|  |
| --- |
|  |
| LL(1) VALIDATOR |
| User manual |

|  |
| --- |
| Matteo Locatelli, Andrea Cattaneo |

Sommario

[Preface 1](#_Toc30360453)

[Grammar structure 1](#_Toc30360454)

[Errors in the grammar 1](#_Toc30360455)

[Lexical errors 1](#_Toc30360456)

[Syntactic errors 1](#_Toc30360457)

[Semantic errors 1](#_Toc30360458)

[Grammar examples 2](#_Toc30360459)

[Correct grammar 2](#_Toc30360460)

[Wrong grammars 2](#_Toc30360461)

[How to use the validator 2](#_Toc30360462)

[Use the package 3](#_Toc30360463)

[Output example 3](#_Toc30360464)

[The web interface – LL1 validator demo 5](#_Toc30360465)

[Where to find it 5](#_Toc30360466)

[User manual 5](#_Toc30360467)

[The UI 5](#_Toc30360468)

[Error Management 5](#_Toc30360469)

# Preface

This document shows the user manual of LL1 validator, a powerful tool able to analyze and validate any LL(1) grammar, and the user manual of LL1 validator demo, a real use case web application that use the LL1 validator package.

# Grammar structure

In order to recognize correctly the grammar, the grammar have to be the following structure:

* A start symbol for the grammar. It’s possible to declare it with #start\_symbol keyword (an example will be show later). If it’s not declared, the axiom will be the first non-terminal in the grammar.
* Each rule have to be the following format:

NT -> (NT | T)\* ;

|  |  |
| --- | --- |
| KEYWORD | CHARACTERS |
| NT | [A … Z | a … z| +| - | . ]+ |
| T | [A … Z | a … z| +| - | . ]+ |

Table 1: List of accepted characters

Firstly, all the words are set as Terminal words, but they become Non-Terminal if they occur at the beginning of a rule.

* It is possible to comment the rules by using the characters // for a single line comment and /\* … \*/ for a multiline comment.

Note: The definition of the rule structure has been made in order to respect the Backus–Naur Form (BNF) notation.

## Errors in the grammar

### Lexical errors

Using one or more character not included in the accepted characters is a lexical error.

### Syntactic errors

Any rule that doesn’t follow the given structure is a syntactic error. No alternative syntax is permitted.

### Semantic errors

There are two kinds of semantic error:

* If the given grammar contains an infinite loop when calculating the nullables Non-Terminals, we have a fatal error that impedes to complete the analysis of the grammar
* If there are some duplicated rules, we have a warning: it’s a non-blocking error.

## Grammar examples

### Correct grammar

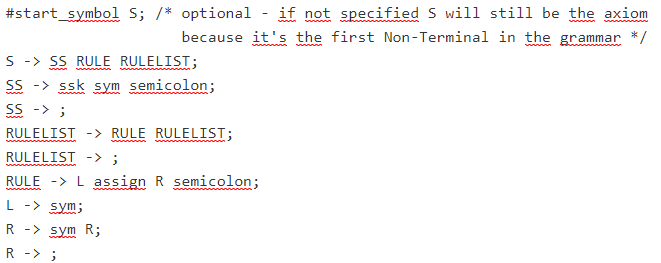
The following example shows an example of a correct grammar

Figure 1: Example of a correct grammar

### Wrong grammars

The following example contains a lexical error: ò is a character not included in *Table 1*, and it generates an error in the lexical analysis.



Figure 2: Example of grammar with lexical error

The next example shows a syntactic error: missing ; at the end of the 2nd rule



Figure 3: Example of grammar with syntactic error

In the last example there is a semantic error: the grammar contains an infinite loop: it’s not possible to calculate if the Non-Terminals are nullable.



Figure 4: Example of grammar with semantic error

# How to use the validator

The validator is an NPM package, so the only requirement is a javascript application that import the package “ll1-validator”.

