Supplement 1

Jocelyn Paine

June 4, 1996

1 Introduction

These exercises follow on from Lesson 1, and show you some more examples of how you can use Prolog to represent everyday situations. Try all the exercises: you may already have done some of them.

2 A geographical knowledge base (PP)

The knowledge base called `countries.pl` contains a number of facts about European countries, capitals, rivers and seas. You can view it by doing `showlib countries.pl` from Prolog.

I hope the meaning is clear. In the facts for `has_capital`, the first name is always that of a country, the second of a town (or city). In those for `is_on`, the first name is a town, and the second is either a river or a sea.

`linked_to` is about rivers and seas: there is one fact for each river that flows into another river or a sea. `next_to` tells you which seas (not rivers) are next to one another.

`borders` is like `next_to`, but is about countries, not seas. It tells you which countries share a common border.

Finally, `on_sea` tells you which countries adjoin a given sea.

2.1 Some questions

To start, please load the new knowledge base by doing

```
library(countries).
```

when in Prolog.

Now try some questions. Can you:

- Ask a question to find the capital of `bulgaria`. Remember that all names of things are in lower case, even though in English, they would begin with a capital letter.
- Ask which country has capital `vaduz`.

- Find on which river `vaduz` lies.
- Find which countries adjoin the `irish_sea`.
- Find which countries have exactly the same name as their capitals.

Was this knowledge base written before or after German unification? Before or after Berlin became capital? Before or after the breakup of the USSR?

2.2 Using `and`

Can you ask questions to find out:

- Which country is both on the `baltic` and the `skagerrak`?
- Which country is on the `black_sea`, and has a capital which is on the `dimbo_vita`?

There is another fact in this knowledge base, called `flows_to`. This is defined as follows:

```
flows_to(X, Y ) :-
    linked_to( X, Y ).

flows_to( X, Y ) :-
    linked_to( X, Z ),
    flows_to( Z, Y ).
```

The idea it expresses is that river X flows to river or sea Y if the water from X flows directly or indirectly to Y. Try the question `flows_to(X,Y).` and compare it with the results from `linked_to(X,Y).`.

Translate the definition into good English (using pronouns instead of variables). Why do you think it works? `flows_to` is your first example of a technique, very common in logic programming, called *recursion*, where something is defined in terms of itself. If you have studied grammars in linguistics, or programmed in high-level languages before, you may have encountered it already. We shall see several more examples.

3 A ski-resort advisor (PP,AI)

In this section, I'm going to show you how you can use Prolog as a simple "expert system": a program that can be used to give expert advice about some rather restricted area of knowledge, and which can justify its conclusions.

Please look at your printout of the `skier.pl` knowledge base. If you don't have it, go into Prolog and do `showlib skier.pl`. This program advises skiers on their choice of resort, depending on fitness (as measured by the number of pressups that can be done without collapsing), purpose (is it serious training, or just entertainment), and the number of lessons already taken.

3.1 Translating into English

Have a good look at the facts. When you have done that, write down in colloquial English what these facts mean. Don't keep to the "if—and" form used by Prolog (unless you are in the habit of phrasing your sentences like that anyway :-). Instead, act as though you are translating from (say) French, and trying to make your translation read as naturally as possible.

You will see that the knowledge base uses the symbols `<` and `>=`. `<` means "less-than", as you might expect. `>=` means "greater-than-or-equal", and is the nearest the keyboard can come to the standard mathematical symbol for this operation. We'll say more about ways to compare numbers in Lesson 5.

Notice that all the conditions are complete and mutually exclusive. For example, the three rules for `is_a` cover all possible combinations of fitness and number of lessons, and no two rules overlap. The same goes for `has_fitness`. When you write your own rules, you need not make them complete (i.e. covering all possible cases), but you should think carefully about which cases are left out. It is a good idea to make rules mutually exclusive. To see what happens when they're not, consider these rules:

```
has_fitness(Skier,poor) :-
    max_pressups(Skier,P),
    P < 15.

has_fitness(Skier,good) :-
    max_pressups(Skier,P),
    P >= 10.
```

Suppose that a skier can do a maximum of twelve pressups before dropping dead. The first rule then says that he has poor fitness; but the second contradicts it, saying he's got good fitness. If you were to make this change, and then ask for advice about someone in this situation (the next section tells how to do this), you'll find that you get two solutions, offering conflicting advice. This is not surprising: given the rules above, both solutions would be logically correct. In general, this is something to avoid. Unfortunately, the need to make rules mutually exclusive can make them rather verbose. There are ways to avoid this, but they require more knowledge of Prolog and of the order in which it visits facts during inference.

3.2 Running the program

Assume you are advising a prospective skier. Not being terribly good, he can perform only four pressups without collapsing, and he has taken so far 174 skiing lessons. Not surprisingly, his purpose in seeking a resort is to have fun rather than taking the subject seriously. You can express this in Prolog as follows:

```
    max_pressups( eddie, 4 ).
    had_lessons( eddie, 174 ).
    wants( eddie, fun ).
```

The name of the skier is of course arbitrary, and any name would do as long as you use it consistently.

