A PROTOTYPE NATURAL LANGUAGE PROCESSING SYSTEM FOR SMALL AUTONOMOUS AGENTS

A. Antonio Arroyo
Machine Intelligence Laboratory
Department of Electrical and Computer Engineering
University of Florida, Gainesville, FL 32611-6200

Abstract

We present methodology for autonomous agent communication using natural language (a subset of English) partially implemented using the Prolog programming language. The system was designed for a hypothetical agent with a fixed vocabulary. The design targets existing voice hardware devices and low-end embedded computer hardware. Elements of the grammar rules, parsing and frame-based semantic analysis are discussed.

1. Introduction

The University of Florida's Machine Intelligence Laboratory builds small autonomous agents to perform various types of research involving robotics including platform design, behavior characterization, sensor development, computer vision, swarm robotics and machine learning. Being able to communicate with a robot with a subset of natural language (English) has been an elusive research ambition. To-date our efforts have concentrated on adding text-to-speech voice synthesizers and single word speaker independent devices that allow communication with the agents using single word commands. Given the computing demands of software-based natural language understanding systems, the realization of a relatively low cost autonomous robot using English sentences as a primary means of communication has been untenable. If we are to take robotics out of the research labs and into the consumer market we must also be able to engage robot user into meaningful conversation—some other than a fixed vocabulary of single commands.

Current efforts in Natural Language Understanding Systems (NLUs) focus on methodology relying on probabilistic language models and algorithms that can be learned from the volumes and volumes of text and are simpler than augmented grammars [1,2]. Given the limitations of existing voice systems (often of the order of a few hundred words), it seems feasible to develop a prototype robotic NLU system using a very limited subset of English.

This paper describes such a preliminary effort. For the purposes of this research we will assume a small robotic platform such as MIL's TJ Robot, powered by MIL's Atmel board and equipped with a system similar to John Martiney's TRoY [3]. We'll call our robot YTIL (pronounced *Why-Till*, You Talk, I Listen). Elements of the grammar rules, parsing and frame-based semantic analysis are discussed.

Our present methodology is partially implemented using latest version of the SWI Prolog Interpreter. Prolog is a reasonably good choice for NLU implementation since Prolog was developed with Natural Language Processing tools included as part of the language [4]. While we do not yet have a Prolog interpreter available for the Atmel

processor, versions of Prolog exist for many other small systems (e.g., the ill-fated IBM PC Jr. announced in 1983).

2. Representation and Understanding

To understand means to produce a computable representation of the meaning of an utterance. We cannot use the utterances themselves because words have multiple meanings or senses. For example, *cook* has both a noun sense and a verb sense; *dish* has multiple noun senses and a verb sense; and *still* has a noun, a verb, an adjective and an adverb sense. In order to represent meaning a more precise language is needed. The representation must be precise and unambiguous giving a different rendering for each of the word meanings and senses while capturing the intuitive structure in the utterance.

We will use a phrase marker (parse trees) representation based on augmented context-free grammar rules. The master words themselves will be maintained in a data structure called the Lexicon, and rules will be included in the grammar to produce words (e.g., plurals) from basic units called morphemes (e.g., root words). A scanner module separates the utterance into lists of words that are handed to the morphological analysis rules.

For example, suppose the utterance is: "*YTIL, are you feeling good today?*„ the scanner would hand the list [*YTIL, are, you, feeling, good, today*] to the parser.

3. Morphological Analysis

While it is possible in small systems to exhaustively list all the words allowed by the system in a lexicon, the nature and limitations of existing voice recognition hardware preclude such a treatment. Most systems do recognize words with distinct phonemes but fail miserably in recognizing different versions of the same word as distinct words (i.e., eat vs. eaten vs. eating vs. eats vs. ate).

It is also highly inefficient to store in the lexicon words that may combine with affixes to produce additional related words. The classical way to address this problem is to preprocess the input sentence into a sequence of morphemes. A word may consist of a single morpheme, but often a word consists of the root form plus an affix. With this preprocessing, only the root, *eat*, would presumably be stored in the lexicon and all other forms would be obtained by combining with the suffixes –ing, -s, and –en, and one additional entry for the irregular form *ate*.

