# Lab 7

**Statement**: **Assignment for a team of 2 students!**

 Can also be done by 1 student for Laboratory Bonus

For the mini DSL, implement parsing (syntactical analysis)  using LL(1) or LR(0) algorithms.

- **Requirement 1**: implement and test

  Input: grammar from seminar, sequence

  Output: string of productions

- **Requirement 2**: test

  Input: grammar for miniDSL, PIF

  Output: The representation of the parsing tree (output) can be:

  - 2.a. productions string (max grade = 8.5)
  - 2.b. derivations string (max grade = 9)
  - 2.c. table (using father and sibling relation) (max grade = 10)

**Delivery time**: 2 weeks (Week 10)

**Deliverables:**

- Documentation
  - grammar used for Req.1
  - output - string of productions for
    - a sequence accepted by the grammar
    - a sequence not accepted by the grammar

    - grammar corresponding to mini DSL for Req.2
        - use the 2 input “programs” from the previous lab (YACC/ANTLR parser) - syntactically correct one & syntactically incorrect one
        - PIF for each of the 2 input “programs” used
        - output for each based on the chosen representation of the parsing tree
    - (for 2.c.) - drawing of the parse tree for more clarity w.r.t the father-sibling table
- Source code

---

 Fun fact:

- yacc (one of the alternatives from Lab 6) uses LALR(1) parsing
- ANTLR: ALL(*) - Adaptive LL(*) aka LL(k)++ - source

---

## What should be done by Week 9 (next lab)

☐ establish teams of 2 (if you choose to work in teams)

☐ think of a standardized format for grammar input : reading from file for example - this can (and should) be used for the grammar from the seminar and the one corresponding to your mini DSL

☐ “Convert” your mini DSL to a grammar

Quick guidelines: G(N, Epsilon, S, P)

 You will need the BNF and the Lexer/Scanner from previous labs

- N(onterminals): LHS of BNF rules become non-terminal symbols (tokens excluded)
    - add nonterminals needed for optional/repeated parts

- Epsilon (Terminals): tokens from lexer - reserved keywords, operators, separators, constants, identifiers (last 2 can be noted in a generic manner as `id` and `ct` for instance - we just need to know they are tokens)
    - Terminals should match **exactly** the token names used in your scanner/PIF (because you will be using the PIF as input for the parsing algorithm)
- Starting Symbol (S): `program` (or however it's called in your mini DSL)
    - entry point of the language grammar
    - the nonterminal representing the entire program
- Productions (P): BNF rules adapted to production format
    - `Nonterminal → sequence of terminals and nonterminals`

☐ Be able to access said grammar and use its components for the selected parsing algorithm

---

# For Week 8 - LL(1) quick summary for Req.1

📒 Use grammar from seminar to ensure the correctness of your algorithms. Pro tip: you can also refer to the examples from the book (see end of document for more details)

LL(1)

- prediction of length 1
- Left - sequence read from left to right
- Left - uses leftmost derivations

📒 Slides from Lecture 7 are your best friends here 😀

Algorithm LL(1) parsing

INPUT:

- LL(1) table with NO conflicts;
- G –grammar (productions)

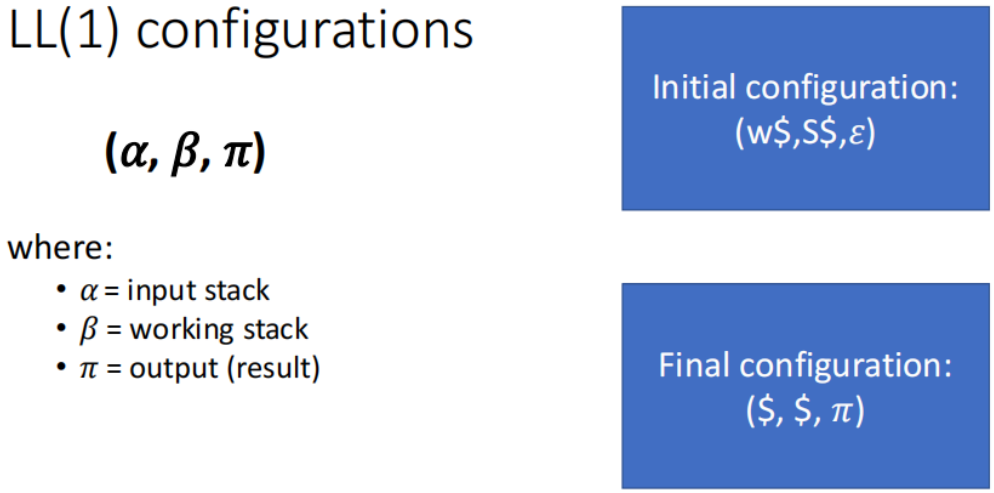
**Input sequence w = a**1..**a**2..**a**n