A real use case is visible from another project, “LL1 validator demo”, that we’ll be show later.

## Use the package

First, you have to import into you javascript file the npm package using the following code:

import { validate } from "ll1-validator";

After that, you can get all the info about that grammar by calling the function validate(grammarString), that returns the following variables:

1. grammar: the grammar extracted from the string text in input;
2. startSymbol: the symbol that the parser has recognized as the axiom;
3. rulesNumber: the number of rules of the given grammar;
4. nullableNonTerminals: an array that contains a Boolean for each Non-Terminal that says if it’s nullable;
5. terminals: an array of words recognized as terminals
6. nonTerminals: an array of words recognized as Non-Terminals;
7. warnings: all the duplicated rules found;
8. firstSets: a dictionary that contains all the iterations in order to obtain the first sets for each Non-Terminal;
9. followSets: a dictionary that contains all the iterations in order to obtain the follow sets for each Non-Terminal;
10. firstSetsDependencies: a dictionary that contains the first set dependencies for each Non-Terminal;
11. followSetsDependencies: a dictionary that contains the follow set dependencies for each Non-Terminal;
12. lookAheads: a dictionary that contains all the lookaheads;
13. isLL1: Boolean value that represent if the given grammar is LL1 or not
14. lookAheadsConflicts: a dictionary that contains, for each Non-Terminal, the list of conflictual terminal words.

## Output example

We call validate(grammarString), where grammarString=“S -> T; T -> a T a; T -> b T b; T -> c T c; T -> q;”. The output result is:

grammar: {

            'S': [[{ type: parser.NONTERMINAL, value: 'T' }]],

            'T': [[ { type: parser.TERMINAL, value: 'a' },

                    { type: parser.NONTERMINAL, value: 'T' },

                    { type: parser.TERMINAL, value: 'a' }

                ],[ { type: parser.TERMINAL, value: 'b' },

                    { type: parser.NONTERMINAL, value: 'T' },

                    { type: parser.TERMINAL, value: 'b' }

                ],[ { type: parser.TERMINAL, value: 'c' },

                    { type: parser.NONTERMINAL, value: 'T' },

                    { type: parser.TERMINAL, value: 'c' }

                ],[{ type: parser.TERMINAL, value: 'q' }]],

        },

startSymbol: 'S',

rulesNumber: 5,

nullableNonTerminals: { S: false, T: false},

terminals: ['a', 'b', 'c', 'q'],

nonTerminals: ['S', 'T'],

warnings: [],

firstSets: {'S': [[],['a','b','c','q'],['a','b','c','q']],

'T': [[['a'], ['b'], ['c']], [['a'], ['b'], ['c']], [['a'], ['b'], ['c']]]},

followSets: {'S': [['↙'],['↙'],['↙']],

'T': [['a', 'b', 'c'],['a', 'b', 'c', '↙'],['a', 'b', 'c', '↙']

        ]},

firstSetsDependencies: {'S': ['T'], 'T':[]},

followSetsDependencies: {'S': [], 'T':['S']},

lookAheads: {'S': [['a', 'b', 'c', 'q']],'T': [['a'],['b'],['c'],['q']]},

isLL1: true,

lookAheadsCoflicts: {}

# The web interface – LL1 validator demo

## Where to find it

You can find the LL1 validator web application at the following link: [LL1 validator web app](https://ll1-validator.netlify.com/)

The application is hosted on Netlify, a web host for small websites, and its repository can be found [here](https://github.com/imcatta/ll1-validator-demo).

