Chapter 11
Projects in Prolog

This chapter contains a list of projects that you may wish to undertake in order to exercise your programming ability. Some of the projects are easy, but some may be appropriate as “term projects” as a part of a course in Prolog. The easier projects should be used to supplement the exercises in the previous chapters. The projects are in no particular order, although those in Section 11.2 are more open ended and ambitious, and will require some knowledge or background reading in various areas of artificial intelligence and computer science. A few of the projects assume knowledge about some particular field of study, so if you are not a mathematical physicist, do not feel discouraged if you cannot write a program to differentiate three dimensional vector fields.

A collection of Prolog programs is published in the report “How to solve it with Prolog”, edited by H. Coelho, J.C. Cotta, and L.M. Pereira. The report is distributed by the Laboratório Nacional de Engenharia Civil, in Lisbon, Portugal. It contains over a hundred small examples, problems, and exercises in areas such as deductive reasoning over databases, natural language, symbolic equation solving, and so forth. The report is not intended to be tutorial in nature, and so the Prolog programs in it are presented with limited explanatory accompaniment.

11.1 Easier Projects

1. Define a predicate to “flatten” a list by constructing a list containing no lists as elements, but containing all of the atoms of the original list. For example, the following goal would succeed:

   ?- flatten([a,[b,c],[[d],[]],e], [a,b,c,d,e]).

   There are at least six distinct ways to write this program.

2. Write a program to calculate the interval in days between two dates expressed in the form Day-Month, assuming they refer to the same year which is not a leap year. Notice that “-” is simply the infix form of a 2-ary functor. For example, the following goal would succeed:

   interval(3-march, 7-april, 35).
3. In Chapter 7 sufficient information is given to construct programs to differentiate and simplify arithmetic expressions. Extend these programs so they will handle expressions containing trigonometric functions, and if you desire, differential geometry operators such as div, grad, and curl.

4. Write a program to produce the negation of a propositional expression. Propositional expressions are built up from atoms, the unary functor not, and binary functors and, or, and implies. Provide suitable operator declarations for the functors, perhaps using the operator declarations (¬, &, #, and Æ) in Chapter 10. The negated expression should be in simplest form, where not is only applied to atoms. For example, the negation of

\[ p \implies (q \land \neg r) \]

should be

\[ p \land (\neg q \lor r) \].

5. A concordance is a listing of words that occur in a text, listed in alphabetical order together with the number of times each word appears in the text. Write a program to produce a concordance from a list of words represented as Prolog strings. Recall that strings are lists of ASCII codes.

6. Write a program that understands simple English sentences having the following forms:

\[
\_ \text{ is a } \_ \\
A \_ \text{ is a } \_ \\
\text{Is } \_ \text{ a } \_?
\]

The program should give an appropriate response (yes, no, ok, unknown), on the basis of the sentences previously given. For example,

John is a man.
ok
A man is a person.
ok
Is John a person?
yes
Is Mary a person?
unknown

Each sentence should be translated into a Prolog clause, which is then asserted or executed as appropriate. Thus, the translations of the preceding examples are:

\[
\text{man(john)}.\\
\text{person}(X) :- \text{man}(X).
\]