KIT108 Artificial Intelligence – Prolog Workshop

Natural Language Processing

Parsing

This tutorial has been developed with reference to the following sources:
"Artificial Intelligence: Structures and Strategies for Complex Problem Solving" by G.F. Luger and W.A. Stubblefield, Benjamin Cummings, 1997.

NATURAL LANGUAGE UNDERSTANDING

The ability to understand natural language is a fundamental aspect of human intelligence and so one of the earliest tasks AI researchers set themselves was to attempt to enable computers to communicate with humans using natural language. Significant funds were also provided for this work by the US military in the hope that it would lead to the development of AI programs capable of translating written text from one natural language to another (natural language translation), in particular from Russian to English. Natural language translation proved to be a much more difficult problem than originally envisaged and the military withdrew funding when it was clear that successful natural language translations systems were far beyond the state of the art at the time.

More recent efforts in natural language understanding have been much less ambitious and have concentrated on improving computer-human interfaces by allowing humans to converse with computers in natural language, but within restricted domains of discourse.

The reason for the difficulty in natural language understanding is that it actually involves several more steps than was initially realised. It is reasonably clear that such a system must identify the sounds being made by the communicating entity (**phonology**), put them together into appropriate words (**morphology**), determine how these words have been combined into sentence by parsing the sentence into individual parts of speech (**syntax**) and then determine what the sentence means (**semantics**). However, closer investigation reveals that a very important step is being carried out at the very beginning of the process; that of analysing the rythym and intonation of the sentence (**prosody**). This step is very difficult to formalize and so has often been ignored in natural language systems. It was also revealed that the process of determining the meaning of a sentence is complicated by common use of ambiguous terms in natural language. Such ambiguities are usually resolved because both speaker and listener have **common accepted knowledge** within the domain of discourse. Without such shared knowledge, effective communication is virtually impossible (imagine a cricket fan describing details of a particular game to someone who knows nothing about cricket). This shared knowledge-base also contains enormous amounts of common-sense knowledge which is being used, subconsciously, to assist in the communication. Even more subtle than this is a common understanding by both parties of the purpose of the communication (ie what the speaker intends to happen as a result of this communication with the listener). This level of analysis is called **pragmatic** analysis and also has received little attention in natural language understanding systems to date.

The need for an extensive shared knowledge-base between speaker and listener means that questions of knowledge representation are as important for natural language understanding as they are for expert systems. In fact, one of the major knowledge representation schemes used in expert systems today, the semantic network, was originally invented as part of early natural language research.

A VERY SIMPLE CONTEXT-FREE GRAMMAR

Although, as we have stated before, parsing a natural language sentence is only one step in the process of natural language understanding, it is the most well understood part of the understanding process. The reason for this is that the rules needed to do it are fairly clear-cut and therefore the process can be readily formalised. Also, a great deal of work has been done on parsing by computer scientists because exactly the same process needs to be carried out on computer language "statements" by any compiler or interpreter. The problems of knowledge representation and retrieval which are very important in natural language understanding are common to other AI sub-fields and have been dealt with elsewhere, so for this tutorial let's concentrate on the parsing process itself.

To illustrate how it is possible to automate the parsing process let as start with a subset of English grammar. In common with much AI research we are going to try out our ideas on a very simple grammar first. Consider a simple domain of discourse called the Dog's World. The world consists of men and dogs. They can bite each other or they can like each other. This grammar consists of only five rules (at least at first).

RULE 1	A sentence is a noun phrase followed by a verb phrase.
RULE 2	A noun phrase can be just a noun.
RULE 3	A noun phrase can be an article followed by a noun.
RULE 4	A verb phrase can be just a verb.
RULE 5	A verb phrase can be a verb followed by a noun phrase.

In addition to these grammar rules, a parser needs a dictionary of words in the language. These words are the terminals of the grammar and are specified in the dictionary along with information about what part of speech they are. The dictionary looks like this:

a	An article.
the	An article.
man	A noun.
dog	A noun.
likes	A verb.
bites	A verb.

A PARSER FOR THIS GRAMMAR

A parser can be built for this grammar by implementing the grammar rules, and the dictionary as **rewrite** (or **production**) **rules**. Interestingly, production rules, like semantic networks, were originally invented for this application well before they were used for expert systems knowledge representation.

Consider the first rewrite rule below. It says that if you had the symbol **sentence** written down, you could replace it by the symbol **noun-phrase** followed by the symbol **verb-phrase**. Alternatively, if you had the symbol **noun-phrase**, followed by the symbol **verb-phrase**, you could write down the symbol **sentence** instead.