This is particularly relevant to the plural forms of most English words. Most plural forms consist of a singular root and the suffix –s, with few irregular forms (e.g., *man* and *men*, not "*mans*„).

Sample rules to handle singular and plural forms for a few nouns are as follows:

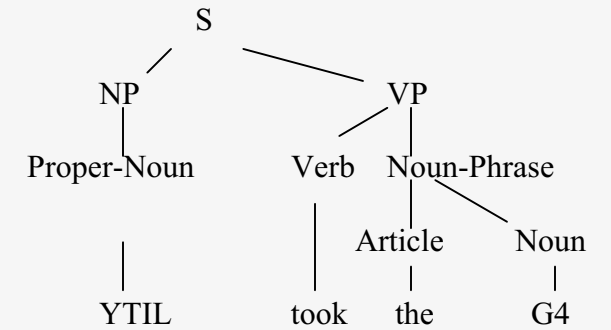
```
noun(S,noun(N)) –> [N],{is_noun(N,S)}.
noun(plural,noun(RootN)) –> [N],
      {name(N,Plname),
       append(Singname,"s",Plname),
       name(RootN,Singname),
       is_noun(RootN,singular)}.
is_noun(men,plural).
is_noun(man,singular).
is_noun(robot,singular).
is_noun(gator,singular).
is_noun(best,_).
```

```
?- noun(A,P,[robot],[]).
A = singular
P = noun(robot)
Yes

?- noun(A,P,[robots],[]).
A = plural
P = noun(robot)
Yes
```

4. Syntactical Analysis

The syntactical analysis phase produces phrase markers (parse trees) to represent sentence structure. In particular, the representation selected must make explicit: (1) the way words in the sentence relate to one-another, (2) how words are grouped into phrases, (3) what words modify other words, (4) what words are of central importance, and if possible (5) the types of relationships that exist between phrases. A tree is easily represented in Prolog by a functor and its arguments. For example the sentence "*YTIL took the G4*,, should yield the parse tree:



The syntax rules consist of augmented context-free grammar (CFG) rules. Augmentation allows context-free grammar rules to handle such things as auxiliary verb forms, subject-verb agreement, relative clauses, etc. Augmentation is modeled in Prolog with additional functor arguments. For example, in English, there must be agreement in person and number between the subject noun phrase and the verb phrase in a given sentence. This is referred to as *subject-verb agreement* in the literature [1,2]. The following CFG rules handle the sample sentence.

```
sentence-->np,vp.
np-->deter,noun|noun.
vp-->verb,np|verb.
deter-->[the]|[a].
noun-->[ytil]|[robot]|[best]|[g4]|[computer].
verb-->[take]|[took]|[takes].
```

```
?- phrase(sentence,[ytil,took,the,g4]).
Yes
```

Unfortunately, the CFG rules do not handle subject-verb agreement in a declarative utterance such as:

```
?- phrase(sentence,[ytil,take,the,g4]).
Yes
```

The answer should have been No, since this is an ungrammatical utterance (the subject NP is in 3rd person singular, therefore the verb must also be in its 3rd person singular form, *takes*).

Augmentation allows us to fix this problem. We add an additional argument(s) to each of the CFG rules to handle person and number as follows:

```
sentence(Agr)-->np(Agr),vp(Agr).
np(Agr)-->deter(Agr),noun(Agr)|noun(Agr).
vp(Agr)-->verb(Agr),np(_)|verb(Agr).
deter(_)-->[the]|[a].
noun(s3)-->[ytil]|[robot]|[best]|[g4]|[computer].
verb(s3)-->[takes].
verb(s1)-->[take].
verb(s2)-->[take].
verb(p1)-->[take].
verb(p2)-->[take].
verb(p3)-->[take].
verb(_)-->[took].
```

```
?- phrase(sentence,[ytil,took,the,g4]).
Agr=s3
Yes
?- phrase(sentence,[ytil,take,the,g4]).
No
?- phrase(sentence,[ytil,takes,the,g4]).
Agr=s3
Yes
```

