Chapter 2

Natural Language Understanding

We shall consider a number of systems which attempt to understand English (or rather natural language) as input. It is only really fairly recently that we have had programs which can cope with very complex sentences and they tend to be very large and slow. More success can be had once the domain of discourse is limited. All the early programs were of this form.

2.1 Eliza

As we said earlier, this was an attempt to play the role of a psychiatrist and works by pure syntactic trickery — although having said this, it never the less does impress people as you will see. Eliza has a list of templates against which it tries to pattern match the input question together with a list of ranked keywords which it uses if no matching can be obtained. The templates are of one of the following forms:

1. `identical character for character`
2. `<string> %`
3. `<string1> % <string2>`

The strings are lists of characters which must be matched exactly and the % is then then matched against the substring which remains. This value of % will then be used in the answer to give a more reasonable reply — thus giving the impression that the machine is really understanding the input. Of course it is only in case 1) where it is known exactly what the user typed in. Let us consider a template match of the form 2) e.g `do you %` with possible responses :-

a) `what difference does it make whether or not I do %`

b) `maybe I % and maybe I don't`

c) `yes`

If the user typed in:-

```
do you believe in god?
```

Then a possible answer, by invoking response b) would be:-

```
maybe I do believe in god and maybe I don't.
```

Hashing techniques are used for fast retrieval and template matching. For example in the implementation on unix a match is made on the first 2 characters initially. If no template match can be made, then a search is made using keywords in the input sentence. The keywords are ranked - the highest ranking word is the one which is fired. e.g. A list of possible responses for the keyword `difficult` might be:

a) `tell me about your difficulties`

b) `what do you mean by difficult`

If there are no keywords in the sentence which match with the keywords in the database then some standard non-committal remark can be made such as

a) `let's change the subject`

b) `go on`

An alternative approach is to choose at random from one of the previous used keywords — a record of say the last 4 keywords is kept. So here the system is making use of a limited history of the session. This is the basic technique used by Eliza except when a template match of the forms 2) or 3) is fired some post-processing must be performed to produce a grammatically correct answer. e.g. If the user mentions the personal pronoun I in the input then this cannot be used in the substituted part as it stands. Essentially this must be changed to `you` but it is not quite that simple. e.g.Suppose there were a template

```
because %
```

with a response template

```
%?  that's incredible!
```

Then on the input

```
because I am clever.
```

a straight-forward substitution would result in

```
I am clever?  that's incredible!
```

and then substituting for I would give

```
you am clever?  that's incredible!
```

Of course this is still not correct and has to go through a further transformation changing `you am` to `you are`. These two phases are called fix and grammar. fix: does the following changes

you	→	me
I	→	you
me	→	you
that you	→	that I

plus some others.
grammar: here simple grammatical errors are corrected. These include

you am	→	you are
me are	→	I am

plus other similar changes. Note: It is important that fix is performed before grammar. example:

```
do you think I am clever
          ⇓
maybe I think I am clever and maybe I don't.
          ⇓ fix
maybe I think you am clever and maybe I don't.
          ⇓ grammar
maybe I think you are clever and maybe I don't.
```

This cannot catch every grammatical error and it is fairly easy for it to give an ungrammatical reply. For example the current unix implementation doesn't take care of `what you` in fix, and so from

```
do you believe in what you do
```

would result in

```
maybe I do believe in what me do or maybe I don't.
```

Of course this would be easy to change fix to accommodate this but there are plenty more examples which can fool the program. The only real knowledge we might say the program has is that contained in fix and grammar — there is no deductive power for instance. In no way can this be truly thought to be intelligent (although the author of doctor tried to carry the above ideas to a true consultation program!). To improve it would require more and more fixes to deal with new cases that arose as there is no real knowledge of syntax or semantics of English.

2.1.1 Prolog Implementation

Refer to the listing to see how such a program could be implemented in Prolog.

2.2 Beyond ELIZA — PARRY

Colby et. al. wrote a program called PARRY based on ideas in ELIZA which had the following properties:

- Includes elements of emotion and other complex features.
- It was to mimic a paranoid patient. Hence users could expect very strange conversations.
- It believed it was being persecuted!

The underlying mechanism of PARRY was:

- A pattern matcher like ELIZA.
- Some knowledge of grammar — such as recognition of some verbs, nouns etc.
- It had a rating of self-esteem which could be changed — so could be more/less paranoid.
- The interviewer could also be rated as friendly, hostile, dominating etc.