Some of the rewrite rules for the Dog's World grammar are shown below. Fill in the others yourself by referring to the grammar and dictionary given previously.

RULE 1	sentence <-> noun-phrase verb-phrase
RULE 2	noun-phrase <-> noun
RULE 3	noun-phrase <-> article noun
RULE 4	.....
RULE 5	.....
RULE 6	article <-> a
RULE 7	article <-> the
RULE 8	noun <-> man
RULE 9	.....
RULE 10	.....
RULE 11	.....

DATA-DRIVEN PARSING

A set of production rules such as this is then able to determine whether a sentence fed to it is grammatically correct and can also break the sentence up into its individual parts of speech (ie parse the sentence). This can be done using a data-driven (or bottom-up) approach. Fill in the missing steps in the trace below yourself and also fill in the blanks in the parse tree as you go along.

step 1 Feed the sentence **the man likes the dog** into the production system.

step 2 The first rule to match will be rule 7, so it will fire, giving the sentence:

article **man likes the dog**

The association between ***article*** and **the** can also be used to start building up a parse tree (ie an and-or graph which will eventually represent the fully parsed sentence) as illustrated below.

step 3 The next rule to match will be rule 8, and when it fires the sentence will be transformed to:

article noun **likes the dog**

Again the association between ***noun*** and **man** can be added to the parse tree.

step 4 .....

.....

step 5 .....

.....

step 6 .....

.....

step 7 .....

.....

step 8 .....

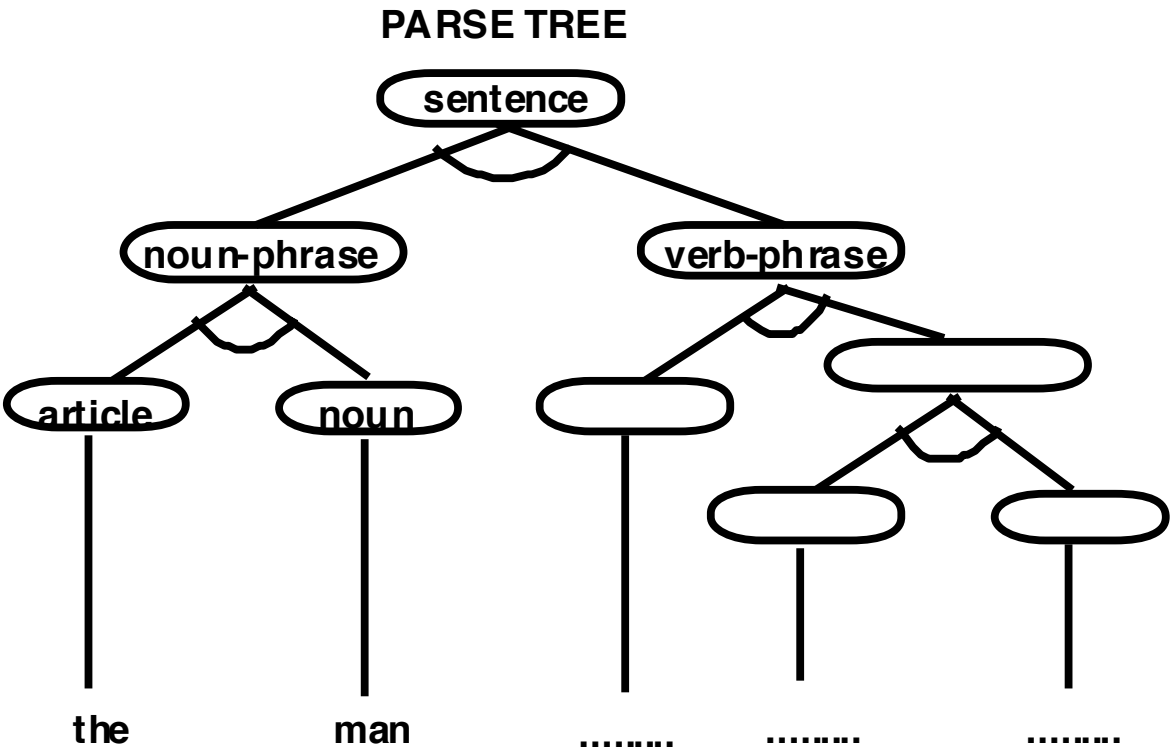
.....

step 9 At this stage the sentence which was originally fed to the system should have been transformed to the form:

noun-phrase verb-phase

If this has happened then Rule 1 can fire and transform it to the symbol:

sentence



If the input sentence can be transformed to the single symbol **sentence** by the production rules in the parser, then the sentence is legal in that grammar. Also, the parse tree which has been created in the process represents the linguistic structure of the sentence and its construction has been a significant step in the natural language understanding process.

