Problem Set 7: Syntactic Analysis
Computer Science 187

Lucas Freitas

April 10, 2013

1. Possessives

The problems below are from section 3.8.2 of PNLA:

Problem 3.15

To handle possessives, we only need to add the following line after the first instance of `det`:

```
det(_, det(NP, [s])) --> np(_, NP), [s].
```

If we try to test the Prolog program for possessives, however, we notice that we have a stack overflow. For instance:

```
?- s(A, B, [frog, s,  cookies], []).
ERROR: Out of local stack
```

That doesn't mean that the program is incorrect. That just means that it is left-recursive, which makes it hard to test due and prone to stack overflows. We will correct that issue in the following items.

Problem 3.16

We can eliminate left-recursion from the possessive grammar by rewriting it into one weakly equivalent to the original one. The way of doing that is transforming a relation of type $A \rightarrow A\beta|\alpha$ into two:

$$A \rightarrow \alpha A'$$

$$A' \rightarrow \beta A'|\epsilon$$

Problem 3.16 (continued)

That can be done by doing:

```
%% basic rules for NP
np(Agr, np(PN)) --> pn(Agr, PN).
np(Agr, np(Det, N)) --> det(Agr, Det), n(Agr, N).

%% NP with possessive (should agree with last element)
np(Agr, np(PN, Rest)) --> pn(_, PN), np_prime(Agr, Rest).
np(Agr, np(Det, N, Rest)) --> det(A, Det), n(A, N), np_prime(Agr, Rest).

%% possessive rules
np_prime(Agr, np([s], N, Rest)) --> [s], n(_, N), np_prime(Agr, Rest).
np_prime(Agr, np([s], N)) --> [s], n(Agr, N).
```

Problem 3.17

The solution for 3.16 already produces a parse tree as in the previous sections.

Sample queries

Sample queries behave as expected:

```
?- s(A,B,[toad,s,cookies,are,frog,s,cookies],[]).
A = agr(pl, third),
B = s(np(pn(toad), np([s], n(cookies))), vp(v(are),
cp(np(pn(frog), np([s], n(cookies)))))) .

?- s(A,B,[toad,s,cookies,s,cake,bakes,the,cookies],[]).
A = agr(sg, third),
B = s(np(pn(toad), np([s], n(cookies), np([s], n(cake)))),
vp(v(bakes), cp(np(det(the), n(cookies))))) .

?- s(A,B,[toad,s,cookies,s,cake],[]).
false.
```

2. Prepositional Phrases

Problem 3.20

To add prepositional phrases, we first need to make adverbial prepositional phrases siblings of S instead of VP, and nominal prepositional phrases, siblings of NP. Thus, all we need to add to our grammar is:

```
%% Prepositional phrases
s(Agr, s(NP, VP, Prep)) --> np(Agr, NP), vp(Agr, VP,_,_), pp(Prep).
...
pp(pp(Prep, NP)) --> p(Prep), np(_, NP).
np(Agr, np(Det, N, Prep)) --> det(Agr, Det), n(Agr, N), pp(Prep).
...
p(p(X)) --> [X], {p(X)}.
p(for).     p(to).      p(in).
p(on).      p(above).   p(below).
p(with).    p(every).
```

Notice how useful weakly equivalence is for helping determine what parse trees look like, and solve the left-recursion on the program.

Sample queries

Sample queries behave as expected:

```
?- s(A,B,[toad,bakes,the,cookies,in,the,kitchen],[]).
A = agr(sg, third),
B = s(np(pn(toad)), vp(v(bakes), cp(np(det(the), n(cookies),
pp(p(in), np(det(the), n(kitchen))))))) .

?- s(A,B,[every,cookie,in,toad,s,box,dies],[]).
A = agr(sg, third),
B = s(np(det(every), n(cookie), pp(p(in), np(pn(toad), np([s],
 n(box))))), vp(v(dies))) .

?- s(A,B,[toad,met,a,box,with,a,cake],[]).
A = agr(sg, third),
B = s(np(pn(toad)), vp(v(met), cp(np(det(a), n(box), pp(p(with),
 np(det(a), n(cake))))))) .

?- s(A,B,[toad,baked,a,cake,for,a,cookie],[]).
A = agr(sg, third),
B = s(np(pn(toad)), vp(v(baked), cp(np(det(a), n(cake),
pp(p(for), np(det(a), n(cookie))))))) .
```

3. Verb Subcategorization

Problem 3.23

Following the guidelines on PNLA, we can add verb subcategorization to our grammar by adding an argument, Type, to represent the transitivity of the verb. We can also create a new component for a grammar: verbal complements (cp). Using that method, all that we need to do is:

```
%% Verbal phrases

vp(Agr, vp(V), intransitive, Conj) -->
               v(Agr, V, intransitive, Conj).
vp(Agr, vp(V, Comps), Type, Conj) -->
               v(Agr, V, Type,Conj), cp(Type, Comps).

%% Verbal complements

cp(transitive, cp(NP)) --> np(_, NP).
cp(ditransitive, cp(NP1, NP2)) --> np(_, NP1), np(_, NP2).
cp(dative(Prep), cp(NP1, pp(pp(p(Prep), NP2)))) -->
               np(_, NP1), pp(pp(p(Prep), NP2)).
...

%% Verbs

v(agr(sg,first), v(X), Type, Conj) --> [X], {vlex(X,_,_,_,Type,Conj)}.
v(agr(sg,second), v(X), Type, Conj) --> [X], {vlex(_,X,_,_,Type,Conj)}.
v(agr(sg,third), v(X), Type, Conj) --> [X], {vlex(_,_,X,_,Type,Conj)}.
v(agr(pl,_), v(X), Type, Conj) --> [X], {vlex(_,_,_,X,Type,Conj)}.

% to meet
vlex(meet, meet, meets, meet, transitive).

% to be
vlex(am, are, is, are, transitive).

% to bake
vlex(bake, bake, bakes, bake, ditransitive).
vlex(bake, bake, bakes, bake, transitive).
vlex(bake, bake, bakes, bake, dative(for)).
vlex(bake, bake, bakes, bake, dative(in)).

% to put
vlex(put, put, puts, put, dative(in)).
```

```
% to die
vlex(die, die, dies, die, intransitive).

% to give
vlex(give, give, gives, give, ditransitive).
vlex(give, give, gives, give, dative(to)).
```

