

---

## LCTG—Notes 6: Compositional Semantics in DCG

---

## Surface Structures vs. Meaning Representations

- The structure-building DCGs seen so far just build surface structures. We want a DCG to which build prolog queries, that can be treated as sentence meanings and be evaluated on a simple database.
- That is, we want the result of evaluating the query `s(T, [harry, sees, barry], [])` to yield `T = sees(harry, barry)`, and the query `s(T, [harry, sees, barry], [])`, `call(T)` to yield the answer Yes, or No, depending on the database.
- To do so, we need to define the meaning of words like *sees* as functions from arguments to propositions.

1

2

---

## $\lambda$ -Terms as Meaning Representations

- The lambda calculus provides a convenient way to identify functions. For example, the interpretation of *sees* is the following:  
 $sees : \lambda x. \lambda y. sees(y, x)$   
—a function of type  $e \rightarrow (e \rightarrow t)$ , where  $e$  is the type of an entity like Harry, and  $t$  (for truth-value) is the type of a proposition.
- The  $\lambda$  calculus is not first order, so we have to fake it in Prolog.

---

## The Wrong Way to do Semantics

- Our first attempt at providing a semantics of this kind for the DCG in Homework 2 might be to build the following sort of DCG
- Note that as usual we overload constants like `frog` as both the lexical item and an element of its interpretation):

3

4

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
%% (Program with inadequate semantics Mk.I)
%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

```

```

:- op(500, xfy, &).    % A dummy logical conjunction operator
:- op(510, xfy, =>).  % Dummy implication, in case we want to do "every"

```

```

%% A predicate is a function NP->S; S is instantiated by 'partial execution'.
s(S) --> np(NP), vp(NP^S).

```

```

%% The meaning of a name NP is the name.
np(PN) --> pn(PN).

```

```

%% The next is more difficult. NPprop, Nprop and Nmod are all
%% propositions and the interpretation is conjunctive. It doesnt work.
np(NPprop&Nprop&Nmod,nogap) -->
    det(E^NPprop), n(E^Nprop),optrel(E^Nmod).

```

5

```

det(every,X^forall(X)).

```

```

%% The rest is plain sailing, but should be examined for some notation.
n(LF) --> [N], {n(N,LF)}.
n(author,X^author(X)).
n(book,X^book(X)).
n(program,X^program(X)).
n(programmer,X^programmer(X)).
n(professor,X^professor(X)).
n(student,X^student(X)).

```

```

pn(LF) --> [PN], {pn(PN,LF)}.
pn(begriffsschrift,begriffsschrift).
pn(principia,principia).
pn(lunar,lunar).
pn(shrdlu,shrdlu).
pn(bertrand,bertrand).
pn(gottlob,gottlob).
pn(gilbert,gilbert).

```

7

```

vp(VP) -->
    tv(NP^VP),
    np(NP).

```

```

vp(VP) -->
    iv(VP).

```

```

%% We also get into difficulty here, and this isnt right
optrel(E^true) --> [].

```

```

optrel(X^S) -->
    relpron(X), vp(X^S).

```

```

%% This also isn't right
det(LF) --> [Det], {det(Det,LF)}.
det(a,X^indef(X)).
det(the,X^def(X)).
det(some,X^exists(X)).

```

6

```

pn(george,george).
pn(terry,terry).
pn(bill,bill).

```

```

tv(LF) --> [TV], {tv(TV,LF)}.
tv(concerns,Y^X^concerns(X,Y)).
tv(met,Y^X^met(X,Y)).
tv(ran,Y^X^ran(X,Y)).
tv(wrote,Y^X^wrote(X,Y)).

```

```

iv(LF) --> [IV], {iv(IV,LF)}.
iv(halted,X^halted(X)).
iv(walks,X^walks(X)).

```

```

relpron(LF) --> [RelPron], {relpron(RelPron,LF)}.
relpron(that,LF).
relpron(who,LF).
relpron(whom,LF).

```

8

## The Wrong Way to do Semantics (Contd.)

This will work OK for examples involving proper names:

```
| ?- s(T,[gilbert,halted],[ ]).
T = halted(gilbert) ?
yes
| ?- s(T,[gilbert,met,george],[ ]).
T = met(gilbert,george) ?
yes
```

9

## A Better Way to do Semantics

Instead, we want something more like the following version It uses an approach ultimately derived from the work of Montague.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
%% (Program 3.12 with semantics)
%% This is similar to but different from Program 4.5 in P&S.)
%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

:- op(500, xfy, &).      % A dummy logical conjunction operator
:- op(510, xfy, =>).    % Dummy implication, for when we want to do "every"

%% root S has no gap
s(S) --> s(S,nogap).
```

%% A predicate is a function NP->S; S is instantiated by 'partial

11

## The Wrong Way to do Semantics (Contd.)

— but it won't do for quantified NPs. We can produce structures that contain all the relevant information:

```
| ?- s(T,[some,program,halted],[ ]).
T = halted(exists(_A)&program(_A)&>true) ?
yes
| ?- s(T,[some,program,that,walks,halted],[ ]).
T = halted(exists(_A)&program(_A)&walks(_A)) ?
yes
| ?- s(T,[some,professor,that,walks,ran, every, program],[ ]).
T = ran(exists(_A)&professor(_A)&walks(_A),forall(_B)&program(_B)&>true) ?
yes
| ?-
```

— but they don't constitute well-formed Prolog queries. Nor is it easy to see how to manipulate them to make them into decent Prolog queries. The quantifiers are in the wrong place.