GOAL-DRIVEN PARSING

Parsers can also be used to generate legal sentences in a grammar, by using a goal-centred (or top-down) approach. In this case the process begins with the symbol **sentence** and fires the production rules from left to right, finishing up with some legal sentence in the grammar. Fill in the steps below, in order to generate a legal sentence in the Dog's World grammar. The sentence you generate will, of course, depend on the choices you make when firing the production rules, but no matter what the actual sentence is, if the process has been followed accurately, it will be a legal sentence.

step 1 Feed the symbol ***sentence*** into the production system.

step 2 The first rule to match will be rule 1, so it will fire replacing ***sentence*** by:
noun-phrase verb-phrase

Now the symbol ***noun-phrase*** can be used to fire either rule 2 or rule 3. Similarly, the symbol ***verb-phrase*** can fire either of two production rules. Choose one of the noun-phrase rules to fire; then choose one of the verb-phrase rules to fire.

step 3 .....

.....

step 4 .....

.....

step 5 .....

.....

step 6 .....

.....

step 7 .....

.....

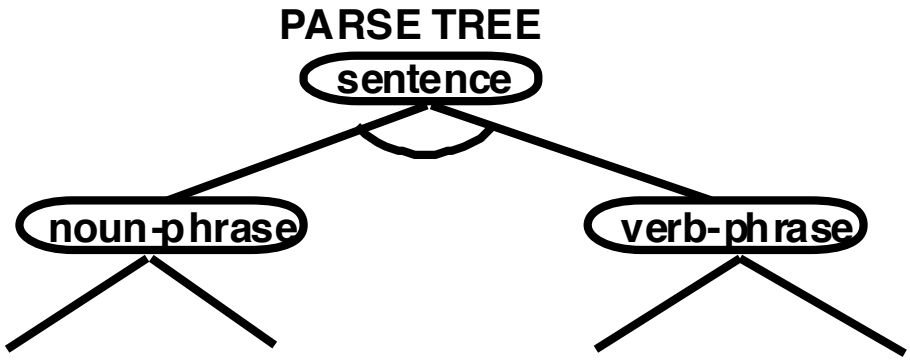
step 8 .....

.....

step 9 By now, hopefully, you have come up with a legal sentence in the context-free dog's world grammar.

.....

Now that you have generated a sentence you can also create a parse tree to represent that sentence. Complete the parse tree below so that is represents the sentence you have just created.



CREATING A CONTEXT-FREE PARSER

Let us take the production rules described above and transcribe them into Prolog rules. Doing this will enable us to create a Prolog program capable of parsing all legal sentences in the Dog's World grammar. We'll call it the Canine Parser (why it is called a context-free parser will be explained later). Each Prolog rule will have two parameters. The first is a list of words. The rule attempts to determine whether some initial part of the list is a legal sentence. The remainder of the list, which is not parsed, will match the second parameter.

Rule 1 in the Canine Parser, can be transcribed to the following Prolog rule:

sentence(Start, End) :- nounphrase(Start, Rest), verbphrase(Rest, End).

If this sentence rule succeeds, the second parameter (ie End in this case) will contain what remains of the sentence after a noun-phrase and a verb-phrase have been parsed out of it. The noun-phrase can be parsed using one of two possible rules:

nounphrase([First | End], End) :- noun(First).
nounphrase([First, Second | End], End) :- article(First), noun(Second).

The parser also requires some facts to be presented to it, indicating what the terminals in the grammar are, and which parts of speech each one represents. For example the two articles and the two nouns in the grammar can be represented by the facts:

article(a).
article(the).
noun(dog).
noun(man).

Let us analyse carefully how these rules and facts can be used in parsing a sentence such as "The dog likes the man". The parse process is initiated by typing in the Prolog query:

? sentence([the, dog, likes, the, man], End).

This will match to the head of **rule 1**, with **Start** bound to the list **[the, dog, likes, the, man]** and **End** remaining unbound at this stage. The tail of rule 1 will then ask the first of two queries:

? nounphrase([the, dog, likes, the, man], Rest).

This will match to the head of **rule 2**, with the list **[Noun|End]** bound to the list **[the, dog, likes, the, man]** and **Rest** remaining unbound for the present. The **|** symbol divides a list into its head (first element) and its tail (rest of the list) so this means that the binding here has in fact matched **First** to **the** and **End** to the list **[dog, likes, the, man]**. **Rule 2** will now ask the query :

? noun(the).

