

Representing context free grammars in Prolog

- Towards a basic setup:
 - What needs to be represented?
 - On the relationship between context free rules and logical implications
 - A first Prolog encoding
 - Encoding the string coverage of a node:
From lists to difference lists
 - Adding syntactic sugar:
Definite clause grammars (DCGs)
- Representing simple English grammars as DCGs

What needs to be represented?

We need representations (data types) for:

- terminals, i.e., words
- syntactic rules
- linguistic properties of terminals and their propagation in rules:
 - syntactic category
 - other properties
 - string covered (“phonology”)
 - case, agreement, . . .
- analysis trees, i.e., syntactic structures

On the relationship between context free rules and logical implications

- Take the following context-free rewrite rule:

$$S \rightarrow NP VP$$

- One can take the categories in such rules to be predicates holding of strings of words, where 'holding' means that the category spans the string.
- A context free rules then corresponds to a logical implication:

$$\forall X \forall Y \forall Z \text{ NP}(X) \wedge \text{VP}(Y) \wedge \text{append}(X, Y, Z) \Rightarrow \text{S}(Z)$$

- Context-free rules can directly be encoded as logic programs.

A direct representation in Prolog

Components of a direct Prolog encoding:

- terminals:
unit clauses (facts)
- syntactic rules:
non-unit clauses (rules)
- linguistic properties:
 - syntactic category:
predicate name
 - other properties:
predicate's arguments, distinguished by position
 - * in general: compound terms
 - * for strings: list representation
 - analysis trees:
compound term as predicate argument

An example

A small example grammar $G = (N, \Sigma, S, P)$

$N = \{S, NP, VP, V_i, V_t, V_s\}$

$\Sigma = \{\text{John, Mary, laughs, loves, thinks}\}$

$S = S$

$P = \left\{ \begin{array}{l} S \rightarrow NP VP \\ NP \rightarrow Det N \\ NP \rightarrow PN \\ PN \rightarrow Mary \\ Det \rightarrow a \\ N \rightarrow clown \\ VP \rightarrow V_i \\ VP \rightarrow V_t NP \\ VP \rightarrow V_s S \\ V_i \rightarrow laughs \\ V_t \rightarrow loves \\ V_s \rightarrow thinks \end{array} \right\}$

An encoding in Prolog (append_encoding1.pl)

```
pn([mary]).      n([clown]).      det([a]).
vi([laughs]).   vt([loves]).      vs([thinks]).
```

```
s(S) :- np(NP),
        vp(VP),
        append(NP,VP,S).
```

```
np(NP) :- pn(NP).
```

```
np(NP) :- det(Det),
          n(N),
          append(Det,N,NP).
```

```
vp(VP) :- vi(VP).
```

```
vp(VP) :- vt(VT),
          np(NP),
          append(VT,NP,VP).
```

```
vp(VP) :- vs(VS),
          s(S),
          append(VS,S,VP).
```

A modified encoding (append_encoding2.pl)

```
pn([mary]).      n([clown]).      det([a]).  
vi([laughs]).   vt([loves]).      vs([thinks]).
```

```
s(S) :- append(NP,VP,S),  
         np(NP),  
         vp(VP).
```

```
np(NP) :- pn(NP).
```

```
np(NP) :- append(Det,N,NP)  
         det(Det),  
         n(N).
```

```
vp(VP) :- vi(VP).
```

```
vp(VP) :- append(VT,NP,VP),  
         vt(VT),  
         np(NP).
```

```
vp(VP) :- append(VS,S,VP),  
         vs(VS),  
         s(S).
```

Difference list encoding (diff_list_encoding.pl)

```
pn([mary|X],X).
n([clown|X],X).
det([a|X],X).
vi([laughs|X],X).
vt([loves|X],X).
vs([thinks|X],X).

s(X0,Xn) :- np(X0,X1),
            vp(X1,Xn).

np(X0,Xn) :- pn(X0,Xn).
np(X0,Xn) :- det(X0,X1),
            n(X1,Xn).

vp(X0,Xn) :- vi(X0,Xn).

vp(X0,Xn) :- vt(X0,X1),
            np(X1,Xn).

vp(X0,Xn) :- vs(X0,X1),
            s(X1,Xn).
```


Definite clause notation (dcg_encoding.pl)

```
pn --> [mary].  
n --> [clown].  
det --> [a].  
vi --> [laughs].  
vt --> [loves].  
vs --> [thinks].
```

```
s --> np, vp.
```

```
np --> pn.  
np --> det, n.
```

```
vp --> vi.  
vp --> vt, np.  
vp --> vs, s.
```

- By calling the built-in predicate `listing` you can see the resulting Prolog encoding.
- The special predicate `'C'/3` is used to encode coverage of terminals. It can be defined as:

```
'C'([Head|Tail], Head, Tail).
```

Obtaining an analysis tree (dcg_tree.pl)

```
pn(mary_node) --> [mary].
n(clown_node) --> [clown].
det(a_node) --> [a].
vi(laugh_node) --> [laughs].
vt(love_node) --> [loves].
vs(think_node) --> [thinks].

s(s_node(NP, VP)) --> np(NP),
                       vp(VP).

np(np_node(PN)) --> pn(PN).
np(np_node(Det, N)) --> det(Det),
                       n(N).

vp(vp_node(VI)) --> vi(VI).
vp(vp_node(VT, NP)) --> vt(VT),
                       np(NP).
vp(vp_node(VS, S)) --> vs(VS),
                       s(S).
```

Adding more linguistic properties (dcg_linguistic.pl)

```
pn      --> [mary] .
n(sg)   --> [clown] .
n(pl)   --> [clowns] .
det(sg) --> [a] .
det(_)  --> [the] .
```

```
vi(3,sg) --> [laughs] .
vi(_,pl)  --> [laugh] .
vt(3,sg)  --> [loves] .
vt(_,pl)  --> [love] .
vs(_,pl)  --> [think] .
vs(3,sg)  --> [thinks] .
```

```
s --> np(Per,Num), vp(Per,Num) .
```

```
np(3,sg)  --> pn .
np(3,Num) --> det(Num), n(Num) .
```

```
vp(Per,Num) --> vi(Per,Num) .
vp(Per,Num) --> vt(Per,Num), np(_,_) .
vp(Per,Num) --> vs(Per,Num), s .
```