Sample queries

Sample queries behave as expected:

```
?- s(A,B,[toad,bakes,the,cookies],[]).
A = agr(sg, third),
B = s(np(pn(toad)), vp(v(bakes), cp(np(det(the), n(cookies))))) .

?- s(A,B,[toad,bakes,frog,the,cookies],[]).
A = agr(sg, third),
B = s(np(pn(toad)), vp(v(bakes), cp(np(pn(frog)), np(det(the),
n(cookies))))) .

?- s(A,B,[toad,bakes,the,cookies,to,frog],[]).
A = agr(sg, third),
B = s(np(pn(toad)), vp(v(bakes), cp(np(det(the), n(cookies),
pp(p(to), np(pn(frog))))))) .

?- s(A,B,[toad,bakes,the,cookies,to,frog,in,the,kitchen],[]).
A = agr(sg, third),
B = s(np(pn(toad)), vp(v(bakes), cp(np(det(the), n(cookies),
pp(p(to), np(pn(frog)))), pp(pp(p(in), np(det(the),
n(kitchen))))))) .

?- s(A,B,[toad,dies],[]).
A = agr(sg, third),
B = s(np(pn(toad)), vp(v(dies))) .
```

4. Auxiliary Verbs

Problem 4.2.1

Using a similar approach to Problem 3, we can add another argument for the conjugation of the verb, `Conj`, and do a similar approach to the one suggested in PNLA. The declaration of all verbe tenses was supressed from this writeup, but can be found at `frog-toad-3a.pl`.

Prolog code

```
%% Auxiliar verbs

vp(Agr, vp(Aux, V), intransitive, Conj) -->
        aux(Aux, Conj / Req), vp(Agr, V, intransitive, Req).
vp(Agr, vp(Aux, Rest), Type, Conj) -->
        aux(Aux, Conj / Req), vp(Agr, Rest, Type, Req).
...
aux(aux(X), Form) --> [X], {aux(X,Form)}.

% can
aux(can, none / nonfinite).

% could
aux(could, finite / nonfinite).

% have
aux(have, nonfinite / past).

% been
aux(been, past / present).

% be
aux(be, nonfinite / present).
```

5. Sample queries

All works as expected.

```
?- s(Agr,S,[toad,gave,some,cookies,to,frog],[]).
Agr = agr(sg, third),
S = s(np(pn(toad)), vp(v(gave), cp(np(det(some), n(cookies)),
pp(pp(p(to), np(pn(frog))))))) .

?- s(Agr,S,[toad,gave,frog,some,cookies],[]).
Agr = agr(sg, third),
S = s(np(pn(toad)), vp(v(gave), cp(np(pn(frog)), np(det(some),
n(cookies))))) .

?- s(Agr,S,[frog,put,the,cookies,in,the,box],[]).
Agr = agr(sg, third),
S = s(np(pn(frog)), vp(v(put), cp(np(det(the), n(cookies)),
pp(pp(p(in), np(det(the), n(box))))))) .

?- s(Agr,S,[frog,put,the,box,some,cookies],[]).
false.

?- s(Agr,S,[frog,put,the,cookies],[]).
false.

?- s(Agr,S,[toad,baked,some,cookies,for,frog],[]).
Agr = agr(sg, third),
S = s(np(pn(toad)), vp(v(baked), cp(np(det(some), n(cookies),
pp(p(for), np(pn(frog))))))) .

?- s(Agr,S,[toad,baked,some,cookies,in,the,kitchen],[]).
Agr = agr(sg, third),
S = s(np(pn(toad)), vp(v(baked), cp(np(det(some), n(cookies),
pp(p(in), np(det(the), n(kitchen))))))) .

?- s(Agr,S,[toad,gave,frog,some,cookies,in,the,kitchen],[]).
Agr = agr(sg, third),
S = s(np(pn(toad)), vp(v(gave), cp(np(pn(frog)), np(det(some),
n(cookies), pp(p(in), np(det(the), n(kitchen))))))) .

?- s(Agr,S,[toad,can,bake,a,cake],[]).
Agr = agr(sg, third),
S = s(np(pn(toad)), vp(aux(can), vp(v(bake), cp(np(det(a),
n(cake)))))) .
```

```
?- s(Agr,S,[toad,could,have,been,baking,a,cake],[]).
Agr = agr(sg, third),
S = s(np(pn(toad)), vp(aux(could), vp(aux(have), vp(aux(been),
vp(v(baking), cp(np(det(a), n(cake)))))))) .

?- s(A,B,[toad,could,have,bake,a,cake],[]).
false.
```