Since there is no such fact, **rule 2** will **fail** and **rule 3** will be tested. In this case the list **[First, Second | Rest]** will be matched to the list **[the, dog, likes, the, man]**, meaning that

First will bind to the value **the** and **Second** will be bound to the value **dog**, leaving the rest of the sentence **[likes, the, man]** to bind to **Rest**. Rule 3 will now ask two queries:

? noun(dog).
? article(the).

Since these two facts do exist in the parser, this means that **rule 3 succeeds** with **End** now bound to the list **[likes, the, man]**. This means that the first of the two queries initiated by Rule 1 has succeeded and that the variable **Rest**, which was bound to the variable **End** when the query was initiated, has now been bound, via **End** to the list **[likes, the, man]**. Effectively, the first query by Rule 1 has parsed the first two words in the sentence and then handed the rest of the sentence over to the second query, whose job it is to try to complete the parsing process.

1. Type **rules 1**, **2** and **3** and **facts 1**, **2**, **3**, and **4** into a Prolog text file called "**confreepars.pl**".

Carefully analyse the process used by the two nounphrase rules described above to parse the beginning of the sentence. Then devise two **verbphrase** rules which will parse the remainder of the sentence. Also create two more facts to include the verbs **likes** and **bites** in the parser.

2. Write these new rules and facts below and then type them into "**confreepars.pl**".

rule 4 .....

.....

rule 5 .....

.....

fact 5 .....

fact 6 .....

In order to simplify the use of the Canine Parser, an initial prolog rule, called the **utterance rule**, can be placed before all the other rules in the parser. This rule accepts the sentence (in list form) to be parsed as a single parameter and passes it to the sentence rule. It also initialises the second parameter of sentence to [], ensuring that the overall parsing process will only succeed if nothing is left over at the end.

3. Type the following rule into the **confreepars** file, before all the other rules in the parser:

utterance(X) :- sentence(X, []).

USING the PARSER.

Now that we have created the Canine Parser, let us see what we can do with it:

4. Test the following sentence to see if it is grammatically correct in the Dog's World grammar by typing in:

?- utterance([the, man, bites, the, dog]).

What answer did you get? .....

5. Now test to see if these sentences are grammatically correct:

the man likes the.	The answer is .....
the men like dogs.	The answer is .....
a dog bites the man.	The answer is .....

6. It is also possible for the parser to complete partial sentences. To illustrate this type the following in:

?- utterance([the, man, likes, the, X]).

Write down the two possible conclusions that the Canine Parser comes up with to complete this sentence.

....

7. The parser can also be used to generate all possible grammatical sentences for this limited dictionary and set of grammar rules. To do this type in the following:

?- utterance(X).

Write down the first two sentences that the parser comes up with:

.....

.....

CREATING A CONTEXT-SENSITIVE PARSER.

A parser will, of course, be much more useful if it is able to ensure other compatibilities between parts of speech when testing to see if a sentence is grammatically correct. For example, in English and many other natural languages, nouns and verbs must agree in number. If a particular noun is singular (ie refers to a single item) then the verb referring to it must also be of the singular form. The means that the sentences "The man likes the dog." and "The men like the dog." are both grammatically correct, but "The men likes the dog." is not. Therefore, the correctness of a particular terminal in this grammar is dependent on its context within the sentence. A parser which is able to use context like this to verify the correctness or otherwise of a sentence is called a context-sensitive parser.

8. To create a context sensitive parser, capable of enforcing noun-verb agreement, it is necessary to change each terminal fact so that it not only specifies the parts of speech, but also specifies the number of each part of speech being used. Make a copy of the "**confreepars.pl**" file and call it "**consentpars.pl**". Modify the first few terminal facts, in the "**consentpars.pl**" file so they look like this:

```
article(a, singular).
article(the, singular).
article(the, plural).
```

9. Now either modify the existing terminal facts, or insert new terminal facts, so that the part of speech and the number is specified for the following terminals:

these man men dog dogs likes like bites bite

10. It is also necessary to modify the sentence rule so that it ensures that the number of the noun-phrase matches the number of the verb-phrase. Do this by retyping it thus:

```
sentence(Start, End) :- nounphrase(Start, Rest, Number),
                        verbphrase(Rest, End, Number).
```

11. At the next level down, the noun-phrase rules must be modified to ensure that the article and the noun agree in number. This can be done, as above, by inserting a third parameter into all clauses in the rule. Retype the noun-phrase rules including this new parameter thus :-

```
nounphrase([First | End], End, Number) :- noun(First, Number).
nounphrase([First, Second | End], End, Number) :-
                                        article(First, Number),
                                        noun(Second, Number).
```

Now, when a query such as **noun(dogs, Number)** is initiated by the second nounphrase rule, it will match the fact **noun(dogs, plural)** thus binding the variable **Number** to the value **plural**. Therefore the rule will only succeed if the second query contains an article which is plural, such as **article(these, plural)**. This enforces article-noun number agreement.

