# AI2 Natural Language Practical Exercise 1

To be handed in: Friday, 7th March 1986

The object of this exercise is to construct a program which can accept assertions like:

> Bill is a student.
> Fred is ambitious.
> If somebody is ambitious then he is industrious.
> Somebody is clever if he is a student and he is industrious.
> Bill is ambitious.

and answer questions like:

| | |
|---|---|
| Is Fred industrious? | (Answer: yes) |
| Is he clever? | (Answer: no) |
| Is somebody clever? | (Answer: yes, Bill) |

Assertions will be translated into Prolog clauses and inserted into the Prolog database, and questions will be translated into Prolog goals and evaluated by Prolog. Since all the inference required will be done automatically by Prolog, your program will only have to perform the translation from English into Prolog.

Your program should handle assertions of the following general forms:

- `if` Stmt1 and ... and StmtN `then` Stmt.
- Stmt `if` Stmt1 and ... and StmtN.
- Stmt.

where each statement (stmt) may take the forms:

- Noun `is` Adjective
- Noun `is` `a` Thing
- Noun `is` `an` Thing

and a noun is one of:

- a proper name (like `Bill`)
- `somebody`
- `he`

In English, there are many cases in which it is difficult to decide what a pronoun like "he" refers to. For simplicity, assume that "he" will always refer to the subject (i.e. the noun) of the most recent statement or question.

Questions may take any of the forms:

- `is` Noun Adjective?
- `is` Noun `a` Thing?
- `is` Noun `an` Thing?

Don't bother checking that "a" and "an" are used properly in statements and questions; "Is Fred an student" should be an acceptable question. If you wish, you may require that proper nouns, "things", and adjectives are taken from some fixed vocabulary, as long as your program is capable of producing the sample session below.

**Step 1:** Construct a context-free grammar for this language.

**Step 2:** Decide what the result of the translation should be for each form of Noun, Statement, Question, etc. The sample output below should provide some clues. How will you handle "he"? What about "somebody"?

**Step 3:** Write the program.

In the file `/ai/ai2teach/nl/prac1.tools` you will find definitions of the following handy predicates:

| | |
|---|---|
| `assertclause(`*C*`)` | Prints the clause *C* and adds it to the Prolog database. |
| `evalquery(`*G*`)` | Uses the term *G* as a goal to query the database and prints the result. |
| `read_in(`*S*`)` | Reads a sentence ending in . or ? and produces (in *S*) a list of the words in lower case. |

You will probably find Prolog grammar rules useful (see the notes from Prolog lecture 7), although you don't have to use them. If you do use them, you will need to use extra arguments to hold the result of the translation (see section 7.7 of the Prolog lecture notes) and you will probably need to add arbitrary Prolog goals (section 7.8). Try writing grammar rules to recognize the language first and then augment them to perform the translation.

In order to translate assertions into Prolog clauses and questions into Prolog terms, you will need to use the infix predicate "`=..`". The goal:

```
M =.. [Functor, Argument1, ..., ArgumentN]
```

instantiates M to the Prolog term:

```
Functor(Argument1,...,ArgumentN)
```

(Be careful: the "`..`" in "`=..`" is part of the predicate name, while the "..." between Argument1 and ArgumentN indicates repetition.) A clause is just a term with functor "`:-`" and the "`,`" is also just a functor, so the clause:

```
happy(P) :- man(P), rich(P), famous(P).
```

is just a convenient way of writing the following term:

```
':-'(happy(P),
     ','(man(P),
         ','(rich(P),famous(P))))
```

(Prolog seems to dislike lists like [':-', ...], and so when using "=.." to build clauses you need to say "N =.. [(':-'), ...]".) The clause "happy(bill)." is an abbreviation for "happy(bill) :- true."

Sample session

```
?- converse.
>> Bill is a student.
asserting:  student(bill) :- true
>> Fred is ambitious.
asserting:  ambitious(fred) :- true
>> If somebody is ambitious then he is industrious.
asserting:  industrious(_442) :- ambitious(_442)
>> Somebody is clever if he is a student and he is industrious.
asserting:  clever(_744) :- student(_744), industrious(_744)
>> Bill is ambitious.
asserting:  ambitious(bill) :- true
>> Is Fred industrious?
evaluating:  industrious(fred) ===> yes
>> Is he clever?
evaluating:  clever(fred) ===> no
>> Is somebody clever?
evaluating:  clever(_1225) ===> yes, bill
>> ......
```