2.2.1 Example — strongly paranoid model

I: Why are you in hospital?
`P: I shouldn't be here.`
I: Who brought you here?
`P: The Police.`
I: What trouble did you have with the Police?
`P: Cops don't do their job`
I: What do they do instead?
`P: Cops arrest the wrong people.`
I: They arrest the right ones sometimes.
`P: The law should get those Italian crooks`
I: Are you Italian?
`P: My background is British but I was born in this country ...`
and so on ...
This conversation certainly looks paranoid!

2.2.2 Tests on PARRY

In an real experiment, 33 psychiatrists were asked to rate the degree of paranoia as illustrated in typed transcripts. Three of them were with human patients and two generated by PARRY. The interviewers of the transcripts were themselves psychiatrists and interacted via a teletype.

There was general agreement amongst the psychiatrists on the rating of the PARRY interviews — as reliably as for the human patients ratings. None stated that they thought there was anything unusual.

Later the psychiatrists were told that some of the transcripts were machine generated, but they were, on the whole, unable to recognise which ones were from PARRY. Of course this whole situation is a bit odd anyway, as human paranoids behave in strange ways. The experiment may be thought to say more about about psychiatrists!

See the slide to find out how ELIZA might get on with PARRY — the final experiment!

2.3 Student (Bobrow MIT '64)

Algebra problems were presented to the system in more or less standard English as for instance they appear in school text books. These were then translated into a system of (linear) equations and then solved by the usual numerical techniques. Because of the stylised form of the English the concepts which are going to appear as variables can be recognised by the use of keywords. e.g. using number or some form of unit after a noun. The system was aware of simple numerical facts such as 3 feet = 1 yard, 8 pints = 1 gallon etc.

2.3.1 examples

1. *The number of customers Fred gets is twice the square of* 20% *of the number of adverts he runs. The number of adverts is 45, what is the number of customers?* The system would translate this internally as the simultaneous equations:

$$\begin{aligned} f &= 2 * (a/5)^2 \\ a &= 45 \end{aligned}$$

   and then this is solved for f giving $f = 2 * 9^2 = 162$.

2. *Mary is twice the age Anne was when Mary was as old as Anne is now. If Mary is 24 how old is Anne?*

   This now becomes:

$$\begin{aligned} M &= 2 * (A - K) \\ M - K &= A \\ M &= 24 \end{aligned}$$

   and solving for A yields A = 18.

Note: Again this is a very restricted domain of discourse.

2.3.2 Keyword Replacement

In systems like Student which rely on replacing certain keywords by variables in an equation we use the following translation rules:

- the keyword is is replaced by =
- the noun phrases in which a number followed by a unit, e.g. 5cm, are replaced by the multiplication of the number and the unit, e.g. 5*cm.

- some special phrases, e.g. X percent less than, have special translation e.g. (100-X)/100.
- questions and commands, e.g. find X, what is X etc., have the effect of marking X as a variable to be solved for.
- any remaining noun phrases are replaced by variable names, identical or similar noun phrases being replaced with the same variable.

2.3.3 Example

The price of a book is £18. If the price is 15% less than the marked price find the marked price.

clause 1 → price = 18*pounds
clause 2 → price = 0.85*marked_price
clause 3 → solve for marked_price

There are drawbacks

- Can only deal with limited subset of English, e.g. verbs like is find etc.
- Can only solve noun phrase reference problem in very limited cases where similar phrases share common string of words. (In the above example we know price of book and the price refer to the same object.) This general problem is extremely difficult.

2.4 Sir

(Semantic Information Retrieval - Raphael MIT '64)

A database of facts was set up. The system knew about the concepts of inclusion, part-whole, ownership, number, spatial position etc.

e.g. If A is above B and B is above C then it can conclude A is above C.

The system could handle arbitrary objects satisfying relations and only reasoned about the meaning of the relations. Deduction rules were associated with the relations such as the transitivity of `above` as expressed earlier. When new relations were added then of course new rules for deduction must be added. The system can only reason about the relations it has in its database though no restriction is placed on the actual objects that can satisfy these relations. See overhead for example session.

