

LCTG—Notes 1b: Prolog and Definite Clause Grammars

Prolog and Logic Programming

- We are used to seeing recursive definitions as procedures in functional programming languages:

```
fun member(X,L) =  
  if L= nil then false  
    elseif X = hd(L) then true else member(X, tl(L));
```

- or even

```
fun member x [] = false  
  | member x (y :: l) = if (x=y) then true else member x l;
```

—where $x :: y$ means “the list with x as the head and y as the tail”

- However, we can also think of them as collections of formulae in first order predicate logic:

$$\forall X \forall Y. \text{member}(X, X :: Y)$$

$$\forall X \forall Y \forall Z. \text{member}(X, Z) \Rightarrow \text{member}(X, Y :: Z)$$

1

2

Algorithm = Logic + Control

- This has given rise to the above slogan, and the family of Logic Programming Languages of which Prolog is the first and simplest example:

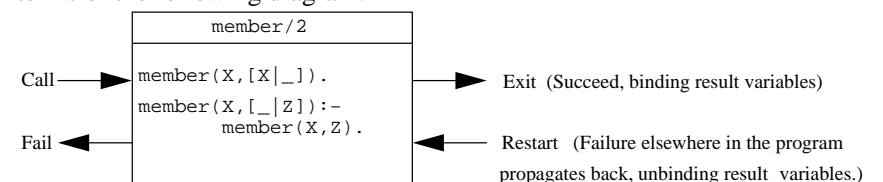
```
member(X, [X|_]).  
member(X, [_|Z]) :- member(X,Z).
```

- Notice that the test needed in the functional program for the empty list case is unnecessary, because falsity is equated with failure in logic programming and the program fails anyway for that case.
- Notice that the **order** in which the formulae or **clauses** are executed is crucial. That is (part of) the “Control” part.

3

Clauses as Functions/Relations

- It is standard to think of a set of definite clauses like `member/2` as procedures in terms of the following diagram:



- Thus Prolog procedures are characterised by being nondeterministic and are run under a **backtracking** control.
- You can observe this process by issuing the command `debug()` ., and using the predicate `spy`, as in `spy(member)` ..
- There is a complete manual for sicstus prolog and an easy short course on the class webpage. If this is your first programming language ever, you should consider taking the Course AI Programming in Prolog.

4

Prolog Definite Clauses and CFPSG

- Rewrite rules $S \rightarrow NP VP$ are isomorphic to prolog **Definite clauses**

$S :- NP, VP.$

—but we need to induce a notion of linear order, which is not implicit in the definite clause.

- First we will do this laboriously, with explicit string indices.
- Then we will use a “sugared” notation offered by prolog, which compiles the indices away and makes them invisible.

5

```
det(P0, P) :- connects(the, P0, P).
```

```
n(P0, P) :- connects(frog, P0, P).
```

```
/* tedious statement of sentence */
```

```
connects(harry, 0, 1).
connects(sees, 1, 2).
connects(the, 2, 3).
connects(frog, 3, 4).
connects(that, 4, 5).
connects(walks, 5, 6).
```

```
%| ?- [dcg0].
%% consulting /amd/nfs/pegasus/disk/ptn053/steedman/prolog/dcg0...
%% consulted /amd/nfs/pegasus/disk/ptn053/steedman/prolog/dcg0 in module user, 0 msec -88 bytes
%yes
%| ?- s(0,6).
%yes
%| ?- s(0,5).
%no
%| ?- s(0,4).
%yes
%| ?-
```

7

Minimal Definite Clause Grammar DCG

```
s(P0, P) :-
    np(P0, P1),
    vp(P1, P).

np(P0, P) :-
    pn(P0, P).

np(P0, P) :-
    det(P0, P1),
    n(P1, P2),
    optrel(P2, P).

vp(P0, P) :-
    tv(P0, P1),
    np(P1, P).

vp(P0, P) :-
    iv(P0, P).

optrel(F,P).

optrel(P0,P) :-
    connects(that, P0, P1), vp(P1,P).

pn(P0, P) :- connects(harry, P0, P).
pn(P0, P) :- connects(harry, P0, P).

iv(P0, P) :- connects(walks, P0, P).

tv(P0, P) :- connects(sees, P0, P).
```

