CSE 567 - Computational Linguistics Project
Natural Language Understanding System in Prolog

Nataraj, Nitin (50246850) nitinnat@buffalo.edu
Umasankar, Sunil (50249002) suniluma@buffalo.edu

May 16, 2018

1 Introduction

This project aims to implement a natural language understanding system that can be embedded into a smart refrigerator. This system acts as a means for the user to ask questions about the contents of the refrigerator. The user can also make declarative statements for which the system answers with a 'Yes' or 'No' answer. The project can be divided into three modules. For the system described above to work, it should have the knowledge about English Vocabulary and Grammar. Then, we need to implement a morphological parser which outputs the semantic representation of the queries from the user. Finally, we require a model checker that makes use of a comprehensive model, about the refrigerator world, in order to answer the input from the user. The project is implemented in Prolog Language, which is known for its usefulness in implementing Natural Language Parsing systems. We shall discuss the three modules in the chronological sequence below.

1.1 Grammar

For the grammar to work we first need to create lemmas to account for the grammatical meaning of the words from input. For example, Nouns will be mapped to 'n', Transitive Verbs will be mapped to 'tv', etc. We are also assuming, here, that words are stemmed. For example, words like drink, drinks, drank and drunk are considered to be the same (Verb).

Next, we need to implement a parser. We decided to go with Left-Corner Parser. The parser basically maps the string to a set of lexical and phrasal rules. We will discuss this in the next section.

1.2 Semantic Analysis

The Left-Corner Parser takes the input as a list of words and returns the Semantic Representation of it. It is a recursive function, that first stems each word

and tags it to a lexical statement or a phrasal rule from the list of rules we predefined. We then, obtain a First Order Logic (FOL) representation for the input.

Next, we need to use this Semantic Representation and check if its true or answer questions based on a model.

1.3 Model Checker

The model checker, takes the FOL representation from the parser and compares the data with a model we defined about the refrigerator world. The model basically contains information about the items that exist within the refrigerator world and the relationship between them. It could also contain information about the interactions of the user with the items. For example, the user could take an apple out of the refrigerator. The model could contain this piece of information and when the system is asked the question, "Did the user take an apple out?", it would reply with a 'Yes'.

2 Implementation

The implementation of each section is discussed in detail below.

2.1 Grammar

The implementation of grammar starts with the lemmas. Here are a few examples in Prolog.

```
lemma(box,n).
lemma(contain,tv).
```

Then we construct lexical items for tagging. For example:

```
lex(pn((Lemma^X)^X),Lemma):-
 lemma(Lemma,pn).
```

This checks if the word passed is proper noun or not.

Similarly, lexical items to check for other kinds of grammatical terms has been defined.

Next, to be able to parse semantic representations, we need to define rules that define the phrasal logic. English is a vast language and thus, there are a lot of rules that need to be defined if we want to parse any input. But, since our system is limited to the refrigerator world, its not a daunting task anymore.

Below is a list of rules that we defined in our system:

```
rule(vp(P^B, []), [dtv(P^X^Y^C), np((X^A)^B),vacpp, np((Y^C)^A)]).
```

This rule basically means, VP = DTV NP NP

```
rule(not(Z),[np(X^Y),be,not,pp((X^Y)^Z)]).
```

This rule applies negation. For example, it can parse, "The ham is not in the box".

Similarly,

```
rule(s(Z),[np(X^Y),be,pp((X^Y)^Z)]).
```

parses the sentence, "The ham is in the box"

```
rule(ynq(Y),[aux, np(X^Y),vp(X,[])]).
```

This rule processes sentences like: "Did the ham expire?"

```
rule(Z, [whpr((X^Y)^Z) , ynq(Y)]).
```

This rule parses, "What is in the box?"

This way, we defined a lot of rules to parse the input given by a user to a refrigerator.

2.2 Semantic Analysis

In this part of code, the parser takes the input sentence from the user and tries to parse out the FOL representation of it. We are using the Left Corner parser to implement this. The code for that is given below:

```
parse(String,S) :-
 leftcorner_recognize(S,String,[]).
 %numbervars(S,0,_),
 %write(S).


leftcorner_recognize(Cat,[Word|StringIn],StringOut) :-
 stem(Word,Stemmed), %Stem the word
 lex(WCat,Stemmed),
 complete(Cat,WCat,StringIn,StringOut).

complete(Cat,Cat,String,String).
complete(Cat,SubCat,StringIn,StringOut) :-
 rule(LHS,[SubCat|Cats]),
 matches(Cats,StringIn,String1),
 complete(Cat,LHS,String1,StringOut).

matches([],String,String).

matches([Cat|Cats],StringIn,StringOut) :-
 leftcorner_recognize(Cat,StringIn,String1),
 matches(Cats,String1,StringOut).
```

