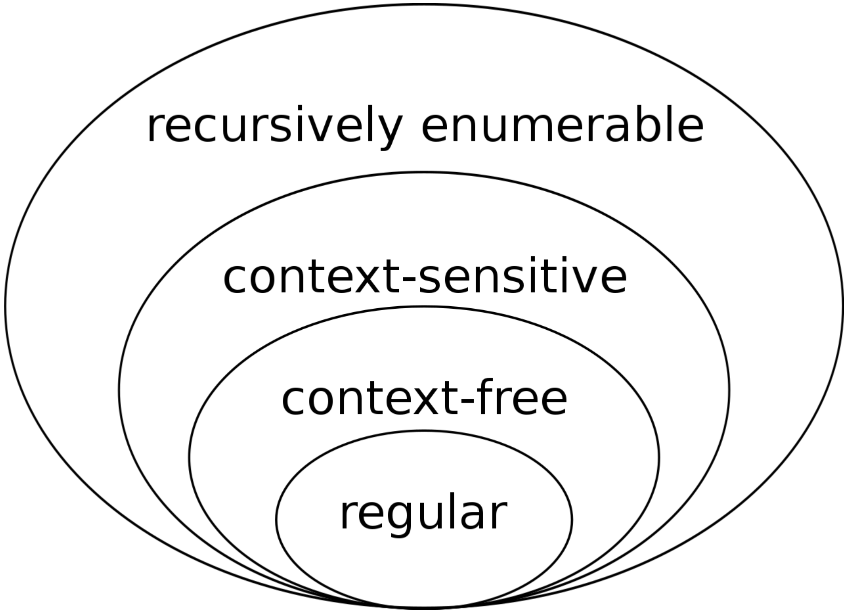


Figure by J. Finkelstein - Own work, CC BY-SA 3.0,
https://commons.wikimedia.org/w/index.php?curid=9405226

The Chomsky Hierarchy is used as an organization of formal grammars. Starting with grammars that are equally powerful to a Turing machine and going to simple variants.

Recursively Enumerable

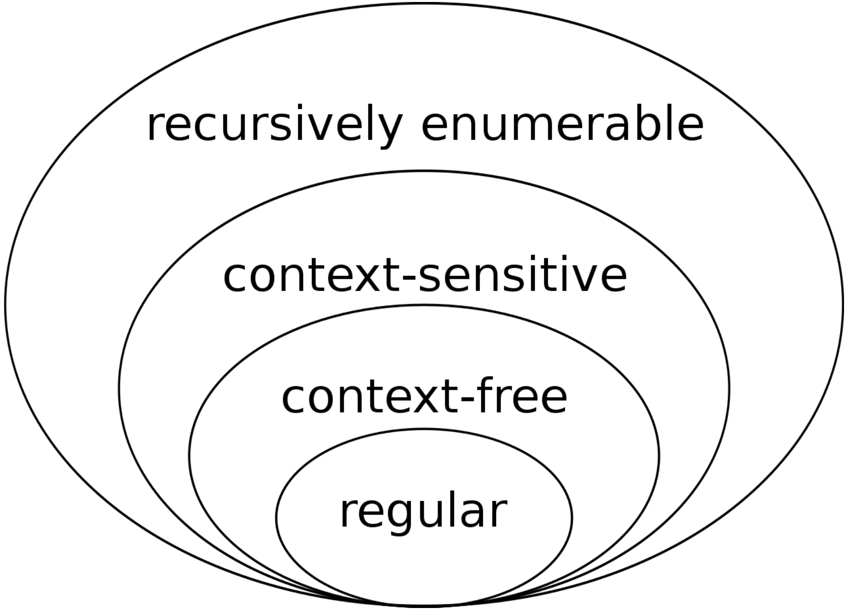


Figure by J. Finkelstein - Own work, CC BY-SA 3.0,
https://commons.wikimedia.org/w/index.php?curid=9405226

Recursively Enumerable Grammars or **Unrestricted** Grammars contain all formal grammars they can generate all languages that a Turing machine can recognise.