6

Difference List Position Encoding in DCGs

```
%%% Difference lists --- see p&S p.126-7

conc_dl(Front-Back1, Back1-Back2, Front-Back2).

%| ?- [conc_dl].
%% consulting /amd/nfs/pegasus/disk/ptn053/steedman/lectures/tl/conc_dl...
%% consulted /amd/nfs/pegasus/disk/ptn053/steedman/lectures/tl/conc_dl in module user, 0 msec -88 bytes
%yes
%| ?- conc_dl([1,2,3|X]-X, [4,5,6|Y]-Y, R).
%R = [1,2,3,4,5,6|Y]-Y,
%X = [4,5,6|Y] ?
%yes
%| ?-

%%% Reflect that S -> NP VP is equivalent to saying
%%% s(W) :- np(U), vp(V), conc_dl(U,V,W).
%%%
%%% This can be partially executed to get rid of the last literal by
%%% writing
%%% s(P0-P) :- np(P0-P1), vp(P1-P).
%%%
%%% The following clause then induces a representation of string position
%%% in terms of the substring following the position.

connects(Word, [Word|Rest], Rest).

%| ?- [dcg0].
% {consulting /home2/steedman/prolog/dcg/dcg0...}
% {/home2/steedman/prolog/dcg/dcg0 consulted, 30 msec 12208 bytes}
%
```

8

```

% yes
% | ?- [conc_dl].
% {consulting /home2/steedman/prolog/dcg/conc_dl...}
% The procedure connects/3 is being redefined.
%   Old file: /home2/steedman/prolog/dcg/dcg0
%   New file: /home2/steedman/prolog/dcg/conc_dl
% Do you really want to redefine it? (y, n, p, or ?) y
% {/home2/steedman/prolog/dcg/conc_dl consulted, 10 msec -208 bytes}
%
% yes
% | ?- s([harry,sees,the,frog,that,walks],[ ]).
%
% yes
% | ?- s([harry,sees,the,frog,that],[ ]).
%
% no
% | ?- s([harry,sees,the,frog],[ ]).
%
% yes

%%% Running in (-,+) mode
% | ?- s(L,[ ]).
%
% L = [harry,sees,harry] ? ;
%
% L = [harry,sees,barry] ? ;
%
% L = [harry,sees,the,frog] ? ;
%
% L = [harry,sees,the,frog,that,sees,harry] ? ;
%
% L = [harry,sees,the,frog,that,sees,barry] ? ;
%
% L = [harry,sees,the,frog,that,sees,the,frog] ? ;
%

```

9

Structure-Building DCG

PROLOG in its infinite mercy offers a sugared notation specifically for DCGs, in which the operator `-->` replaces `:-` and the position variables are suppressed. An arbitrary call can be appended enclosed in `{. . .}` brackets. To call the DCG you need to know that any arguments of a term like `s(Tree)` have to appear before the sentence list, and the empty list has to appear after. So to call the first rule below you need a goal like `s(Tree,[the, cat, sat, on, the, mat],[])`. We will use this notation from now on, remembering that it is just a sugared version of the earlier one.

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%                               %%
%%   Syntactic structure-building DCG in DCG notation                %%
%%                               %%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%% Stuff to overcome Sictus' ever-so-helpful abbreviation of list terms

portray(Term) :- is_list(Term), write(Term).

is_list([]).
is_list(_:_).

%%% End of stuff

s([s,[NP, VP]]) --> np(NP), vp(VP).

np([np,[PN]]) --> pn(PN).

```

11

```

% L = [harry,sees,the,frog,that,sees,the,frog,that,sees,harry] ? ;
%
% L = [harry,sees,the,frog,that,sees,the,frog,that,sees,barry] ? ;
%
% L = [harry,sees,the,frog,that,sees,the,frog,that,sees,the,frog] ?
%%% NB we never get e.g the frog sees harry.

```

10

```

np([np,[Det, N, Rel]]) -->
  det(Det), n(N), optrel(Rel).

vp([vp,[TV, NP]]) -->
  tv(TV),
  np(NP).

vp([vp,[IV]]) -->
  iv(IV).

optrel([rel,[epsilon]]) --> [].

optrel([rel,[that, VP]]) -->
  [that], vp(VP).

