Chapter 3

Definite Clause Grammars for NL

In this chapter we look at a couple of simple examples of solving some NL problems using DCGs. The first shows how noun-verb agreement can be achieved using the variables which can appear in non-terminals in DCGs. Only the present tense will be illustrated. We then consider the problem of translating from one language to another. Here a sentence in French is parsed/recognised and at the same time the English translation is constructed in one of the variables.

We begin with a general introduction to the syntax and translation of DCG in Prolog. Most versions of Prolog have the capability of defining languages and operating on them by means of definite clause grammars.

These types of grammars are similar to cfgs but are strictly more powerful as they do have some context sensitive features.

3.1 Simplest Form

The most basic form of dcs are essentially the same as cfgs. We give an example to illustrate this. Once the rules are presented to the Prolog interpreter they are translated into pure Prolog clauses. If a listing is performed then the translations can be seen.

3.1.1 Example

Here we have a grammar to recognise various forms of numbers. Notice how close it is to the normal cfg definition.

\[
\text{digit} \rightarrow [\text{0}] | [\text{1}] | [\text{2}] | [\text{3}] | [\text{4}] | [\text{5}] | [\text{6}] | [\text{7}] | [\text{8}] | [\text{9}] \%\text{ a digit is in 0-9}
\]

\[
\text{nat_num} \rightarrow \text{digit} \%\text{ a natural number is a}
\text{nat_num} \rightarrow \text{digit},\text{nat_num} \%\text{ sequence of digits}
\]

\[
\text{int} \rightarrow \text{nat_num} \%\text{ an integer is a natural number}
\]
int --> sign, nat_num. % possibly with a sign
real --> int. % a real is given in normal
real --> int, ['.'], nat_num. % decimal notation
real --> sign, ['.'], nat_num.

sign --> [-] | [+].

The objects inside []'s are terminal elements of the language, the other identifiers play the role of non-terminals.

We can use '|' to separate alternatives on the rhs of a rule or give them as separate rules.

The above is translated into the following Prolog clauses

\[
\begin{align*}
\text{nat_num}(A, B) & : - \\
& \text{digit}(A, B). \\
\text{nat_num}(A, B) & : - \text{digit}(A, C), \\
& \text{nat_num}(C, B). \\
\text{sign}(A, B) & : - \\
& ( 'C'(A, -, B) | \\
& 'C'(A, +, B) ). \\
\text{real}(A, B) & : - \\
& \text{int}(A, B). \\
\text{real}(A, B) & : - \\
& \text{int}(A, C), \\
& 'C'(C, '.', D), \\
& \text{nat_num}(D, B). \\
\text{real}(A, B) & : - \\
& \text{sign}(A, C), \\
& 'C'(C, '.', D), \\
& \text{nat_num}(D, B). \\
\text{int}(A, B) & : - \\
& \text{nat_num}(A, B). \\
\text{int}(A, B) & : - \\
& \text{sign}(A, C), \\
& \text{nat_num}(C, B). \\
\text{digit}(A, B) & : - \\
& ( 'C'(A, 0, B) | \\
& 'C'(A, 1, B) | \\
& 'C'(A, 2, B) | \\
& 'C'(A, 3, B) | \\
& \ldots ) .
\end{align*}
\]
'C'(A, 4, B) |
'C'(A, 5, B) |
'C'(A, 6, B) |
'C'(A, 7, B) |
'C'(A, 8, B) |
'C'(A, 9, B) ).

If we take the clause \texttt{nat} \texttt{num}(A,B), then the interpretation of this predicate is that the sequence of elements in B is an end part of A, and that the sub-sequence of A obtained by cutting off B satisfies the definition of natural number. In other words we are essentially using the pair (A,B) as a difference list.

So for example \texttt{nat_num}([1,2,3],[]) is true.

More generally so too is \texttt{nat_num}([1,2,3|X],X) for any X. The special built in predicate 'C' is used for recognising the terminals and 'C'(X,Terminal,Y) means that Terminal is the head of X and it's tail is Y — as if it were defined as 'C'([X|Xs],X,Xs).

To make the above easier to use we could write a procedure which would convert a number in normal representation into a sequence of the separate symbols i.e. ‘+123’ would be converted to the list ['+',1,2,3] and ‘1.23’ would be converted to [1,'.',2,3] etc. Note it is essential to quote the ‘.’ otherwise Prolog will take it as a terminator for a Prolog clause and so give an error. Also if the number contains a sign it must be quoted as \texttt{name} requires an atom and +123 for example is not an atom (as begins with ‘+’).

\begin{verbatim}
convert_to_list(Atom,Lst):-
  name(Atom,AtomLst),
  % convert to list of ascii numbers
  list_to_vals(AtomLst,Lst).

ascii_to_char(Asc,Chr) :-
  name(Chr,[Asc]).

list_to_vals([],[]).  % apply down a list

list_to_vals([H|T],[HV|TV]) :-
  ascii_to_val(H,HV),
  list_to_vals(T,TV).
\end{verbatim}

Instead of using the translated predicates to recognise elements of a language we can also use the built in predicate \texttt{phrase}. \texttt{phrase(NT,Lst)} means Lst is generated by the non-terminal NT. e.g. \texttt{phrase(real,[1,2,'.',4,5])} is true.