2.5 Lifer

(Hendrix et al '70's)

This is a fairly sophisticated natural language interface to a database of US naval ships throughout the world. The data contains information concerning the present location of particular ships together with details of size,displacement, number of men, name of captain etc. It is what is called a VLDB i.e. Very Large Data Base. The system uses a *semantic grammar* which is an extension of the normal notion of formal grammar. So instead of just knowing say `Nautilus` is a noun the system also knows it is a ship's name. It also knows such things as `displacement` is an attribute of a ship and not just a noun.

e.g. A pattern for a question which the system might handle could take the form:

`what is the <attribute> of <ship-name>`

then this could match the actual question

`what is the displacement of Nautilus`

It also has an ellipsis facility for processing incomplete questions.

e.g. If the following question had already been asked

`what is the length of the Constellation`

and then the next question were

`of the Nautilus`

then the system would assume that this meant

`what is the length of the Nautilus`

It would then answer this interpretation of the question but must tell the user that it is so doing! There is also a spelling corrector which checks word against its databases vocabulary and sees if the word can be transformed into one of these standard words by some typo such as transposing characters or the omission of a character. (It is actually the interlisp spelling corrector which it uses). Again the user must be informed that a word has been corrected in case this was not the intention. If the parser can not understand the input, the an error message is given and the user is told how much of the sentence is understood and in some cases what the system thinks the user wanted. The language can be extended by means of natural language commands.

For example synonyms can be set up such as

`define JFK like KENNEDY`

then `JFK` can be used subsequently anywhere `KENNEDY` could.

Lifer has been quite successful and apparently is used by the US Navy. Of course again the language of discourse is limited to naval facts and the database of knowledge and the dictionary is large.

2.5.1 Queries for Lifer

A typical query that Lifer could parse is:

```
What is the <attribute> of <ship-name>
```

This could match the following instances:

```
What is the displacement of Nautilus
```

or more complex example

```
What is the length, beam and displacement of Nautilus
```

The grammar rule for <attribute> takes the form:

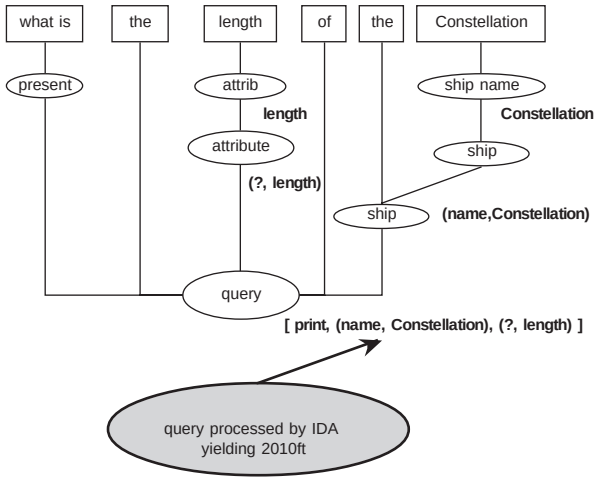
```
<attribute>  →  <attrib>  / [(?, <attrib>) ]
             →  <attrib>  and <attribute> /
                          [(?, <attrib>)|<attribute>']
             →  <attrib>  <attribute> /
                          [(?, <attrib>)|<attribute>']
```

The ' denotes the recursive translation for the other symbols. This could be easily expressed in Prolog using DCGs. An example of this will be seen later. The actual system used for implementing LIFER was InterLisp. In Prolog, the translation of the parsed expression would be stored in a variable and be in the form suitable for passing to IDA the underlying query language of the database.

Examples of other semantic grammar rules:

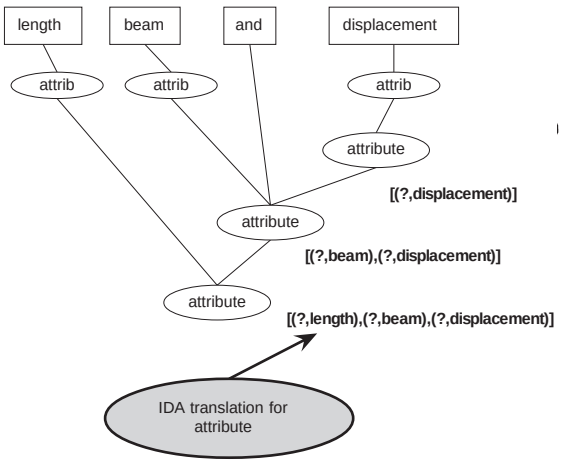
```
<present>  →  <what is>
<query>    →  <present> the <attribute> of <ship>
              /[print,<ship>'|<attribute>']
<ship>     →  the <ship> /<ship>|
           →  <ship-name> /(name,<ship-name>)
```

An example of parsing the first sentence together with its translation to the IDA example appears below:



Parse Tree

The translation and parse tree for attribute is illustrated below:



Parse Tree for attribute

We present a Prolog program using DCGs for the IDA query language. An interpreter is also given for the language which can deal with a simple form of ellipsis. (See listing).