Additional arguments may also be used to automatically generate the parse tree. In the version below, we add an extra argument to each predicate, saying how the tree for a whole phrase is constructed from the trees of the various sub-phrases. The rule

s(Agr,*sentence*(NP,VP))-->*np*(Agr,NP),*vp*(Agr,VP).

indicates that in order to parse a sentence *S* one must find a noun phrase (*NP*) followed by a verb phrase (*VP*), and then combine the parse trees of these two constituents, using the

functor *sentence(.,.)* to make the tree for the sentence.

```
s(Agr,sentence(NP,VP))-->np(Agr,NP),vp(Agr,VP).
np(Agr,np(DET,N))-->deter(Agr,DET),noun(Agr,N).
np(Agr,np(N))-->noun(Agr,N).
vp(Agr,vp(V,NP))-->verb(Agr,V),np(_,NP).
vp(Agr,vp(V))-->verb(Agr,V).
deter(_,deter(the))-->[the].
deter(_,deter(a))-->[a].
noun(s3,noun(ytil))-->[ytil].
noun(s3,noun(robot))-->[robot].
noun(s3,noun(best))-->[best].
noun(s3,noun(g4))-->[g4].
noun(s3,noun(computer))-->[computer].
verb(s3,verb(takes))-->[takes].
verb(s1,verb(take))-->[take].
verb(s2,verb(take))-->[take].
verb(p1,verb(take))-->[take].
verb(p2,verb(take))-->[take].
verb(p3,verb(take))-->[take].
verb(_,verb(took))-->[took].
```

```
?- phrase(sentence(Agr,Tree),[ytil,took,the,g4]).
Agr = s3
Tree = sentence(np(noun(ytil)), vp(verb(took),
                np(deter(the), noun(g4))))
Yes
```

5. Semantic Analysis

Semantic analysis uses so-called *thematic-role frames* [6] to capture detailed knowledge about how acts happen. *Thematic-role frames* describe the action conveyed by the verbs and nouns appearing in typical declarative sentences. Much of what happens in the world involves actions, and objects undergoing change. It is natural, therefore, that many sentences in human language amount to descriptions that specify actions, identify the object undergoing change, and indicate which other objects are involved in the change.

In linguistic terms, verbs often specify actions, and noun phrases identify the objects that participate in the action. Each noun phrase's *thematic role* specifies how the object participates in the action. The sample sentence "*YTIL took the G4,*„ for example, carries information about how YTIL and an Apple G4 laptop relate to the verb *took*. A procedure that understands such a sentence must discover that YTIL is the *agent* because it performs the action of taking, and that the G4 is the *thematic object* because it is the object taken.

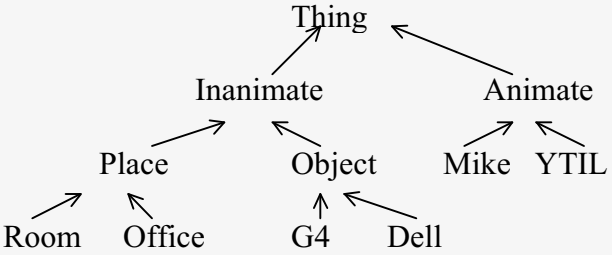
The number of thematic roles embraced by various theories varies considerably. Some people use half-dozen thematic roles. Others use three or four times as many. The exact number does not matter as much, as long as it is great enough to expose natural constraints on how verbs and thematic role instances form sentences in the domain of interest.

A sample thematic role frame is given by:

Thematic-role frame

Verb		Source	
Agent		Destination	
Coagent		Time	
Beneficiary		Location	
Thematic object		Duration	
Instrument		Trajectory	

If we further restrict the world view of the YTIL robot to, say, rooms in the Machine Intelligence Lab, we can arrive at various meanings for each of the verbs in YTIL's vocabulary (perhaps a better criteria is to look at the set of all possible behaviors of the robot or at the set of actions that are feasible. Nevertheless, we assume this is finite and small).



