What is a grammar

Here is an {\it informal} definition: A {\bf grammar} for a language \( L \) is a {\bf finite set of rules} for generating the strings in \( L \). This is, any given string in \( L \) can be built by a particular sequence of applications of the rules in the grammar

- A formal definition of grammar requires a closer look into the concepts of rule, variable and terminal symbols
Rules

A **rule** is a relation between two strings $s$ and $t$, usually denoted

$$ s \rightarrow t $$

instead of the usual mathematical notation of $(s, t)$

The relation $s \rightarrow t$ is called **production**
and read “$s$ produces $t$” or “$t$ replaces $s$”
Variables and terminals

The strings involved in this relation are formed with symbols in the language’s alphabet, called **terminals**, and some additional symbols called **variables**

**Illustration:**

Assume that the language’s alphabet is \{0, 1\} and that the variables are \(s, u, \text{ and } v\). Then,

\[0u11v0, s, svu1\]

are examples of strings that may appear in a production
Formal definition of grammar

Definition: A grammar is a quadruple
\[ G = (V, A, R, s) \]
where
- \( V \) is a finite set of symbols called variables,
- \( A \) is the alphabet of the language,
- \( R \) is the finite set of rules, and
- \( s \) is an element in \( V \) called start variable.
Example of a grammar

Here is a (formally defined) grammar:

\[ G = (\{s\}, \{0, 1\}, \{s \rightarrow \lambda, s \rightarrow 0s, s \rightarrow 1s\}, s) \]
Derivations

The central operation with a grammar is the so-called string derivation, or simply, derivation

**Definition:** A *derivation* is a sequence of productions that begins with the start variable and ends with a string of terminals (no variables allowed in the final string)

Here is a derivation with the previous grammar:

\[ s \rightarrow 0s \rightarrow 01s \rightarrow 010s \rightarrow 010 \]
The language of a grammar

**Definition:** The *language of a grammar* $G$, denoted $L(G)$, is the set of all terminal strings that can be derived with $G$.

This is, if $G = (V, A, R, s)$, then,

$$L(G) = \{s: s \text{ in } A^* \text{ and } s \text{ derived with the rules in } R\}$$

Two common questions:

- **Given a grammar, What is its language?**
- **Given a language, Can it be generated with a grammar?**
Problem: Find \( L(G) \) for the grammar
\[
G = (\{s\}, \{0, 1\}, \{s \rightarrow 0s1, s \rightarrow \lambda\}, s)
\]
Answer: Let’s examine some typical productions of this grammar:
\[
s \rightarrow 0s1 \rightarrow 00s11 \rightarrow 000s111 \rightarrow 000111
\]
\[
s \rightarrow \lambda
\]
From these examples it can be inferred that
\[
L(G) = \{0^n1^n: n \text{ natural}\}\
\]
Examples (cont.)

What about a finite language?

Problem: Is there a grammar for generating the language $L = \{00, 01, 10, 11\}$?

Answer: Yes, there is

$G = (\{s\}, \{0, 1\}, \{s \rightarrow 00, s \rightarrow 01, s \rightarrow 10, s \rightarrow 11\}, s)$

Remark: Every finite language can be generated by a grammar. However, there are many infinite languages that cannot be generated by a grammar.
Recursive productions

**Definition:** A **rule is recursive** if it has a repeated variable on both sides of the arrow (i.e. both strings in the relation)

**Example:** \( v \rightarrow 0s1v1 \) is a recursive rule

**Definition:** A **derivation is recursive** if two strings in its sequence of productions have a common variable

**Example:** \( s \rightarrow 01v \rightarrow 011s \rightarrow 011 \) is a recursive production
Recursive grammars

**Definition**: A grammar $G = (V, A, R, s)$ is said to be **recursive** if it can generate a recursive production.

**Examples**:

- $G = (\{s\}, \{0, 1\}, \{s \rightarrow 0s1, s \rightarrow \lambda\}, s)$ is recursive since it contains a recursive rule.