2.6 Other Systems

I just mention four other systems.

2.6.1 Artificial paranoia

This is based on psychiatric study of paranoia. - see Raphael's book

2.6.2 SHRDLU

Winograd's SHRDLU system is famous as one of the first natural language systems that really seemed to work. It was based on semantic grammars and was restricted to the blocks world. This was concerned with the manipulation of coloured 3-D shapes on a table top. It was actually done on a graphics screen but could have been done by a robot arm.

The work was done in a high level programming language built on top of LISP called Planner — this had built in backtracking and had some of the features of Prolog. For more details see Winograd's book or for an overview Raphael's or Winston's book.

2.6.3 Lunar

Lunar which was a system for asking questions about rocks brought back from the moon had a natural language interface. Systems which are being built these days are based on much more sophisticated grammars than previously considered. One which I was slightly involved in was based on the grammatical theory built up by linguists called X-bar theory and was capable of handling really complex sentences.

2.6.4 Chat80

Chat80 is a system based on an extension of DCG's devised by F Pereira and written in Prolog. It allows the user to interrogate a database of geographical facts in NL.

2.7 Complexity of Semantics of Natural language

The meaning of a sentence can depend on many parts of the discourse. consider the problem of *anaphora* — determining what pronouns etc. refer to in previous sentences/utterances. Example

1. `John visited the doctor.`

2. `He thanked him for his time.`

The above would commonly be interpreted as `He` referring to `John` and `him` referring to `doctor` (using common-sense knowledge). But it is conceivable that the correct interpretation is the other way around. For suppose we have an earlier utterance:

3 John set off on his rounds to repair computers.

In this case it is more likely to be the other way round.

It is even possible for the meaning to depend on sentences appearing later!

Example 1 (Silly)

a) Statistics show that every 10 secs a man is mugged in New York.

Common sense would tell us that a different man is mugged every 10 secs. However if we add: b) and he's getting fed up with it! then there is only one man. You could imagine this kind of situation if a joke were being told. Presumably a true NL understanding program must cope with this, i.e. recognise the joke or pun.

Here is a more realistic example. Example 2

a) My son's team has not lost one game. — presumably they have won all games, however:

b) It was the last one they played.

— now it appears that they only won the last game; at least that's the only one we're sure about. These sort of problems (and more!) show how difficult it is to understand NL. To cope with such ambiguities we really have to store all possible interpretations which are updated as the discourse continues. We would incorporate some heuristics/common-sense reasoning to choose the most likely interpretation.

One theoretical approach to trying to tackle the above sort of problems has recently been developed by Barwise & Perry and is the theory of Situation Semantics — details can be found in their book *Situations & Attitudes*.

2.8 Conceptual Dependency

Roger Schank and his students developed a theory called conceptual dependency to try and understand basic story lines. He claimed many stories can be represented by just a few concepts. He developed a number of NL systems which would map the English descriptions of the stories into his representation. Different English sentences with the same meaning would be mapped to the same representation.

We shall outline a simplification of one of his systems called SAM *Script Applier Mechanism*. This would take a representation of a story and match it to a standard script which could then be used to flesh out the scenario and produce a paraphrase of the situation. So there would be an understanding of the story in terms of the script. They believed that you only need relatively few scripts to understand many human situations.

Shank used about eleven basic ACTs to represent meaning. We will just list four of these, simplifying the ideas somewhat:

1. `ptrans(Obj1,Obj2,Place1,Place2)` change the location of a physical object — `Obj1` moves `Obj2` from `Place1` to `Place2`.

2. `ingest(Actor,Food,Means)` take something to the inside of an animate object — `Actor` ingests `Food` by method `Means`.

3. `atrans(Actor1,Obj,Actor2,Actor3)` change an abstract relationship, such as ownership/possession with respect to an object — `Actor1` changes `Obj` from `Actor2`'s possession to `Actor3`.

4. `mtrans(Actor1,Info,Actor2)` transfer information — transfer `Info` from `Actor1` to `Actor2`.

There are also other concepts besides the ACTs such as *Times*, *Locations* and `Attributes`.

