Natural Language to Predicate Calculus

Logic/Prolog Module, Week 3-4
Esty (Katherine) Thomas

Goal:

The goal of this assignment was basically to create a program to parse sentences and transform them into statements of First Order Logic. This particular iteration was intended to parse sentences generated using a particular simple phrase structure grammar (Appendix A) and a rather small lexicon (Appendix B), and to translate from English into FOL the subset of those sentences using only the determiners "all" and "some" in the noun phrases.

Methods:

Parsing:

The parser consists almost entirely of the grammar, with a few extra features. The grammar is described using Prolog DCG notation (head --> elem1, elem2, …). Each type of phrase has the relevant semantic features for parameters, and also has as one of its parameters the structure of its parse tree, including variables in order for each of its subparts. The parse tree for a terminal (det, noun, verb, adj, or rel) looks like this: Category(Word). This is a parse tree for a simple np, with a determiner and noun: np(det(the), noun(tree)).

Translating:

The translator works upon the parse trees passed out by the parser. Its basic structure is that described in the tutorial.

At the simplest level, adjectives and nouns are stripped out of their parse tree and turned into "function-shaped" statements using the =.. operator (so noun(pie) → pie → pie(x)). Only two determiners are accepted: "all" is the universal quantifier, while "some" corresponds to the existential quantifier. Noun phrases also become "function-shaped" using the =.. operator. If the parent noun phrase contains a noun and an adjective, the two statements are stuck together with '&' (so 'some pie' → exists(x, pie(x)), 'all tasty pie' → all(x, pie(x)&tasty(x))).

The translation of a verb phrase depends upon the preceding noun phrase's translation.

Verb phrases within a universal statement produce an implicatory statement. If the verb phrase contains a form of 'to be' followed by an adjective, then for "all(x, B)" we get "all(x, B => C)", where C is the adjective applied to x. If the verb phrase contains an intransitive verb, we get "all(x, B => C)", where C is the verb applied to x. If the verb phrase contains a transitive verb and an object noun phrase, we get "all(x, B => C)", where C is the verb applied to x and the translation of the noun phrase applied to x2.

Universal verb phrases within a relative clause produce a statement with '&'. The correspondences are much the same as above.

Existential verb phrases produce a statement where each element is merely put in sequence. So for a verb phrase containing a form of 'to be' and an adjective, for "exists(x, B)" we get "exists(x, B, C)", where C is the adjective applied to x. It is similar for intransitive and transitive verbs.

Translation of a full sentence, containing an NP and a VP, requires translating the NP and using that translation for the VP. Variables are generated with gensym.

Results:

A transcript of interactions:

```
?- sentence(T,[the,boy,runs],[]).
T = sentence(np(det(the), noun(boy)), vp(verb(runs))) .

?- sentence(T,[girls,like,an,apple],[]).
T = sentence(np(noun(girls)), vp(verb(like), np(det(an), noun(apple)))) .

?- sentence(T,[a,government,that,conscripts,people,is,evil],[]).
T = sentence(np(det(a), noun(government), rel(rel(that), vp(verb(conscripts),
np(noun(people))))), vp(verb(is), adj(evil))) .

?- sentence(T,[the,boy,whom,the,girl,likes,likes,a,watermelon],[]).
T = sentence(np(det(the), noun(boy), rel(rel(whom), np(det(the), noun(girl)), vp(verb(likes)))),
vp(verb(likes), np(det(a), noun(watermelon)))) .

?- sentence(T,[the,boy,run],[]).
false.

?- sentence(T,[girls,likes,an,apple],[]).
false.

?- sentence(T,[a,government,that,conscripts,people,are,evil],[]).
false.

?- sentence(T,[the,boy,who,the,girl,likes,likes,a,watermelon],[]).
false.

?- sentence(T,[the,boy,which,the,girl,likes,likes,a,watermelon],[]).
false.

?- sentence(T,[the,government,conscripts],[]).
false.

?- sentence(T,[the,girl,runs,the,boy],[]).
false.

?- prlogic([all,boys,run],L).
L = all(x17, boys(x17)=>run(x17)) .

?- prlogic([all,boys,like,all,watermelons,that,contain,some,divine,flavor],L).
L = all(x18, boys(x18)=>like(x18, all(x19, watermelons(x19)&contain(x19, exists(x20,
flavor(x20)&divine(x20)))))) .

?- prlogic([some,boy,eats,some,apple],L).
L = exists(x21, boy(x21), eats(x21, exists(x22, apple(x22)))) .

?- prlogic([some,governments,conscript,some,pacifist,people],L).
L = exists(x23, governments(x23), conscript(x23, exists(x24, people(x24)&pacifist(x24)))) .

?- prlogic([all,governments,that,conscript,some,pacifist,people,are,evil],L).
L = all(x25, governments(x25)&conscript(x25, exists(x26,
people(x26)&pacifist(x26)))=>evil(x25)) .
```

Discussion:

There are several things that could have been done with this project that I did not do because of lack of time.

I did not distinguish between "a" and "an", although I did distinguish between transitive and intransitive verbs, because the latter seemed relevant and its absence bothered me, which was not true for the former.

I did not reduce plural nouns and singular verbs to their base forms in the parse tree, because I could not figure out a graceful way to handle the change in form – everything I contemplated would have greatly complicated my terminal-handling rules. This did not particularly affect the translator; it just makes the FOL statements look a little funny.

Only sentences where the NP has a determiner which is one of "all" or "some" can be processed by the translator.

Also unfortunately missing is a user-friendly interface, so that each sentence must be typed in already atomized.

I am previously familiar with phrase-structure grammars from a syntax class, so I took my cues for handling of semantic features from that class (particularly the [+/- Obj] tag on the relative clauses/pronouns).

Appendix A (Grammar)

Grammar:
S -> NP[#], VP[#, {+/-tr,be}]
NP[#] -> Det, Noun[#]
NP[#] -> Det, Noun[#], REL[#, +/-Obj]
NP[#] -> Det, Adj, Noun[#]
NP[#] -> Det, Adj, Noun[#], REL[#, +/-Obj]
NP[pl] -> Noun[pl]
NP[pl] -> Noun[pl], REL[#, +/-Obj]
REL[#, +Obj] -> Rel, NP[#i], VP[#i, +tr, +Obj]
REL[#, -Obj] -> Rel, VP[#, Tr, -Obj]
VP[#,+be,(-Obj)] -> Verb[#, +be], Adj
VP[#,+tr,(-Obj)] -> Verb[#, +tr], NP[#i]
VP[#,-tr,(-Obj)] -> Verb[#, -tr]
VP[#,+tr,+Obj] -> Verb[#, +tr]

For NPs containing relative clauses, the number of the noun cascades iff the relative clause is [+Obj] (i.e., the head of the NP is the object of the verb in the relative clause, e.g., the boy whom the girl likes)

Appendix B (Lexicon)

Lexicon:
Nouns* = {apple, boy, girl, government, watermelon, flavor, person/people}
Determiners = {a, an, the, all, some}
Verbs* = {conscript, eat, like, run, is/are, contain}
Adjectives = {evil, divine, big, tasty, pacifist}
Relative Pronouns = {that, whom, who, which}

*Plural nouns and singular verbs, except for the irregulars, are checked using the very simple algorithm {root + 's'}