```
s --> np,vp.

np --> det,n.

vp --> v,np.
vp --> v.

det --> [the].
det --> [a].

n --> [woman].
n --> [man].

v --> [shoots].
```

Suppose we wanted to deal with sentences like ``She shoots him'', and ``He shoots her''. What should we do?

```
pro --> [he].
pro --> [she].
pro --> [him].
pro --> [her].

np--> pro.
```

```
s([she,shoots,him],[]).
yes

s([a,woman,shoots,she],[]).
yes

s([her,shoots,a,man],[]).
yes

s([her,shoots,she],[]).
yes
```

Using extra arguments

```
s --> np(subject),vp.

np(_) --> det,n.
np(X) --> pro(X).

vp --> v,np(object).
vp --> v.

det --> [the].
det --> [a].

n --> [woman].
n --> [man].

v --> [shoots].
```

```
pro(subject) --> [he].
pro(subject) --> [she].
pro(object) --> [him].
pro(object) --> [her].
```

Checking the coverage

You can check it out by posing the query:

?- **`s(X,[]).`**

As you step through the responses, you'll see that only acceptable English is generated.

Syntactic ‘sugar’

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

is really syntactic sugar for

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

Using extra arguments

Ok, so we obviously need to ask what

```
s --> np(subject),vp.
```

translates into. Here's the answer:

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

Extra arguments

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

Using a DCG to build parse trees

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

np(np(DET,N)) --> det(DET),n(N).

vp(vp(V,NP)) --> v(V),np(NP).
vp(vp(V))    --> v(V).

det(det(the)) --> [the].
det(det(a))   --> [a].

n(n(woman)) --> [woman].
n(n(man))   --> [man].

v(v(shoots)) --> [shoots].
```

To parse the sentence ``A woman shoots'' we pose the query:

```
s(T,[a,woman,shoots],[]).
```

That is, we ask for the extra argument T to be instantiated to a parse tree for the sentence. And we get:

```
T = s(np(det(a),n(woman)),vp(v(shoots)))
```

Can we now build semantics representations?