UNIT 5: TOP-DOWN PARSING TECHNIQUES

We want to construct a compiler for a language of definition and application of sequential machines. In the language you can define as many machines and process as many strings as you want. The language format is as follows:

- To declare a sequential machine, the MS instruction is used:

```
MS name_of_the_sequential_machine
{
inputs { symbol₁, symbol₂,…,symbolₙ}
outputs { symbol₁, symbol₂,…,symbolₖ}
states { state₁, state₂, …}
transitions {
(state₁, symbol_input₁, state₂, symbol_output₁)
(state₁, symbol_input₁, state₂, symbol_output₁)
…
(state₁, symbol_input₁, state₂, symbol_output₁)
}
}
```

- For the sequential machine to process a string, it is used:

```
process ( name_of_automaton, string, initial_state)
```

The name of the sequential machine is a string of alphabetic characters. The statement of the set of states, the set of input and output symbols, and the set of transitions can be in any order, but they must always be included in every statement. The set of states, transitions, input and output symbols must have at least one value. An example of a sequential machine definition in this language would be:

```
MS Afirst
{
outputs {1,0}
states {a, c}
inputs {L,M,N}
transitions { (a,L,a,1)
(a,M,a,1)
(a,N,c,0)
(c,L,a,0)
(c,M,a,0)
(c,N,c,1)
}
}
process (Afirst, LLM, a)
process (Afirst, LLM, c)
```

To use the process function, the sequential machine used must be previously declared. The function displays, for the previous example, the following information:

```
MS: Afirst     Input: LLM  Output: 111
MS: Afirst     Input: LLM  Output: 011
```

It is required:

1. Define the grammar G that would generate valid sentences of this programming language.
2. Can the grammar G of the first exercise be used to perform an LL(1) analysis? If not, modify it so that it can. Generate the LL(1) table. Does this language require semantic verifications?

David Griol Barres, Antonio Berlanga de Jesús, Jesús García Herrero, Juan Manuel Alonso Weber