- $G = (\{s, u\}, \{0, 1\}, \{s \rightarrow 0u, u \rightarrow 1s, s \rightarrow \lambda\}, \{s\})$ is recursive since it can generate, for instance, the production $s \rightarrow 0u \rightarrow 01s \rightarrow 01$. 
A first principle

The previous examples illustrate the following theoretical fact:

**Theorem:** A grammar for an infinite language must be recursive.

(Yes. Otherwise, how could you generate (potentially) an infinite amount of strings with a finite number of rules?)
A second principle

**Theorem**: Any language generated by a grammar is an inductively defined language

**Remark**: The practical side of this principle is that any string in a language generated by a grammar can be obtained with an inductive procedure. This is indeed very practical if the induction is simple
Let $G = (\{s, v\}, \{0, 1\}, \{s \rightarrow \lambda \mid 0v, v \rightarrow 1 \mid 1v\}, s)$. It is easy to see that $L(G) = \{01^n: n \text{ natural}\} \cup \{\lambda\}$. By inspection we derive the induction:

**Basis**: $\lambda$ and $01$ in $L(G)$

**Induction**: If $0y$ is in $L(G)$, then $01y$ is in $L(G)$

An example of string generation with induction:

Since $01$ in $L(G)$, by making $01 = 0y$ we get $y = 1$. Thus, $01y = 011$ is also in $L(G)$
Inductive string generation

Assume that we control string generation with the string’s length. In this case, the inductive procedure will be of type Nat \rightarrow Strings. And the procedure will be:

\textbf{Generate}(n):

- If \( n = 0 \) return \( \lambda \)
- Else if \( n = 1 \) return “no strings generated”
- Else \( \text{string} = 01 \)
  - While \( |\text{string}| < n \)
    - \( \text{string} \leftarrow \text{cat}(\text{string}, 1) \)
- Return \( \text{string} \)
Finding the induction in a grammar

Here is the method:

1. Put all strings in the grammar that can be generated without recursion in the basis of the inductive constructor

2. For each recursive derivation establish a “pattern of generation”, and “translate it into an induction rule” (this will be obvious in each case). If more than one recursive derivation is identified, link them with or’s
Application to the previous example

The non-recursive derivations in the previous grammar are:

\[ s \rightarrow \lambda \text{ and } s \rightarrow 0v \rightarrow 01 \]

Therefore, \( \lambda \text{ and } 01 \) in \( L(G) \) are basic conditions.

Now, the only recursive derivation is of the form of:

\[ s \rightarrow 0v \rightarrow 01v \rightarrow 011v \rightarrow 0111. \]

Here, the “pattern” is represented by

\[ 01v \rightarrow 011v \] which indicates that

\[ 0y \text{ in } L(G) \text{ implies } 01y \text{ in } L(G) \]
String parsing with grammars

**Example:** Let’s consider the grammar

\[ G = (\{s\}, \{0, 1\}, \{s \rightarrow 0s1 | \lambda\}, s) \]

and the string

000111. Clearly, 000111 is derived with \( G \) through the sequence of productions:

- \( s \rightarrow 0s1 \rightarrow 00s11 \rightarrow 000s111 \)
- \( \rightarrow 000111 \)

The parse tree is:
Meaning

As stated before, strings convey information. This is, there is a meaning attached to each string. For example, the string 3+4 means “3 plus 4”

Some strings may have more than one possible meaning. For example, the string 3-4-2 could mean “tree minus four minus two” of “three minus the result of four minus two”

**Definition**: The meaning of a string generated by a grammar is defined to be the parse tree of the string with respect to this grammar.
Parsing trees and derivations

The parsing tree may depend on the order in which the variables are replaced in a derivation, this is, sometimes, different orders in the replacement of the variables give rise to different parse trees for the same string. If a policy for the replacement of the variables is assumed (let’s say: all derivations are leftmost derivations), no difference in the parsing trees may arise as a result of different orders in the derivation.
Leftmost derivations

**Definition:** A *leftmost derivation* is a derivation in which each production is the result of the replacement of the leftmost variable in the string.