%pn(harry) --> [harry].
%% etc. -- very tedious, therefore ....

pn(Word) --> [Word], {pn(Word)}.
pn(harry).
pn(barry).
pn(they).

iv(Word) --> [Word], {iv(Word)}.
iv(walks).
iv(walk).

tv(Word) --> [Word], {tv(Word)}.
tv(sees).
tv(see).

det(Word) --> [Word], {det(Word)}.
det(a).
det(some).

```

12

```

n(Word) --> [Word], {n(Word)}.
n(frog).
n(frogs).

%| ?- ['hw1.p'].
%% consulting /amd/nfs/pegasus/disk/ptn053/steedman/lectures/tl/hw1.p...
%% consulted /amd/nfs/pegasus/disk/ptn053/steedman/lectures/tl/hw1.p in module user, 0 msec -88 bytes
%yes
%| ?- s(T, [harry, walks], []).
%
%T = [s,[[np,[harry]], [vp,[walks]]]] ?
%
%yes
%| ?- s(T, [harry, walk], []).
%
%T = [s,[[np,[harry]], [vp,[walk]]]] ?      % Note the overgeneralisation !
%
%yes
%| ?- s(T, [harry, sees, a, frog, that, walks], []).
%
%T = [s,[[np,[harry]], [vp,[sees,[np,[a,frog],[rel,[that,[vp,[walks]]]]]]]]]] ?
%
%yes
%| ?- s(T, [harry, sees, a, frog, that, walk], []).
%
%T = [s,[[np,[harry]], [vp,[sees,[np,[a,frog],[rel,[that,[vp,[walk]]]]]]]]]] ?
%
%yes
%| ?-

%%%Running s in (-,+ mode:

%| ?- s([s,[[np,[harry]], [vp,[sees,[np,[a,frog],[rel,[that,[vp,[walks]]]]]]]]]]], L, []).
%L = [harry,sees,a,frog,that,walks] ?

```

13

Agreement

- The above grammar wrongly accepts (“overgenerates”) sentences violating “number agreement” like **Harry walk* and **Boys talks*, as well as correctly allowing *Harry walks* and *Boys talk*.
- Since agreement can occur between nonadjacent string elements as in *Harry₁/*boys seems/*seem to like himself₁*, this phenomenon was original thought to require rules of greater than context free power.
- However it is quite easy to impose agreement by annotating symbols like *NP* with one or more features, or attribute-value pairs, written here as subscripts, capitals for unspecified attributes and lowercase for values, as in the next slide. Provided that the values of attributes are just constants (rather than structures with variables, stacks, or other non-finite structures) this does not affect their context-free power.
- The homework asks you to do this as a DCG.

15

```

%yes
%| ?- s([s,[[np,[a,frog],[rel,[epsilon]]], [vp,[sees,[np,[harry]]]]]]], L, []).
%L = [a,frog,sees,harry] ?
%yes
%| ?-

```

14

The Toy CFPS Grammar with Agreement

$$\begin{aligned}
 S &\rightarrow NP_{AGR} VP_{AGR} \\
 NP_{AGR} &\rightarrow \begin{cases} Det_{AGR} & N_{AGR} \\ PN_{AGR} \\ TV_{AGR} & NP \\ \dots \end{cases} \\
 VP_{AGR} &\rightarrow \begin{cases} PN_{AGR} \\ TV_{AGR} & NP \\ \dots \end{cases} \\
 S_{comp} &\rightarrow that S \\
 N_{sing} &\rightarrow \{man, woman, boy, girl, \dots\} \\
 N_{plur} &\rightarrow \{men, women, boys, girls, \dots\} \\
 TV_{sing} &\rightarrow \{sees, likes, loathes, \dots\} \\
 TV_{plur} &\rightarrow \{see, like, loath, \dots\} \\
 Det_{sing} &\rightarrow \{the, a, \dots\} \\
 Det_{plur} &\rightarrow \{the, several, \dots\} \\
 &\dots
 \end{aligned}$$

16