TD1 : Grammars and Syntactic Specifications

Thibaut Benjamin[1] and Burkhart Wolff[2][0000−0002−9648−7663]

[1] LMF, Université Paris-Saclay, France
thibaut.benjamin@universite-paris-saclay.fr
[2] LMF, Université Paris-Saclay, France
wolff@lmf.cnrs.fr

Abstract. This TD gives an introduction to the concepts of grammars and Chomski Hierarchies. A bridge to programming languages and real-world specifications of the syntax of real languages is provided.
The final exercise serves as basis for an introduction to functional programming presented later in this course.
The accompanying material can be found on the the following site: [1]

Keywords: Grammars, Parsers, Railroad Diagrams, Programming in SML

1 TD 1: Classic Formal Language Theory

1.1 Exo 1

We consider the grammars :

1. $G1 = \{\ S ::= 0,\ S ::= 1\ \}$
2. $G2 = \{\ S ::= a\ S,\ S ::= T,\ T ::= b\ T,\ T ::= \varepsilon\ \}$
3. $G2' = \{\ S ::= a\ S,\ S ::= T,\ T ::= T\ b,\ T ::= \varepsilon\ \}$
4. $G3 = \{\ S ::= (\ S\ )\ S,\ S ::= [\ S\ ]\ S,\ S ::= \varepsilon\ \}$
5. $G4 = \{\ S ::= a\ S\ b,\ S ::= \varepsilon\ \}$

Tasks:

- Classify these grammars: Which Chomski type do the grammars above have? Are they left- or right regular ?
- Conceive a grammar that produces the language $\{\ a^n b^n c^n \mid n > 0\ \}$. Which Chomski type will it have ?
- Conceive a grammar that produces the language $\{\ a^n b^m c^n d^m \mid n,m > 0\ \}$. Which Chomski type will it have ?
- Let *G1* et *G2* be two grammars of the same Chomski type. (Make an educated guess.)
  - Can you decide $L(G1) = L(G2)$?
  - Can you decide $L(G1) \cup L(G2)$?
  - Can you decide $(G1) \cap L(G2)$?
  - Can you decide $L(G1) = L(G2)$?

1.2 Exo 2

Let's return to the example used in the first lesson to illustrate the lexical aspects:

```
function f(x : integer) return integer is
z : integer := -5;
-- on suppose l'existence de la variable y ci-dessous
begin x := y + z * 2;
      x := -x-2;
      if (x = y or z >= y) then return x-123;
end;
```

1. Provide a grammar for describing a function header in this mini-language; we will ignore the function body for simplicity. A function can take any number of parameters, including none, enclosed in parentheses. A list of parameters of the same type can be grouped by separating them with commas. The list is then followed by a colon and the type name. Multiple such sequences of parameters can be separated by semicolons.
2. Give Examples of declarations following your grammar.
3. Using your grammar, provide the syntax tree for the last example.

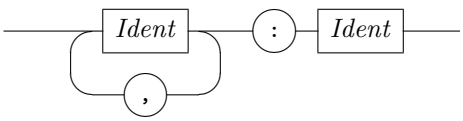
1.3 Exo 3

A particular form of diagrams, so-called *railroad diagrams*, are commonly used technical documentations to specify the concrete syntax of programming languages and textual exchange formats like jason, bibtex or XML.

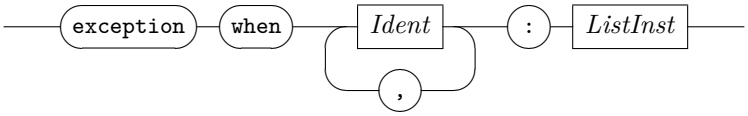
Provide an unambiguous context-free grammar for the language specified by the below:

– *Block*:

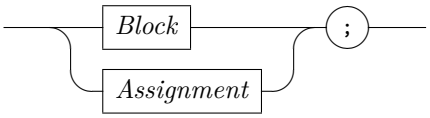
– *Decls*:



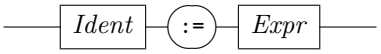
– *Handler*:



– *ListInst*:

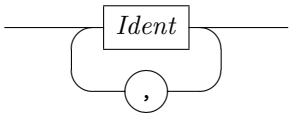


– *Assignment*:



Non-terminals are the labels of the boxes, terminals are those of the arcs. The non-terminal 'Expr' represents an expression; it will not be detailed here. The root of the grammar is 'Block'.