**Example:** Let $G=\langle \{s, u\}, \{0, 1\}, \{s \rightarrow s0u|\lambda, u \rightarrow u1s|\lambda\}, s \rangle$. Then, $s \rightarrow s0u \rightarrow 0u \rightarrow 0u1s \rightarrow 01s \rightarrow 01$ is a leftmost derivation while $s \rightarrow s0u \rightarrow s0u1s \rightarrow 0u1s \rightarrow 01s \rightarrow 01$ is not!!!
**Ambiguity**

**Definition:** A grammar is said to be **ambiguous** if its language contains a string that has two different parse trees under leftmost derivation.

**Example:**

\[ G = (\{s\}, \{2, 3, 4, -\}, \{s \rightarrow s-s | 2 | 3 | 4\}, s), \text{ string: 3-4-2} \]

**First parse tree:**

```
  S
 /\ \
/   \ /
S   - S
 |   /
3   S
```

**Second parse tree:**

```
  S
 /\ \
/   \ /
S   - S
 |   /
2   S
```

**Example:**

\[ G = (\{s\}, \{2, 3, 4, -\}, \{s \rightarrow s-s | 2 | 3 | 4\}, s), \text{ string: 3-4-2} \]

**First parse tree:**

```
  S
 /\ \
/   \ /
S   - S
 |   /
3   S
```

**Second parse tree:**

```
  S
 /\ \
/   \ /
S   - S
 |   /
2   S
```
Disambiguation

It is sometimes possible to manipulate the rules in a grammar to eliminate ambiguity. This is what is called disambiguation of the grammar. Following is a disambiguation of the previous grammar:

\[
G' = (\{s, u\}, \{2, 3, 4, -\},
    \{s \rightarrow s-u \mid u, u \rightarrow 2 \mid 3 \mid 4\}, s)
\]

Parsing tree for 3-4-2:
Summary

• Informal definition of grammar
• Rules and productions
• Variables and terminals
• Formal definition of grammar
• String derivations
• The language of a grammar
• Recursive productions and recursive grammars
• Two theoretical principles
• Method for finding a grammar’s inductive constructor
• Parsing, parsing tree and leftmost derivations
• Meaning and ambiguity
Exercises

1. Given the grammar

\[ G = (\{s, u\}, \{., 0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}, \{s \rightarrow u | us, u \rightarrow . | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9\}, s) \]

a) Derive the strings 9012 and 0.92 using leftmost derivations

b) Describe the language of this grammar as a set

c) Modify the grammar’s rules so to produce only nonnegative finite decimal numbers

d) Modify the grammar in c) to produce all (negative and nonnegative) finite decimal numbers

e) Find an inductive form for the grammar in d)

f) Find the parse trees for 9.001 and -12.34
Exercises

2. \( G = (\{s\}, \{ (, ) \}, \{s \rightarrow (s)s\}, s) \)
   a) Describe the language of this grammar as a set
   b) Find an inductive form for the grammar
   c) Find the parse trees for \(((())\) and \((())\)

3. For each of the following languages:
   i. \( L = \{ba^n b : n \text{ natural}\} \)
   ii. \( L = \{w : w \text{ is a well-defined Boolean formula with 3 variables}\} \)
   iii. \( L = \{0, 1\}^* \)
   a) Find a grammar
   b) Find an inductive form based on this grammar
4. Some Boolean formulas are ambiguous. One example is *A and B or C and A*. Construct:
   a) A grammar that reflects this ambiguity for Boolean formulas with three variables and disjunctions and/or conjunctions (This grammar must exhibit two different parse trees for the same string)
   b) Disambiguate the grammar in a) by modifying its set of rules