OUTPUT:

- sequence accepted or not?
- If yes then string of productions

## Steps

1. Construct FIRST and FOLLOW - see Lecture 7 slides for both algorithms
2. Construct LL(1) parse table - Lecture 7 slides
3. Analyse sequence based on moves between configurations




- M is the table from previous step

## Algorithm LL(1) parsing (cont)

```
alpha := w$;beta := S$;pi := ε; config =(alpha,beta, pi)
go := true;
while go do
      if M(head(beta),head(alpha))=(b,i) then
                  ActionPush(config)
      else
            if M(head(beta),head(alpha))=pop then
                  ActionPop(config)
            else
                  if M(head(beta),head(alpha))=acc then
                        go:=false; s:="acc";
                  else  go:=false; s:="err";
                  end if
            end if
      end if
end while
```

```
if s="'acc"' then
      write("Sequence accepted");
      write(pi)
   else
      write(" Sequence not accepted,
            syntax error at", head(alpha))
```

S.Motogna - FL&CD

# LR(0) coming next week (week 9)

# Req.2

- think of how to use your grammar as input

### Week 8 - for LL(1)

## Remark

A grammar is LL(1) if the LL(1) parse table does NOT contain conflicts – there exists at most one value in each cell of the table M(A,a)

S.Motogna - FL&CD

- if the grammar from your mini DSL is not LL(1) and you want to use LL(1) - transform it

## Remarks

1) LL(1) parser provides location of the error

2) Grammars can be transformed to be LL(1)

   example:

   I -> if C then S | if C then S else S  // is not LL(1)

   I -> if C then S T

   T -> ε | else S  // is LL(1)

S.Motogna - FL&CD

---

# Possible output representations

... exemplified on a simple grammar...

```
G = ( {S, A, B, C}, {a, b, c}, P, S}
where P :
1. S → A B
2. A → a A | a
3. B → b C
4. C → c
```

... and the input sequence `a a b c`

This will match:

```
S → A B
A → a A → a a
B → b C
C → c
```

## Production String

```
S → A B
A → a A
A → a
B → b C
C → c
```

## Derivation string

- show every step of the derivation
  - leftmost derivation for LL(1)

```
S
⇒ A B
⇒ a A B
⇒ a a B
```

```
⇒ a a b C
⇒ a a b c
```

## Parsing tree table using father sibling relation

The parse tree looks like this

```
          S (1)
        /      \
     (2) A       B (6)
      /  \    /    \
   (3) a (4) A   b (7)   C (8)
         |     |
      (5) a     c (9)
```

... and the father sibling table

| Index | Symbol | Father | Sibling |
|---|---|---|---|
| 1 | S | 0 | 0 |
| 2 | A | 1 | 6 |
| 3 | a | 2 | 4 |
| 4 | A | 2 | 0 |
| 5 | a | 4 | 0 |
| 6 | B | 1 | 0 |
| 7 | b | 6 | 8 |
| 8 | C | 6 | 0 |
| 9 | c | 8 | 0 |

---

Check out *the book* : https://drive.google.com/file/d/1istyZgczVIDQf5sjGqBkptaKGyE92BDB/view?usp=drive_link

- for LL(1) see Page 56 🙂 - example of a grammar and steps