What meanings can the verb *take* exhibit? We will consider the following:

- *Take1* means transport. Either a source or destination or both should appear.
- *Take2* means swindle. The source and destination roles are absent when this meaning is intended. Only animates can be swindled.
- *Take3* means to consume or ingest something. The beneficiary is the same as the agent. There are no consumables in our sample world.
- *Take4* means to steal. Animates are not stolen.
- *Take5* means to initiate and execute a social event with another person. The particle *out* is always used.
- *Take6* means to remove. The particle *out* is always used. People cannot be removed.
- *Take7* means to assume control. The particle *over* signals this meaning.
- *Take8* means to remove from the body or to depart. The particle *off* is always used.

Rules can be added to fill in the triggered thematic-roles with the instantiated variables from the parse tree. Using *Take1* and *Take4* we obtain the rules:

```
meaning(Agent,Object)-->steal(Agent,Object)|
                        transport(Agent,Object).

transport(Agent,Object)-->
    sentence(_,sentence(np(Agent),vp(verb(took),Object))),
                    {write(transport)}.
steal(Agent,Object)-->
  sentence(s3,sentence(np(Agent),vp(verb(takes),Object))),
                    {write(steal)}.
steal(Agent,Object)-->
    sentence(_,sentence(np(Agent),vp(verb(took),Object))),
                    {write(steal)}.
```

```
?- meaning(Agent,Object,[ytil,took,the,g4],[]).
steal

Agent = noun(ytil)
Object = np(deter(the), noun(g4)) ;
transport

Agent = noun(ytil)
Object = np(deter(the), noun(g4)) ;

No
?- phrase(meaning(Agent,Object),[ytil,took,the,g4]).
steal

Agent = noun(ytil)
Object = np(deter(the), noun(g4)) ;
transport

Agent = noun(ytil)
Object = np(deter(the), noun(g4))

Yes
?- phrase(meaning(Agent,Object),[ytil,takes,the,g4]).
steal

Agent = noun(ytil)
Object = np(deter(the), noun(g4)) ;

No
?- phrase(meaning(Agent,Object),[ytil,take,the,g4]).

No
```

6. Conclusion

This research demonstrates that a prototype natural language understanding system is feasible for a small robot running the Prolog programming language. Many simplifying assumptions were applied for the sake of brevity in this paper. For example, Imperative sentences were not parsed, even though we would expect a lot of the dialog with a robot to consist of such command forms. Imperative sentences can be handled with rules such as:

```
sentence(Type,X,sentence(NP,VP)) -->
      imperative(Type,X),
      noun_phrase(X,NP),verb_phrase(X,VP).
imperative(imperative,_),[you] --> [].
imperative(declarative,_) --> [].
```

Also, prepositional phrases and particles were not included in the examples. Prepositions are important because they help discover which

thematic-role to trigger. Particles are important because they distinguish among contending verb meanings. Particles are usually parsed as "parts of the verb.,, For example, *take out*, is handled as the composite verb *take out* and not as the verb *take* with preposition *out*.

Verb phrases consist of auxiliary verbs sequences followed by a main verb and we have both active and passive forms. Rules for handling these are also straightforward and were omitted from the paper for the sake of clarity.

Finally, we should also be prepared to handle questions (both yes-no questions and wh-questions), which involve constituent movement and gaps. Rules for handling are similar to the rules for Imperative sentences above. Relative clauses are also handled in a manner similar to wh-questions. Inclusion of these rules is beyond the scope of this preliminary paper. Perhaps YTIL is not as far away from reality as we had expected…

7. References

[1] *Artificial Intelligence: A Modern Approach*, Stuart Russell and Peter Norvig, Prentice-Hall, NY, 2003

[2] *Generalized Phrase Structure Grammar*, Gazdar G., Klein E., Pullum, G., and Sag I., Oxford: Basil Blackwell, 1985

[3] (T)ango (R)omeo (o)scar (Y)ankee: "A Talking, Representation of Yourself,, John A. Martiney, and A. Antonio Arroyo, FCRAR 2003.

[4] *Programming in Prolog*, Clocksin W., and Mellish C., Springer-Verlag, NY, 1994

[5] *Nattural Language Understanding*, James Allen, Benjamin Cummins, CA, 1995

[6] *Artificial Intelligence*, Patrick Henry Winston, Addison-Wesley, NY, 1992