Symbolic Programming - Chapter 7 - Definite Clause Grammars

Varjak Wolfe

November 26, 2021

These notes follow the online coursebook Learn Prolog Now.

Context-Free Grammars DCGs are a special notation for defining grammars.

CFGs are a finite collection of rules which tell us that certain sentences are correct and what their structure is.

A simple context free grammar for a small fragment of English:

s $->$ np vp
np $->$ det n
vp $->$ v np
vp $->$ v
det $->$ a
det $->$ the
n $->$ woman
n $->$ man
v $->$ shoots

The arrow -¿ means its a rule. The symbols s, np, vp, det, n, v are nonterminals. In this case, each of them has a meaning from linguistics: s = sentence, np = noun phrase, vp = verb phrase and det = determiner. i.e. each symbol is shorthand for a grammatical category.

n = noun, v = verb

Finally, we have symbols a, the, woman, man, shoots. These are terminal symbols or words or lexical items.

This grammar contains 9 context free rules. A CFR consists of a single nonterminal, followed by an arrow and a finite sequence made of terminal and/or nonterminals.

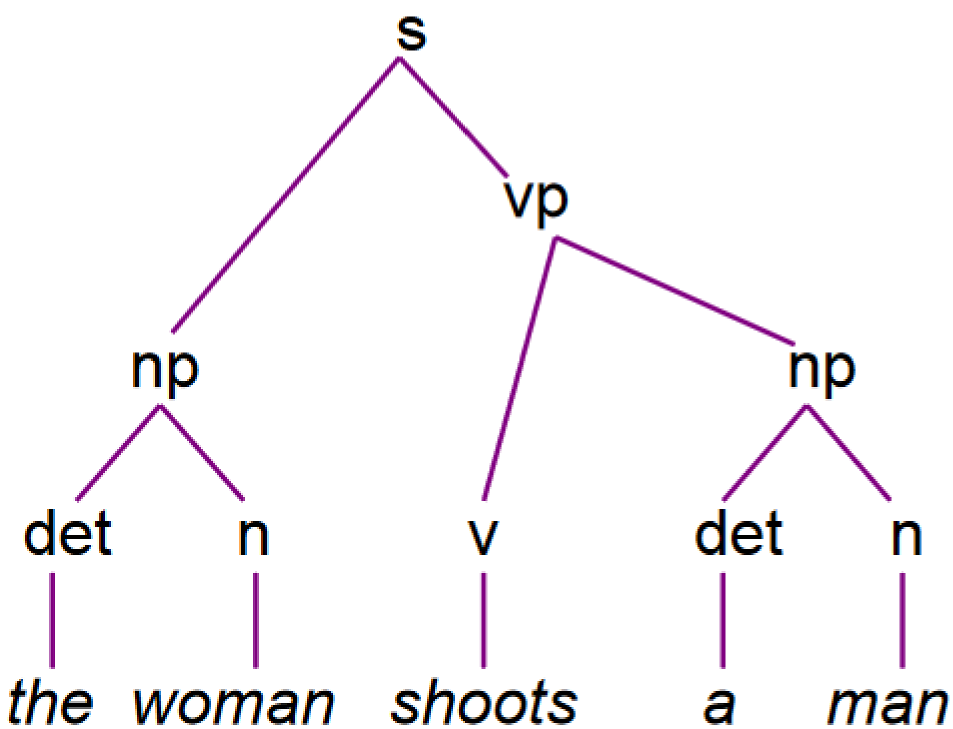
Rule 1 tells us a sentence consists of a noun phrase and a verb phrase; and so on.

Is the string ”a woman shoots a man” grammatically correct in our CFG?

s $->$ np vp
np $->$ det n
det $->$ a or the
vp $->$ v np or v

A woman shoots a man = det n v det n

s = det n v det n
s = np v np
s = np vp
Therefore it is grammatically correct for our CFG
Parse Tree
The tree that can be used to answer the above question:



This is a parse tree; one which represents the syntactic structure of a string. They provide information about the string and its structure.

If we are given a string of words and a grammar, and it turns out we can build a parse tree then we can say that the string is grammatical for this particular grammar.

The language generated by a grammar consists of all the strings that the grammar classifies as grammatical.

Recogniser

A context free recogniser is a program which correctly tells us whether or not a string belongs to the language generated by a CFG. Basically, it classifies strings as either grammatical or ungrammatical.

Parser

A context free parser correctly decides whether a string belongs to the language generated by a context free grammar and it also tells us the structure.

A recogniser says yes or no but a parser also provides a parse tree.

CFG Recognition in Prolog

Using a list to represent a sequence of tokens: [a, woman, shoots, a, man]

The rule s − > np vp can be thought as concatenating an np-list with a vp-list resulting in an s-list. We can concatenate using append/3. See recogniser.pl in PrologDCG on GitHub.

The problems with this recogniser:

- It doesnt use the input string to guide the search
- Goals such as np(A) and vp(B) are called with uninstantiated variables

A more efficient implementation: difference lists

Dffierence Lists

[a,b,c]-[] is the list [a.b.c]

[a,b,c]-[d] is the list [a.b.c]

[a,b,c—T]-T is the list [a.b.c]

X-X is the empty list [ ]

See Recogniser.pl for an implementation

The recogniser using difference lists is a lot more efficient than using append/3.

Definite Clause Grammars

DCGs have the simplicity of append but the efficiency of difference lists.

They are a nice notation for writing grammars that hides the underlying difference list variables.

s −− > np, vp.
np −− > det, n.
vp −− > v, np.
vp −− > v.
det −− > [the].
det −− > [a].
n −− > [man].
n −− > [woman].
v −− > [shoots].

A DCG rule such as s −− > np, vp. is really a syntactic variant of:

s(A,B):- np(A,C), vp(C.B).

Another DCG example:

s −− > s, conj, s.
s –¿ np, vp.
np –¿ det, n.
vp –¿ v, np.
vp –¿ v.
det –¿ [the].
det –¿ [a].
n –¿ [man].
n –¿ [woman].

v –¿ [shoots].
conj –¿ [and].
conj –¿ [or].
conj –¿ [but].
We have added some recursive rules in this case though.

DCG Without Left-Recursive Rules

s −− > simple_s, conj, s.
s −− > simple_s.
simple_s −− > np, vp.
np −− > det, n.
vp −− > v, np.
vp −− > v.
det −− > [the].
det −− > [a].
n −− > [man].
n −− > [woman].
v −− > [shoots].
conj −− > [and].
conj −− > [or].
conj −− > [but].

DCGs for Formal Languages

A formal language is simply a set of strings. We will define the language $a^n b^n$, where the must be the same number of a's as b's.

s −− > [ ].
s −− > l,s,r.
l −− > [a]
r −− > [b]