QUESTION: Is there a general way to to translate optional parts, repetitions or repetitions with a separator symbol like ',' in



in grammars ?

1.4 Exo 4

We consider the language of propositional logical formulas constructed using atomic propositions, hereafter symbolized by the lexical unit P, the binary operators $\wedge$ (logical "and"), $\vee$ (logical "or"), $\Rightarrow$ (implication), $\Leftrightarrow$ (equivalence), and the unary operator $\neg$ (negation). Such a language can be described by the following grammar:

$$\mathrm{F} ::= \mathrm{P} \mid (\ \mathrm{F}\ ) \mid \mathrm{F} \wedge \mathrm{F} \mid \mathrm{F} \vee \mathrm{F} \mid \mathrm{F} \Rightarrow \mathrm{F} \mid \mathrm{F} \Leftrightarrow \mathrm{F} \mid \neg\ \mathrm{F}$$

1. Give the syntax tree for the formula $(\mathrm{p} \wedge \mathrm{q}) \vee \neg \mathrm{r}$, where p, q, and r are instances of P.
2. Show that the grammar is ambiguous. Give an unambiguous grammar for the same language, but imposing the following precedence on the operators (a "$<$" indicates that the left-hand symbol has lower precedence than the right-hand one, an "$=$" indicates that they have the same precedence). Assume the binary operators are left-associative: $\langle\Leftrightarrow\rangle < \langle\Rightarrow\rangle < \langle\wedge\rangle = \langle\vee\rangle < \langle\neg\rangle$
3. Give the syntax tree associated with the word $\mathrm{p} \wedge \mathrm{q} \vee \mathrm{g} \Rightarrow \neg\ \mathrm{r} \wedge \mathrm{p}$.
4. Give an unambiguous grammar that respects the precedence rules above but prohibits unparenthesized mixtures of $\wedge$ and $\vee$, as well as any associativity with $\Rightarrow$ and $\Leftrightarrow$. The operators $\wedge$ and $\vee$ remain left-associative.

The following words must be invalid:

$$p \wedge q \vee r \wedge g,\ p \Rightarrow q \Rightarrow r,\ p \Leftrightarrow q \Leftrightarrow r.$$

The following words are valid: $p \wedge (q \vee r) \wedge g$, $p \vee q \vee r$:

$$p \Rightarrow (q \Rightarrow r),\ (p \Leftrightarrow q) \Leftrightarrow r$$

1.5 Exo 5

Provide an unambiguous grammar for a restricted version of C language expressions. Consider the following operators, listed in order of increasing precedence:

1. = (binary, right-associative)
2. +, - (binary, left-associative)
3. *, / (binary, left-associative)
4. * (unary, dereferencing pointers)

The grammar must allow manipulation of identifiers, and constants.

1.6 Exo Extra

Install the Isabelle/System version 2025-2. We will use it for some small TD's and all TPs.

Isabelle and in particular Isabelle/HOL can be got from `https://isabelle.in.tum.de/installation.html` Isabelle is available for MacOS, Linux and Windows. (We recommend MacOS or Linux).

Follow closely the hints concerning the security alerts you get on your platform and how to get around them.

Try to start Isabelle on a command shell; for example by

- `Isabelle2025-2/bin/isabelle jedit` (Linux), or
- `Isabelle2025-2.app/bin/isabelle jedit`(MacOS).)

end

References

1. Wolff, B.: Teaching Website: PolyTech Compiler Course. `https://usr.lmf.cnrs.fr/~wolff/teach-material/2025-2026/ET4-Compil/index.html` (2025), [Online; accessed 8-Dec-2025]