Now the above grammar allows numbers with leading zeroes such as 003. It is not too difficult to alter the grammar by introducing new non-terminals to reject such numbers. The changes are as follows:

\begin{verbatim}
nonz_digit --> [1]| [2]| [3]| [4]| [5]| [6]| [7]| [8]| [9]. % a nonz_digit is in 1-9
digit --> [0]| nonz_digit.
red_nat_num --> digit| nonz_digit, nat_num.
red_int --> red_nat_num| sign, red_nat_num.
red_real --> red_int| red_int, ['.'], nat_num.

This will then return false for phrase(red_nat_num, [0,0,2,3]) but true for phrase(nat_num, [0,0,1,2]).

3.2 More Complex Examples

Dcgs also have the facility of embedding Prolog code in their bodies. This is achieved by placing the required code in `{}`. Anything inside `{}` is then left unaltered by the interpreter. Parameters can also appear as arguments to the non-terminal symbols so that results can be returned as a side effect to the language recognition or used in further calculations.

3.2.1 Example

We add a parameter to the previous example so that after recognising a correct number it will contain the value of that number. We also need to carry how many digits appear in the number.

e. g. phrase(nat_num(N), [1,2,3,1]) will instantiate the variable N to the value 1231

We shall not bother with the reduced form but take the original grammar.

digit(0) --> [0]. % a digit is in 0-9
digit(1) --> [1].
digit(2) --> [2].
digit(3) --> [3]. % etc
:
digit(9) --> [9].

nat_num(N, 1) --> digit(N). % a natural number is a sequence of digits

nat_num(N, ND) --> digit(D), nat_num(N2, ND1),
% ND is the number of digits
{plus(ND1, 1, ND),
\[\text{power\_ten}(D,ND1,P),\]
\[\text{plus}(P,N2,N)\}.

\[\text{int}(N,D) \rightarrow \text{nat\_num}(N,D). \text{ an integer is a natural number}\]
\[\text{% possibly with a sign}\]
\[\text{int}(N,D) \rightarrow [+],\text{nat\_num}(N,D).
\]
\[\text{int}(N,D) \rightarrow [-],\text{nat\_num}(N1,D),\]
\[\{ N \text{ is } - N1\}.
\]

\[\text{real}(R) \rightarrow \text{int}(R,\ldots). \text{ a real is given in normal decimal notation}\]
\[\text{real}(R) \rightarrow \text{int}(I,\ldots),[\ldots],\text{nat\_num}(N,ND),\]
\[\{\text{neg\_power\_ten}(ND,InvP),\]
\[\quad (I \geq 0 \rightarrow R \text{ is } I + N \times InvP;\]
\[\quad \quad R \text{ is } I - N \times InvP) \}.
\]
\[\text{real}(R) \rightarrow [+],[\ldots],\text{nat\_num}(N,D),\]
\[\{ \text{neg\_power\_ten}(D,InvP),\]
\[\quad R \text{ is } N \times InvP \}.
\]
\[\text{real}(R) \rightarrow [-],[\ldots],\text{nat\_num}(N,D),\]
\[\{ \text{neg\_power\_ten}(D,InvP),\]
\[\quad R \text{ is } - N \times InvP \}.
\]

\[\text{power\_ten}(D,0,D) :-!.
\]
\[\text{power\_ten}(D,E1,P) :-\]
\[\quad E1 > 0,\]
\[\quad \text{plus}(E,1,E1),\]
\[\quad \text{power\_ten}(D,E,P1),\]
\[\quad P \text{ is } P1 \times 10,!.
\]

\[\text{neg\_power\_ten}(0,1):-!.
\]
\[\text{neg\_power\_ten}(N,P):-\]
\[\quad N > 0,\]
\[\quad \text{succ}(N1,N),\]
\[\quad \text{neg\_power\_ten}(N1,P1),\]
\[\quad P \text{ is } P1/10 .
\]

A call of for example \text{real}(R,[-,2,3,’.’,4,5],[]) returns true and instantiates \(R\) to the real number \(-23.45\).

A call of \text{phrase}(\text{int}(N,D),[-,1,2,3]) instantiates \(N\) to the number \(-123\) and \(D\), the number of digits, to 3.
3.2.2 Example

% parse simple english sentence
% for time being just present as a list of identifiers
% e.g. [the,big,cat,kicks, the, black, dog]
% first we have a straight forward generator
% will not repeat adjective like big big girl!
% Also check whether we need an 'an' or an 'a'.

adjnounph(CV) --> noun(CV).
adjnounph(CV) --> adjective(CV,Adj),noun(_).
adjnounph(CV) --> adjective(CV,Adj),adjective(_,Adj2),
{Adj \(\leq\) Adj2},
noun(_).

nounphrase --> det(CV),adjnounph(CV).
sentence --> nounphrase,verb,nounphrase.

% now some explicit examples
det(cons) --> [the]| [a].
det(vowel) --> [an].

verb --> [hit]| [kicks]| [kisses].
noun(cons) --> [cat]| [boy]| [girl].
noun(vowel) --> [owl]| [ox].

adjective(cons,big) --> [big].
adjective(cons,black) --> [black].
adjective(cons,brown) --> [brown].
adjective(cons,tabby) --> [tabby].
adjective(vowel,awful) --> [awful].
adjective(vowel,awesome) --> [awesome].

The above will now correctly recognise an awful owl hit an ox and reject a awesome
cat kisses a awful girl!