To the extent of our interest these Grammars can generate all the languages we could potentially be interested on.

Context Sensitive

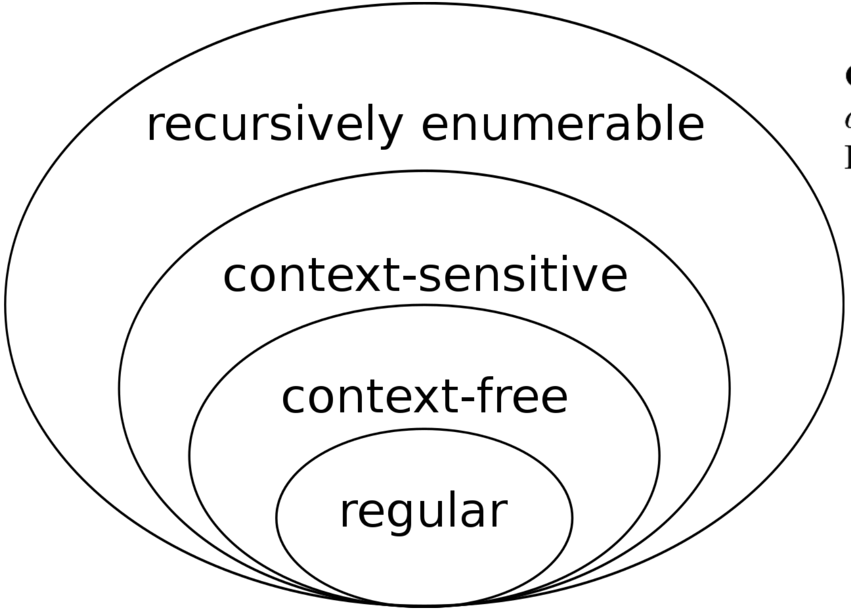


Figure by J. Finkelstein - Own work, CC BY-SA 3.0,
`https://commons.wikimedia.org/w/index.php?curid=9405226`

Context Sensitive grammars they have derivations in the form $asc \to abc$ where $s \in V_N$ and $a, b, c \in V_T$.
In other words, their derivations depend on context.

Context Free

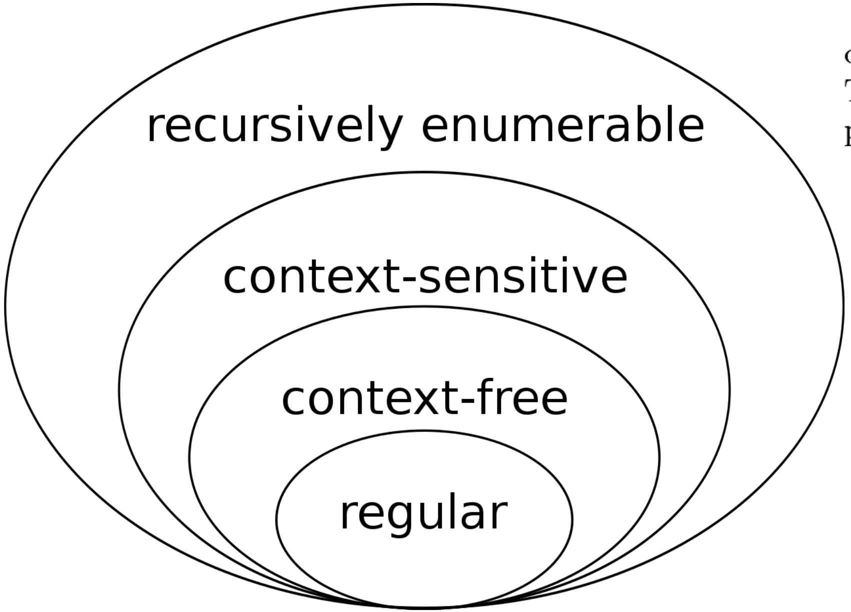


Figure by J. Finkelstein - Own work, CC BY-SA 3.0,
`https://commons.wikimedia.org/w/index.php?curid=9405226`

Context Free or **Context independent** grammars do not depend to context and all rules are like the previous definitions. These grammars are particularly interesting because the can generate precisely the languages that a programming language can emulate.