12. Using the noun-phrase rules as a guide, modify the two verb-phrase rules so that they will enforce noun-verb agreement. Note that it is essential that the verb in the verb-phrase agrees in number with the noun in the noun-phrase at the beginning of the sentence. For example "The men likes the dog." would need to be rejected.

However, where the verb-phase contains another noun-phrase, it is not necessary for this second noun-phrase to agree in number with either the first noun-phrase or the verb. For example "The man likes these dogs." is valid, even though the first noun-phrase and the verb are singular while the second noun-phrase is plural. Write down the two modified verb-phrase rules below, then type them into "**confreepars.pl**".

(Remember that Prolog allows the **underline** symbol _ to be placed in an argument in situations where we **don't care** what value appears in that argument when the matching process is being carried out.)

.....

.....

.....

TESTING the CONTEXT-SENSITIVE PARSER.

Now that we have created a context-sensitive version of the Canine Parser, let us see what we can do with it:

13. Type in appropriate Prolog queries to test the following sentences for grammatical correctness:

the man likes the dogs.	The answer is .....
the men likes dogs.	The answer is .....
a dog bites the man.	The answer is .....
a dog bites these man.	The answer is .....

14. Then get the parser to complete partial sentences, this time keeping noun-verb agreement in mind. To illustrate this, type in the following:

? utterance([these, men, like | X]).

Write down three possible endings that the Canine Parser comes up with to complete this sentence.

....

15. Get the context-sensitive parser to generate all possible grammatical sentences by typing in:

? utterance(X).

Write down the first two sentences that the parser comes up with:

.....

.....

CANINE PARSER EXERCISE SHEET

STUDENT NAME .....

Question 1.
Make a copy of the "**consentpars.pl**" file and call it "**adjectpars.pl**". Add Prolog rules to **adjectpars** to enable it to accept sentences with one adjective (or none) in front of each noun. For example, the grammar should include as valid sentences, the following sentences:

The large dog bites the man.
The hairy dog likes the bald man.
The small dog likes the hairy man.
Dogs bite noisy men.

Add Prolog facts into the parser to include the following adjectives:

large, small, hairy, bald, noisy, quiet.

Write down the new facts and rules below (a computer printout can be attached here):

.....

.....

.....

.....

.....

.....

.....

.....

.....

Question 2.
Test the **adjectpars** on the following sentences. In each case indicate whether the parser accepts the sentence as valid or invalid:

A large dog bites a small dog. .....

These dogs bite savagely. .....

Men and dogs like each other. .....

The men like quiet dogs. .....

Question 3.
Make a copy of the "**adjectpars.pl**" file and call it "**conjunctpars.pl**". Add some more Prolog rules which will enable the "**conjunctpars.pl**" to accept complex sentences (ie. sentences which can contain one or more sub-sentences connected to each other with one of three possible conjunctions; **and**, **or**, or **but**). The parser would now need to be able to accept the following sentences as valid:

The large dog bites the man and noisy dogs like noisy dogs.
Small dogs bite noisy men but large men like noisy dogs.

The best way to do this is to create a new predicate which defines a complex sentence, and write a recursive rule which says that a complex sentence can be a sentence followed a conjunction followed by another complex sentence. Don't forget to also write a rule which can terminate this recursion

(Hint: look back at the Travel Navigator tutorial for a reminder on how to write recursive rules such as these). Show the new Prolog rules you have included in the Canine Parser in order to provide this extra facility below (a computer printout can be attached here):

.....

.....

.....

.....

.....

.....

.....

.....

.....

Question 4.
Now test **conjunctpars**. on the following sentences. In each case indicate whether the parser classifies the sentence as valid or invalid:

The large dog bites the noisy man but small dogs like large men and large dogs like small men. .....

Noisy dogs bite small dogs but large dogs bite small noisy dogs. .....

Dogs like quiet men and men like quiet dogs but large dogs like large men and small dogs like small men. .....