2.8.1 Scripts

Here is an example of a script, first in English, then in the representational form:
Restaurant

1. customer goes to the restaurant
2. customer goes to table
3. waiter brings food
4. customer eats food
5. customer pays cashier
6. customer leaves restaurant

We now represent the script as a Prolog assertion `script(Name,Script,Defaults)` where `Name` is just the name of the script, `Script` is a list of relations with free variables describing the script in terms of ACTs, and `Defaults` is a list of pairs, associating the variables to default names if they are not instantiated by the actual example which matches a subset of the script.

```
script(restaurant,
    [ptrans(Actor,Actor,EarlierPlace,Restaurant),
     ptrans(Actor,Actor,Door,Table),
     ptrans(Waiter,Food,Kitchen,Table),
     ingest(Actor,Food,mouth(Actor)),
     atrans(Actor,Money,Actor,Waiter),
     ptrans(Actor,Actor,Restaurant,ElseWhere)],
    [(Actor,customer),(EarlierPlace,place1),
     (Restaurant,restaurant), (Door,door),
     (Table,table),(Food,meal),
     (Waiter,waiter), (Kitchen,kitchen),
     (Money,cash),(ElseWhere,place2)] ).
```

2.8.2 SAM

When a short story is given, SAM looks for key words which will trigger the correct script, e.g. *Odeon* might trigger the *cinema* script or *McDonalds* might trigger the *restaurant* script. When this is done, the representation of the story is then matched to the script stored in the database, so that variables become instantiated. In general, only some of the terms are present in the story (i.e. the story will be a subsequence of the script), the script will fill out the rest. Any variables not instantiated from the story are then given their default values.

Here now is the simple Prolog version:

```
% filename: sam.pl
% simple form of Schank's SAM program
% on conceptual dependancy
% will paraphrase story based on standard script

sam(Story,Script) :-
    find(Story,Script,Defaults),
    match(Script,Story),
    override_defaults(Defaults).

find(Story,Script,Defaults) :-
    filler(Slot,Story),nonvar(Slot),
    trigger(Slot,Name),
    script(Name,Script,Defaults).

% Story is a just a subsequence of Script

match(Script,[]).
match([Line|Script],[Line|Lines]) :-
    match(Script,Lines).
match([_|Script],Story) :-
    match(Script,Story).

% Slot is a word in the story

filler(Slot,Story) :-
    member(Action,Story),
    Action =.. [_|Args], % this is just Prolog's unif operation
    member(Slot,Args).

% override default names by real names if present

override_defaults([]).
```

```
override_defaults([(N,N)|L]) :-  % unify with default if variable
   override_defaults(L).
override_defaults([(N1,N2)|L]) :-
    nonvar(N1),override_defaults(L).

trigger(mcdonalds,restaurant).
trigger(pizzahut,restaurant).
```

If we have the story:

Bill went to McDonalds, ate a bigmac and left

the the NL system would translate to

```
[ptrans(bill,bill,_,mcdonalds),
 ingest(_,bigmac,_),
 ptrans(Actor,Actor,_,_)]
```

the output from SAM will be

```
[ptrans(bill,bill,place1,mcdonalds),
 ptrans(bill,bill,door,table),
 ptrans(waiter,bigmac,kitchen,table),
 ingest(bill,bigmac,mouth(bill)),
 atrans(bill,cash,bill,waiter),
 ptrans(bill,bill,mcdonalds,place2)]
```

and this would be paraphrased:

Bill went to McDonald's. He was shown from the door to the table. The waiter brought Bill a bigmac. Bill ate the it by mouth. Bill paid the cashier. Bill left McDonald's.

Here is another example of a cinema script

```
% cinema script

script(cinema,
    [ptrans(Actor,Actor2,EarlierPlace,Cinema),
     atrans(Actor,Money,Actor,Cashier),
     ptrans(Actor,Actor,Foyer,Seat),
     mtrans(Actor,Film,Actor),
     ptrans(Actor,Actor2,Cinema,ElseWhere)],
    [(Actor,cinema_goer),(Actor2,cinema_goer2),
     (EarlierPlace,place1),(Foyer,foyer),
     (Cinema,cinema), (Door,door),
     (Seat,seat),(Film,film),
     (Cashier,cashier),
     (Money,cash),(ElseWhere,place2)] ).

trigger(odeon,cinema).
trigger(ritz,cinema).
```

Schank and his group went on to develop the foundations of an interesting technique called *Case Based Reasoning*. This allows one to develop expert systems based on past cases using pattern matching algorithms to choose the most appropriate case for the problem to be solved. This will be considered in the applications course.