2.3 Model Checker

The model checker checks the following:

1. If input is a declarative, check if true or false.

```
modelchecker(s(X,[]),[true_in_the_model]):-
 sat([],X,_),!.
```

2. If input is a yes-no question, check if yes or no.

```
modelchecker( ynq( F ), [yes_to_question]):-
   sat([],F,_).
```

3. If input is a content question, find answer

```
modelchecker(q(F),Y):-
 sat([],exists(F),Y),!.
modelchecker(q(F),[wh_false]).
```

The model used as the refrigerator world is the following:

```
model([box1,box2,box3,box4,banana1,ham1bowl1,s,egg1,sam1,t,f,milk1,shelf1],
           [ [box, [box1,box2,box3,box4]],
             [thing,[box1,box2,box3,box4,ham1,milk1,egg1,bowl1,banana1,f,shelf1]],
             [milk,[milk1]],
             [person,[s,t,sam1]],
             [egg,[egg1]],
             [ham,[ham1]],
             [banana,[banana1]],
             [blue, [box1]],
             [green, [box2]],
             [yellow, [box3]],
             [white, [bowl1,box4]],
             [skim, [milk1]],
             [freezer, [f]],
             [bowl, [bowl1]],
             [sue, [s]],
             [tom, [t]],
             [sam, [sam1]],
             [contain,[ [box1,ham1],[box2,banana1],[f,box4] ]],
             [in,[ [ham1,box1] ]],
             [drink,[ [sam1,milk1] ]],
             [on,[ [ham3,bowl1] ]],
             [belong,[ [box1,s] ]],
             [put,[[s,box3,bowl1]]]
           ]).
```

3 Results

In this section, we would like to talk about the results we derived from the system. Two specific outputs play a vital role in the working of the system. The outputs of these are discussed below.

3.1 Parser

Below are the semantic FOL representations that the parser outputs to the Prolog console.
"A blue box contains some ham"

```
[debug] ?- parse([a,blue,box,contains,some,ham],X).
X = s(exists(_10842, and(and(box(_10842), blue(_10842)),
    exists(_11018, and(ham(_11018), contain(_10842, _11018
    ))))), []) ;
```

"A blue box contains ham"

```
[debug] ?- parse([a,blue,box,contains,ham],X).
X = s(exists(_10828, and(and(box(_10828), blue(_10828)),
    exists(_11004, and(ham(_11004), contain(_10828, _11004
    ))))), []) ;
```

"The white box that the freezer contains belongs to Sue"

```
[debug] ?-  parse([the,white,box,that,the,freezer,
    contains,belongs,to,sue],X).
X = s(the(_10886, and(and(and(box(_10886), white(_10886))
    , the(_11112, and(freezer(_11112), contain(_11112,
    _10886)))), belong(_10886, sue))), []) ;
X = s(the(_10886, and(and(and(box(_10886), the(_11112,
    and(freezer(_11112), contain(_11112, _10886)))), white
    (_10886)), belong(_10886, sue))), []) ;
```

"Is there an egg inside the blue box"

```
[debug] ?- parse([is,there,an,egg,inside,the,blue,box],X)
    .
X = ynq(exists(_10910, and(and(egg(_10910), the(_11034,
    and(and(box(_11034), blue(_11034)), inside(_10910,
    _11034)))), _10924))) ;
```

"Are there two eggs inside the blue box"

```
[debug] ?- parse([are,there,two,eggs,inside,the,blue,box
    ],X).
X = ynq(two(_10898, and(and(egg(_10898), the(_11040, and(
    and(box(_11040), blue(_11040)), inside(_10898, _11040)
    ))), _10912))) ;
```

"What does the green box contain"

```
[debug] ?- parse([what,does,the,green,box,contain],X).
X = q(_10844, and(thing(_10844), the(_10952, and(and(box(
    _10952), green(_10952)), contain(_10952, _10844))))) ;
```

"Who put every yellow box on the white bowl"

```
[debug] ?- parse([who,put,every,yellow,box,on,the,white,
    bowl],X).
X = q(_10872, and(person(_10872), forall(_10922, imp(and(
    box(_10922), yellow(_10922)), the(_10928, and(and(bowl
    (_10928), white(_10928)), put(_10872, _10922, _10928))
    ))))) ;
```