## User manual

### The UI

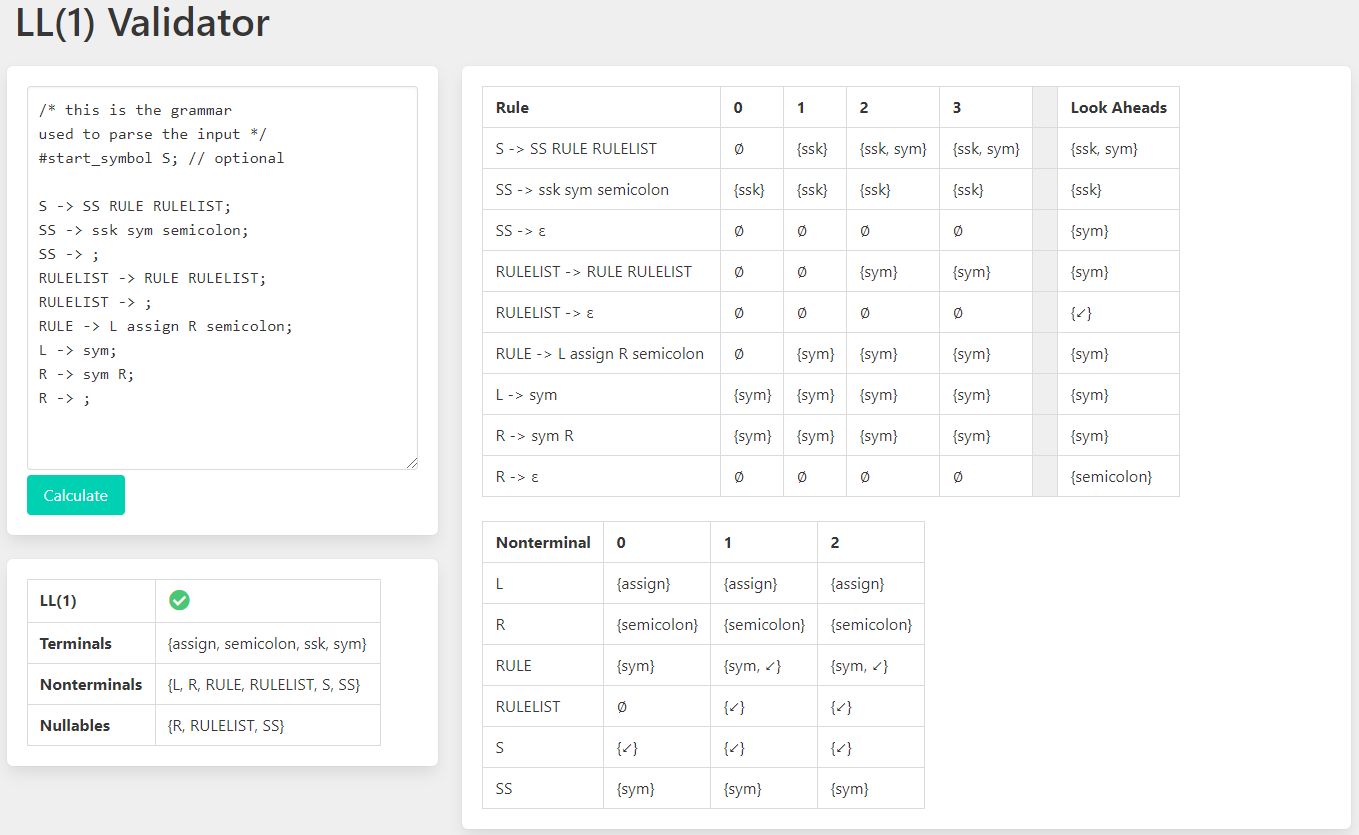


Figure 5: Interface of the web app

The UI is simple: on the top left we have a text field where it’s possible to write a custom grammar. Under that we have the *Calculate* button, which starts the LL1 validation. Under this panel we have a small panel with the vital infos: if the grammar is LL(1), the Terminal words, the Non-Terminal words and the nullable Non-Terminals. On the right we have the representation of the parsed grammar: each rule have the iterations that produces the firsts set, and the lookaheads. On the bottom right we have the iterations that produces the follow sets.

### Error Management

### 