10

%% execution'. But the subject is now a function over predicates:  
s(S1,Gap) --> np(VP^S1,nogap), vp(VP,Gap).

```
%% In fact, ALL NPs, even proper names, are functions over properties
%% (so betrand becomes np((bertrand^S)^S), while walks as always yields
%% vp(X^walks(X)). MAKE SURE YOU UNDERSTAND HOW THIS MAKES THE ABOVE
%% S RULE YIELD S1 = walks(bertrand) BEFORE YOU READ BEYOND THE NEXT LINE):
np((E^S)^S,nogap) --> pn(E).
```

```
%% The real point of this is to make full NPs do the right thing with
%% quantifiers (look at the definition of determiners etc
%% below. They turn it via partial execution into the equivalent of
%% the following:
```

```
%%np((X^S2)^S3,nogap) -->
%%      det((X^S1)^S3), n(X^S0),optrel((X^S0)^S1)).
np(NP,nogap) -->
      det(N2^NP), n(N1),optrel(N1^N2).
```

%% ... And gaps

12

```
np((X^S)^S,gap(np,X)) --> [].
```

```
vp(X^S1,Gap) --> % NB Here we depart from P&S
    tv(Y^X^S), % There is more to this than meets
    np((Y^S)^S1,Gap). % the eye !
```

```
vp(VP,nogap) -->
    iv(VP).
```

```
%% The following bind S0, S1, S2, and S3 in the NP definition:
optrel(N^N) --> [].
```

```
optrel((X^S1)^(X^(S1&S2))) -->
    relpron(X), vp(X^S2,nogap).
```

```
optrel((X^S1)^(X^(S1&S2))) -->
    relpron(X), s(S2,gap(np,X)).
```

13

```
pn(lunar,lunar).
pn(shrdlu,shrdlu).
pn(bertrand,bertrand).
pn(gottlob,gottlob).
pn(gilbert,gilbert).
pn(george,george).
pn(terry,terry).
pn(bill,bill).
pn(mathematics,mathematics).
```

```
tv(LF) --> [TV], {tv(TV,LF)}. % NB Here we depart from P&S
tv(concerns,Y^X^concerns(X,Y)). % Note that this is a more
tv(met,Y^X^met(X,Y)). % traditional semantics for TV
tv(ran,Y^X^ran(X,Y)). % See the transitive vp defn above
tv(wrote,Y^X^wrote(X,Y)).
tv(write,Y^X^wrote(X,Y)).
tv(concern,Y^X^concerns(X,Y)).
```

```
iv(LF) --> [IV], {iv(IV,LF)}.
```

15

```
det(LF) --> [Det], {det(Det,LF)}.
det(a,(X^S1)^(X^S2)^exists(X,S1&S2)).
det(the,(X^S1)^(X^S2)^def(X,S1&S2)).
det(some,(X^S1)^(X^S2)^exists(X,S1&S2)).
det(every,(X^S1)^(X^S2)^all(X,S1=>S2)).
```

```
n(LF) --> [N], {n(N,LF)}.
n(author,X^author(X)).
n(book,X^book(X)).
n(man,X^man(X)).
n(program,X^program(X)).
n(programmer,X^programmer(X)).
n(professor,X^professor(X)).
n(student,X^student(X)).
n(subject,X^subject(X)).
```

```
pn(E) --> [PN], {pn(PN,E)}.
pn(begriffsschrift,begriffsschrift).
pn(principia,principia).
```

14

```
iv(halted,X^halted(X)).
iv(walks,X^walks(X)).
```

```
relpron(LF) --> [RelPron], {relpron(RelPron,LF)}.
relpron(that,LF).
relpron(who,LF).
relpron(whom,LF).
```

This version behaves better. For example, given an appropriate database, including facts like the following –

```
halted(shrdlu).
program(shrdlu).
```

– we can not only build queries:

```
| ?- s(T,[some,program,halted],[ ]).
T = exists(_A,program(_A)&halted(_A)) ?
yes
```

```
| ?- s(T,[every,program,that,some,student,wrote,halted],[ ]).
```

16

```
T = all(_A,program(_A)&exists(_B,student(_B)&wrote(_B,_A))=>halted(_A)) ?
yes
| ?-
```

— We can also (given an appropriate database of prolog facts and appropriate prolog definitions of `exists`, `&` etc.) *run* them to yield truth values:

```
| ?- s(T,[some,program,halted],[ ]), call(T).
T = exists(lunar,program(lunar)&halted(lunar)) ?
yes
| ?- s(T,[some,professor,halted],[ ]), call(T).
no
```

17

- There are two ways to deal with this:
  - Apply movement rules (or equivalent “quantifying in”, “storage”, etc.) to quantifiers at the level of logical form, to give the existential wide scope (cf. Hobbs and Shieber 1987 *CL*, not in bulkpack). This was essentially Montague’s own solution.
  - Build underspecified logical forms that represent both scopes.
- Having gotten rid of movement transformations in the syntax, and having achieved low expressive power and a monotonic relation between syntax and semantics on that basis, it seems a pity to bring back transformations at lf, compromising these gains.
- An interesting variant of the second solution is to treat all NPs other than true universals as *Skolem terms*, underspecified as to their outscoping universals.
- As far as this course goes, we will leave this an open problem.

19

---

## A Further Problem: Quantifier Scope Ambiguity

- The program now couples semantic interpretation and syntactic derivation in a very attractive, fully compositional way, as envisaged by Montague.
- But it still does not capture the semantics of quantifiers completely. Consider the following example:
 

```
| ?- s(T,[every,student,wrote,some,program],[ ]).
T = all(_A,student(_A)=>exists(_B,program(_B)&wrote(_A,_B)))
yes
```
- The program only yields *one* interpretation, whereas there should be a second: `exists(_B,program(_B)&all(_A,student(_A)=>wrote(_A,_B)))`.

18