Rules of this grammar involve the reduction of one non-terminal symbol:

They are of the form S -> wS'w'.

Regular grammars

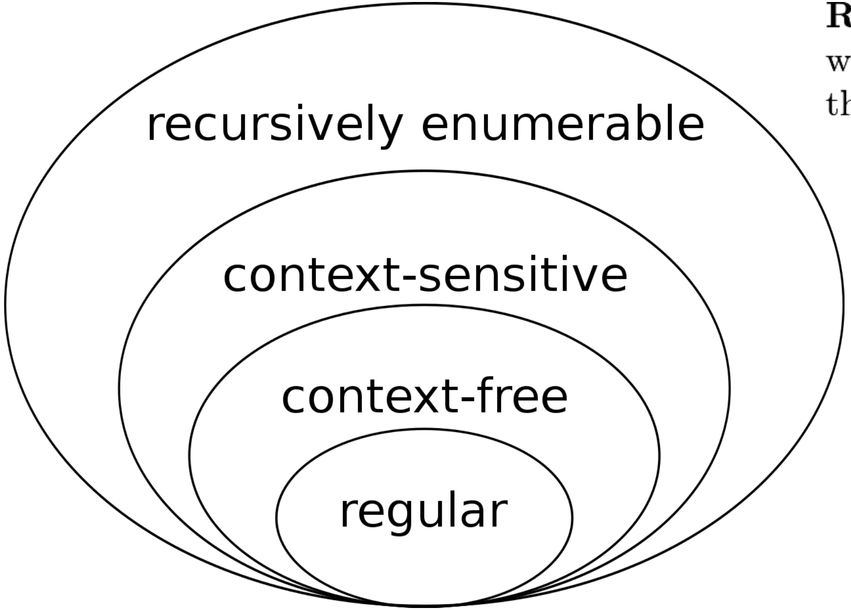


Figure by J. Finkelstein - Own work, CC BY-SA 3.0,
`https://commons.wikimedia.org/w/index.php?curid=9405226`

Regular grammars generate Regular languages which are commonly used to define search patterns and the lexical structure of programming languages.

Context-Free Grammars

For our purpose we are interested on Context-free grammars because they are the most general grammars that can guarantee a finite computation with a given upper bound on parsing time.

In particular we can decide if a word belong to a language given a grammar in Polynomial time.

Let’s see a algorithm that explains the parsing process.

Context-Free LL Parsing

Consider the grammar G :

- $V_T = \{(,), a, +, \$\}$ terminal symbols
- $V_N = \{S, F\}$ non terminal symbols
- P production rules:
  1. $S \rightarrow F$
  2. $S \rightarrow (S + F)$
  3. $F \rightarrow a$
- S starting symbol

We want to decide if the word $(a + a)$ can be generated by G.

LL Parsing Example

Following the Leftmost parsing: $(a + a)\$$

	(	)	**a**	+	\$
S	P_2	-	P_1	-	-
F	-	-	P_3	-	-

Starting from S following the table the parser has to rewrite using the P_2:

$$S \rightarrow (S + F)$$

So we create a stack :

$$\text{stack} = [(, S, +, F,), \$]$$

kunst universität linz

LL Parsing Example

Following the Leftmost parsing: $(a + a)\$$

	(	)	**a**	+	\$
S	P_2	-	P_1	-	-
F	-	-	P_3	-	-

Leftmost character matches with (in the stack so we remove it:

$$\textit{stack}\ \text{pop}$$

Now the stack becomes

$$\text{stack} = [S, +, F,), \$]$$

LL Parsing Example

Following the Leftmost parsing: $(a + a)\$$

	(	)	**a**	+	\$
S	P_2	-	P_1	-	-
F	-	-	P_3	-	-

Leftmost character is an S but we need an a so we apply P_1:

$$S \rightarrow F$$

Now the stack becomes

$$\text{stack} = [F, +, F,), \$]$$

LL Parsing Example

Following the Leftmost parsing: $(a + a)\$$

	(	)	**a**	+	\$
S	P_2	-	P_1	-	-
F	-	-	P_3	-	-

Leftmost character is an F but we need an a so we apply P_3:

$$F \to a$$

Now the stack becomes

$$\text{stack} = [a, +, F,), \$]$$

LL Parsing Example

Following the Leftmost parsing: $(a + a)\$$

	(	)	**a**	+	\$
S	P_2	-	P_1	-	-
F	-	-	P_3	-	-

Two next leftmost characters match the output so we remove them from stack:

$$\text{stack pop pop}$$

Now the stack becomes

$$\text{stack} = [F,), \$]$$

kunst universität linz

LL Parsing Example

Following the Leftmost parsing: $(a + a)\$$

	(	)	**a**	+	\$
S	P_2	-	P_1	-	-
F	-	-	P_3	-	-

The leftmost character is a F we apply P_3 to transform it to a :

$$F \to a$$

Now the stack becomes:

$$\text{stack} = [a,), \$]$$

LL Parsing Example

Following the Leftmost parsing: $(a + a)\$$

	(	)	**a**	+	\$
S	P_2	-	P_1	-	-
F	-	-	P_3	-	-

All characters match so we remove them

$$\text{stack pop all}$$

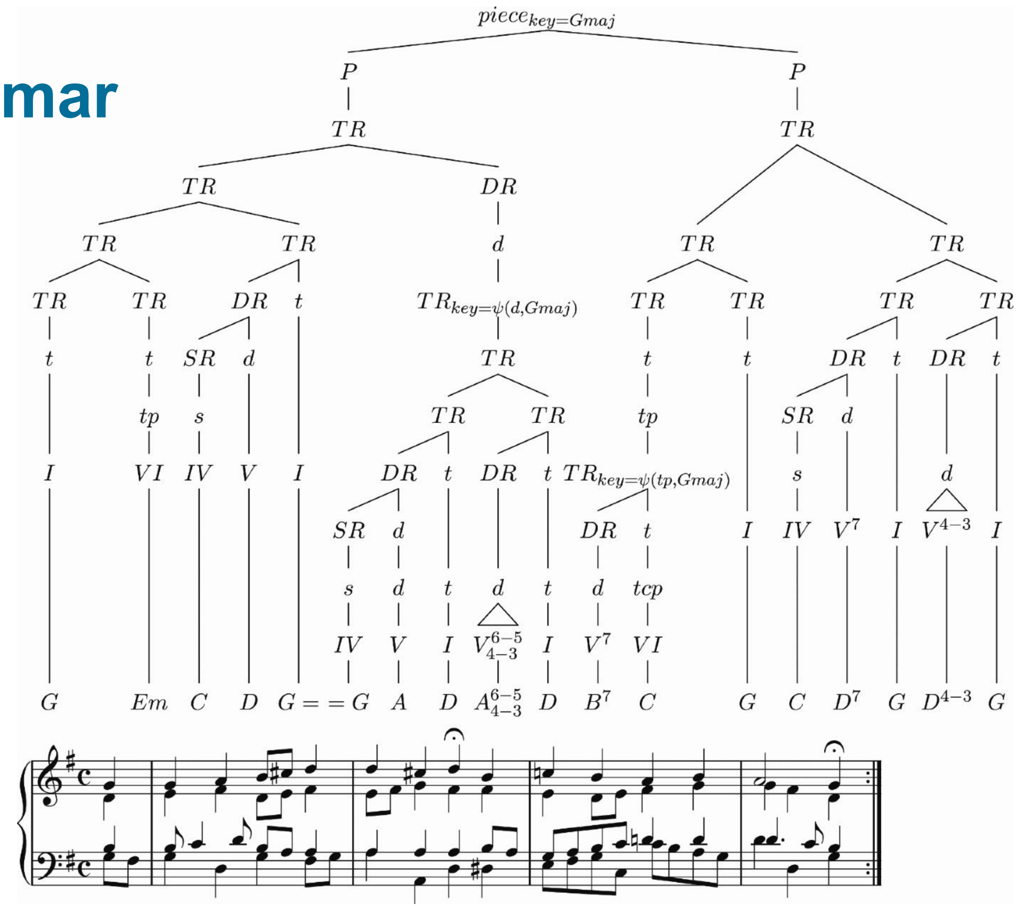
The stack is empty and we can conclude:

$$\text{stack} = empty$$

kunst universität linz

Harmonic Analysis Grammar

Harmonic analysis of the beginning of Bach’s chorale ‘Ermuntre Dich, mein schwacher Geist’, mm.1–4



from Towards a generative syntax of tonal harmony by Martin Rohrmeier

Exercise

Find a context free grammar that generates the language:

$$L = \{a^i b^j \mid i, j \in \mathbb{N} \text{ and } j \geq i\}$$

Minimal grammar only has 3 rules.

Hint start with *S*

kunst universität linz www.ufg.at

Generative Theory of Tonal Music (GTTM)

A theory conceived by Fred Lerdahl during the 1980's was inspired by Generative grammars and linguistics. It consists with a list of production rules (loosely defined) and a harmonic, rhythmic and melodic vocabulary.

Tree like analysis of music is performed in 4 levels according to the rules. However this approach was never computational. Some have tried to create a automatic version of the theory but until now the complexity of the music and the analysis have made it very difficult.

Example

$$i - \quad - \quad - - - - - V - - - - - -$$

$$i - \quad - \quad - - - iv - V - - - - - -$$

$$i - \quad - \quad - i^6 - iv - V - - - - - -$$

$$i - vii^0 \, {}^{6}_{5} - i^6 - iv - V - - - i - V$$

$$i - vii^0 \, {}^{6}_{5} - i^6 - iv - V - V^6 - i - V$$

Context-free grammar for eight-bar chord sequences

- $V_T = \{\ I, III, IV, V\ \}$
- $V_N = \{$ 8-bars, 4.1, 4.2, opening-cadence, opening-cadence$'$ middle-cadence $\}$
- $s =$ 8-bars

8-bars	$\rightarrow$	4.1	4.2
4.1	$\rightarrow$	Opening-cadence	Opening-cadence
4.1	$\rightarrow$	Opening-cadence$'$	Opening-cadence
4.2	$\rightarrow$	Middle-cadence	Opening-cadence
Opening-cadence	$\rightarrow$	$\mid\ I\ \mid\ I\ \mid$	
Opening-cadence	$\rightarrow$	$\mid\ I\ \mid\ V\ \mid$	
Opening-cadence$'$	$\rightarrow$	$\mid\ I\ \mid\ III\ \mid$	
Opening-cadence$'$	$\rightarrow$	$\mid\ I\ \mid\ IV\ \mid$	
Middle-cadence	$\rightarrow$	$\mid\ I\ \mid\ IV\ \mid$	
Middle-cadence	$\rightarrow$	$\mid\ I\ \mid\ V\ \mid$	
Middle-cadence	$\rightarrow$	$\mid\ IV\ \mid\ I\ \mid$	

Resources

- Formal Grammars in Prolog:

  https://swish.swi-prolog.org/example/grammar.pl

- The Haskell School of Music :

  https://www.cs.yale.edu/homes/hudak/Papers/HSoM.pdf

- Simple CFG Generator in Python:

  https://github.com/Hevia/GramPy

- Use Markov Chain to weight CFG production rules (Python):

  https://github.com/williamgilpin/cfgen

Exercise

Modify gramPy Production rules to generate sentences such as:

- *The funky cat barks at the fat bear.*
- *The spooky lion sings, and the blue frog moos.*
- *The fat cat sings at the ugly turtle, and the stinky lizart yawns, and the …*