Lab session 12: Natural Language Processing

Pierre-Alexandre Murena

July 5, 2024

1 Context-free grammars in Prolog

The goal of this exercise is to implement a simple grammar for English within Prolog. For a reminder on how to use Prolog, please check lab 8.

1.1 Attempt 1: Using standard Prolog syntax

Before we start writing the grammar, we need to introduce a new element of Prolog. Indeed, in this application, we will require to use lists. However, it is not necessary to have a perfect understanding of list operations, just of the `append` predicate. This predicate works as follows: `append(X, Y, Z).` is true when Z is the concatenation of X and Y. This predicate has arity 3, which means that it is possible to concatenate only 2 lists in native Prolog.

Question 1. Using the arity 3 predicate, implement an arity 4 predicate `append(A, B, C, ABC)` which tests whether list ABC is the concatenation of A, B and C.

We can write a simple context-free grammar in Prolog using the following conventions:

- An individual word is written as a constant (starting with lower case).
- Phrases are lists. For instance, the phrase *it rains* would write `[it, rains]`
- Non-terminals (= types of phrases) are written as unary predicate: for instance `np(hello)` tests whether *hello* is a valid noun phrase.
- The types of the words are declared with assertions on the list containing the word as unique element: for instance `v([walks])` asserts that *walks* is a verb.

A simple fragment of grammar is implemented in `attempt1.pl`: it states that:

- a sentence is made up of a noun phrase and a verb phrase;
- a verb phrase is made up of a verb;
- a noun phrase is made up of a determiner and a noun.

The program also contains the types of a few words.

Question 2. Using this program, check that *the man walks* is a valid sentence in this language. How can you generate all valid sentences?

When generating these sentences, you can realize that many valid English sentences are not produced. This is because our grammar is too restricted for now.

Question 3. Extend this grammar by adding the first 5 rules defining noun phrases introduced in slide 47. You will need to define adjective phrases, which for now are made up only of a determiner and an adjective.

Question 4. Implement a relative clause made up of a WH word and of a verb phrase. Extend your definition of noun phrases to phrases of the form NP RelClause. What can you observe when you try to generate all noun phrases?

In the current state, the grammar only allows to use sentences of the form Subject Verb, but does not allow for Subject Verb Complement (for instance *The man sees the dog*).

Question 5. Add the verb sees to the vocabulary assertion, and add a rule to define verb phrases as verb and noun phrase.

You can see several issues with this implementation:

1. If you try to generate all sentences, you will get trapped into infinite recursive loops, generating sentences like *She sees the woman who sees the hungry man who sees the dog* etc. This problem is inherent to the grammar we implemented: nothing prevents from nesting too many relative clauses.

2. More problematic: depending on the order in which you organized the rules, the inference engine might get trapped into an infinite recursion when trying to verify if a sentence is valid.

3. Some incorrect sentences are produced, of the form *she vanishes the dog*. This is due to the fact that the grammar is not make the difference between transitive and intransitive verbs.

1.2 Attempt 2: Using Prolog syntax for CFGs

Prolog allows to directly implement CFGs using the arrow notation seen during the lecture. For instance, the rule stating that a noun phrase can be made up of a determiner and a noun, can be written:

```
np --> determiner, n.
```

In file `attempt2.pl`, you will find an implementation of a simple rule.

Question 6. Reimplement the rules from previous task (except for the relative clause), but using this syntax.

The validity of a phrase can be checked using the `phrase` predicate:

```
phrase(np, [this, dog]).
```

will for instance test whether the phrase *this dog* is a noun phrase.

Question 7. Add transitive and intransitive verb categories, and modify the verb phrases accordingly.

Imagination is the limit! You can try implementing as many rules as you wish and test the produced sentences.

2 Finite-state transducer

In file `fst.py`, you will find a simple implementation of a finite-state transducer. In this implementation, node 0 is the initial node, the terminal states are stored in the dictionary `terminal_state` and the transitions are stored in the dictionary `transitions`. Each transition is a quadruple (source state, target state, input character, output character). For instance, transition (0, 1, 'a', 'b') means that we transition from state 0 to state 1 by writing 'a' in the input tape and 'b' in the output tape.

Question 8. Initialize this transducer to represent the following morphological transformations (one initialization per pair):

- {(cats, cat +N +Pl)}
- {(cats, cat +N +Pl), (birds, bird +N +Pl) }
- {(mice, mouse +N +Pl) }
- {(suden, susi +N +Sg +Gen), (sudessa, susi +N +Sg +Ine) }