This specific example explains the wordnet implementation wherein 'ham' is mapped to its ancestor 'meat'. The word meat does not exist in the lemmas or the model.

```
[debug] ?- parse([does,a,box,contain,meat],X).
X = ynq(exists(_12128, and(box(_12128), exists(_12216,
    and(ham(_12216), contain(_12128, _12216)))))) ;
```

"Is there pork"

```
?- parse([is,there,pork],X).
X = ynq(exists(_882, and(pork(_882), _922)))
```

"Is there milk"

```
?- parse([is,there,milk],X).
X = ynq(exists(_592, and(milk(_592), _632))) ;
X = ynq(exists(_592, and(milk(_592), _632))) ;
```

"Where is the beef"

```
?- parse([where,is,the,beef],X).
X = q(_1156, and(location(_1156), the(_1230, and(beef(
    _1230), _1244)))) ;
```

"A box belongs to Sue"

```
?- parse([a,box,belongs,to,sue],X).
X = s(exists(_1162, and(box(_1162), belong(_1162, sue))),
     []) ;
```

3.2 Model Checker

For the parser output above, we have run the model checker separately and recorded the output from it.

"A blue box contains some ham"

```
[debug]  ?- modelchecker(s(exists(_10842, and(and(box(
    _10842), blue(_10842)), exists(_11018, and(ham(_11018)
    , contain(_10842, _11018))))), []),X).
X = [true_in_the_model].
```

"A blue box contains ham"

```
[debug]  ?- modelchecker( s(exists(_10828, and(and(box(
    _10828), blue(_10828)), exists(_11004, and(ham(_11004)
    , contain(_10828, _11004))))), []),X).
X = [true_in_the_model].
```

"The white box that the freezer contains belongs to Sue"

```
[debug]  ?- modelchecker( s(the(_10886, and(and(and(box(
    _10886), white(_10886)), the(_11112, and(freezer(
    _11112), contain(_11112, _10886)))), belong(_10886,
    sue))), []),X).
X = [false_in_the_model] ;
```

"Is there an egg inside the blue box"

```
[debug]  ?- modelchecker(ynq(exists(_10910, and(and(egg(
    _10910), the(_11034, and(and(box(_11034), blue(_11034)
    ), inside(_10910, _11034)))), _10924))),X).
X = [no_to_question] ;
```

"Are there two eggs inside the blue box"

```
[debug]  ?- modelchecker(ynq(two(_10898, and(and(egg(
    _10898), the(_11040, and(and(box(_11040), blue(_11040)
    ), inside(_10898, _11040)))), _10912))),X).
X = [no_to_question] ;
```

"What does the green box contain"

```
[debug]  ?- modelchecker(q(_912, and(thing(_912), the(
    _1020, and(and(box(_1020), green(_1020)), contain(
    _1020, _912))))),X).
X = [[_1020, box2], [_912, banana1]]
```

"Who put every yellow box on the white bowl"

```
[debug]  ?- modelchecker(q(_12128, and(person(_12128),
    forall(_12178, imp(and(box(_12178), yellow(_12178)),
    the(_12184, and(and(bowl(_12184), white(_12184)), put(
    _12128, _12178, _12184)))))),X).
false.
```

"Is there pork"

```
?- modelchecker(ynq(exists(_882, and(pork(_882), _922))),
    X).
X = [no_to_question] ;
```

"Is there milk"

```
?- modelchecker( ynq(exists(_592, and(milk(_592), _632)))
    ,X)
|    .
X = [no_to_question]
```

"Where is the beef"

```
?- modelchecker(q(_586, and(location(_586), the(_660, and
    (beef(_660), _674)))),X).
X = [wh_interrogative_false] ;
false.
```

"The box belongs to sue"

```
?- modelchecker(s(exists(_1162, and(box(_1162), belong(
    _1162, sue))), []),X).
X = [true_in_the_model].
```

3.3 Process

The output of process function is shown below:

```
?- process([a,blue,box,contains,some,ham]).
That is correct

true.

?- process([a,blue,box,contains,ham]).
That is correct

true.
```

```
?- process([the,white,box,that,the,freezer,contains,
    belongs,to,sue]).
That is not correct

true.

?- process([is,there,an,egg,inside,the,blue,box]).
no

true.

?- process([are,there,two,eggs,inside,the,blue,box]).
no

true.

?- process([what,does,the,green,box,contain]).
banana


true.

?- process([who,put,every,yellow,box,on,the,white,bowl]).
no

true.

?- process([does,a,box,contain,meat]).
yes

true.

?- process([is,there,pork]).
no

true.

?- process([is,there,milk]).
no

true.

?- process([a,box,belongs,to,sue]).
That is correct

true.
```