Implementing context-free grammars

Detmar Meurers: Intro to Computational Linguistics I
OSU, LING 684.01

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 → NP VP

- Nonterminals in such a rule can be understood as predicates holding of the lists of terminals dominated by the nonterminal.
- A context-free rules then corresponds to a logical implication:

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

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

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

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

N = {S, NP, VP, V$_i$, V$_t$, V$_s$}

Σ = {a, clown, Mary, laughs, loves, thinks}

S = S

P =

S	→	NP VP	NP	→	Det N
VP	→	V$_i$	NP	→	PN
VP	→	V$_t$ NP	PN	→	Mary
VP	→	V$_s$ S	Det	→	a
V$_i$	→	laughs	N	→	clown
V$_t$	→	loves			
V$_s$	→	thinks			

An encoding in Prolog
dcg/append_encoding1.pl

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

vp(VP) :- vi(VP).
vp(VP) :- vt(VT), np(NP), append(VT,NP,VP).
vp(VP) :- vs(VS), s(S), append(VS,S,VP).

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

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


A modified encoding
dcg/append_encoding2.pl

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

vp(VP) :- vi(VP).
vp(VP) :- append(VT,NP,VP), vt(VT), np(NP).
vp(VP) :- append(VS,S,VP), vs(VS), s(S).

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

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


Difference list encoding
dcg/diff_list_encoding.pl

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

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

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

Basic DCG notation for encoding CFGs

A DCG rule has the form "*LHS* `-->` *RHS*." with

- *LHS*: a Prolog atom encoding a non-terminal, and
- *RHS*: a comma separated sequence of
  - Prolog atoms encoding non-terminals
  - Prolog lists encoding terminals

When a DCG rule is read in by Prolog, it is expanded by adding the difference list arguments to each predicate.

(Some Prologs also use a special predicate 'C'/3 to encode the coverage of terminals, defined as `'C'([Head|Tail],Head,Tail).`)

Examples for some cfg rules in DCG notation

- S → NP VP
  `s --> np, vp.`
- S → NP thinks S
  `s --> np, [thinks], s.`
- S → NP picks up NP
  `s --> np, [picks, up], np.`
- S → NP picks NP up
  `s --> np, [picks], np, [up].`
- NP → ϵ
  `np --> [].`

An example grammar in definite clause notation

dcg/dcg_encoding.pl

```
s --> np, vp.

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

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

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


The example expanded by Prolog

```
?- listing.

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

np(A, B) :-
        pn(A, B).

np(A, B) :-
        det(A, C),
        n(C, B).
```

```
vp(A, B) :-
        vi(A, B).

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

vp(A, B) :-
        vs(A, C),
        s(C, B).
```

```
pn([mary|A], A).

n([clown|A], A).

det([a|A], A).

vi([laughs|A], A).

vt([loves|A], A).

vs([thinks|A], A).
```


More complex terms in DCGs

Non-terminals can be any Prolog term, e.g.:

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

This is translated by Prolog to

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

Restriction:

- The *LHS* has to be a non-variable, single term (plus possibly a sequence of terminals).

Using compound terms to store an analysis tree

dcg/dcg_tree.pl

```
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).

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


Adding more linguistic properties

dcg/dcg_linguistic.pl

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

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

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

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

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


Additional notation: The RHS of DCGs can include

- **disjunctions** expressed by the "`;`" operator, e.g.:

  ```
  vp --> vintr;
         vtrans, np.
  ```

- **groupings** are expressed using parenthesis "`( )`", e.g.

  ```
  vp --> v, (pp_of; pp_at).
  ```

- **extra conditions** expressed as prolog relation calls inside "{ }" (! can also occur, and need not be enclosed by {}):

  ```
  s --> np(Case), vp, {check_case(Case)}.

  s --> {write('in rule 1'),nl},
        np, {write('after np'),nl},
        vp, {write('after vp'),nl}.
  ```


Additional notation for the RHS of DCGs: Meta-variables

On the *RHS*, variables can be used for non-terminals and terminals, i.e. as meta-variables. E.g.:

```
verb([up]) --> [pick].

vp --> verb(Particle),      % pick
       np,                  % the ball
       Particle.            % up
```

Note: The value of the variable has to be known at the time Prolog attempts to prove the subgoal represented by the variable.