Enter this information into a file using VED. Then ask Prolog what resort he should go to. Which facts do you think it used to infer the answer?

Now, consider another skier — Charles. Charles has had 10 lessons, his purpose is entirely serious, and he can do 62 pressups in one session. Enter these three facts in such a way that they replace the three you have about Eddie.

Where does Prolog say Charles should go? Which facts would it have used?

3.3 Changing the knowledge-base

People who write expert systems often need to update the knowledge bases to reflect changes in their subject. There are now a number of commercially available expert systems for giving advice on tax — how to get the best relief on your mortgage, how to minimise corporation tax, and so on — and at least one experimental system for advising on Social Security benefits, though the Government seems to have decided not to make the latter generally available to claimants. Such systems have to be modified yearly to keep up with the annual amendments to income tax and decreases in benefits. In other cases, changes are needed because our knowledge, or the way we expressed it in rules, was incorrect and needs modifying. This is often the case with systems for treating electronics faults, for instance. This section puts you in the place of an expert systems programmer.

As it stands, the meaning of good fitness is rather optimistic. It would be more sensible to say that at least 50 pressups are needed. Change the knowledge base to reflect this.

Also, we would get finer gradation with a new fitness level: medium. So let's say now that someone who can do at least 25 pressups is medium fit. If they can't do that many, they're still of poor fitness. Make such a change.

Now, let's have two new ratings together with beginner and advanced: pro and semi-skilled.

Pro is better than advanced. Obviously, someone can be a pro only with a fitness level of good. And every pro must not only have good fitness, but also have taken at least 70 skiing lessons.

We shall downgrade the meaning of beginner. Now, we shall say that everyone is a beginner if they have taken nine or fewer lessons. A skier who has taken up to 20 lessons is also a beginner if his (or her) fitness is poor; but counts as semi-skilled if of medium (or good) fitness. With 20 lessons or above, any person whose fitness is poor is still a beginner no matter how many lessons they have had. Likewise, a skier of medium fitness is semi-skilled, regardless of the number of lessons.

Actually, you are also only semi-skilled even up to 40 lessons, even if your fitness is good. But should you have had 40 or more, you count as advanced.

Now, add new facts (replacing existing ones if necessary) so they correctly reflect these regulations. Test what you've done by adding facts as you did for Eddie and Charles, and asking questions about rating.

Note that Prolog can't find resorts for the new ratings. This is not surprising, as there are no facts covering them. Think of some new resorts, and add them to your knowledge base.

3.4 Comparison with expert systems (AI)

Some of you have seen the Alvey expert system video in the introductory lecture for this course, or have encountered them in H.C.I. or A.I. tutorials.

Like an expert system, Prolog uses rules to infer new facts from old. But the user interface differs from that of a typical expert system. How?

If you don't know anything about expert systems, then imagine yourself as a novice skier who wants advice on the most suitable resort. You also have little experience with computers, and you can't type. Do you think this means of getting advice is ideal?

What about the expert system programmer? What aids would be useful when updating the knowledge base?

4 Inference (AI)

Lesson 1 introduced the term *inference*, which is the deduction of new facts from old ones. In expert systems work, and logic programming generally, we distinguish two main kinds of inference.

4.1 Forward chaining

One kind is called *forward chaining*, and sometimes *data directed*, because inference is triggered by the arrival of new data in the knowledge base. For example, suppose we have the knowledge base:

```
has_exam( X, finals ) :- is_in_year( X, 3 ).
lives_in( X, in_college ) :- is_in_year( X, 3 ).
needs_to( X, work ) :- has_exam( X, finals ).
```

and we add the fact

```
is_in_year( john, 3 ).
```

A forward-chaining system would immediately trigger any rules whose conditions are satisfied by this fact. In this example, the conditions the first two rules are both satisfied — i.e. both are true when we set X=`john`. So the system would create the two facts below and add them to the database

```
has_exam( john, finals ).
lives_in( john, in_college ).
```

As each new fact is created, the system would look for yet more rules that its arrival could trigger. In this case, the fact `has_exam(john,finals)` would trigger rule 30, because `has_exam(X,finals)` is true if X=`john`. This would result in the addition of another new fact:

```
needs_to(john,work).
```

The fact `lives_in(john,in_college)` would not trigger anything else though.

Generally speaking, in a forward-chaining system, once a fact is created, it is there to stay until explicitly deleted. If this is so, answering a question need not itself require any inference. If on receiving a new fact, the system makes all possible inferences from it, then answering a question need only involve scanning these facts one by one.

4.2 Backward chaining

In distinction to forward chaining is *backward chaining*. This is sometimes called *goal-directed inference*, because inferences are not made until the system is made to prove a particular goal (i.e. a question). This is a lazy kind of inference. It does no work until absolutely necessary, in distinction to forward chaining, where the system eagerly snaps up new facts and immediately tries applying existing rules to them as soon as they arrive.

So, if we had the same knowledge base again and we were to add

```
is_in_year( john, 3 ).
```

the system would not try to trigger any conditions. But suppose we were then to ask the question

```
needs_to(X,work).
```

The system would try to prove this by searching for a suitable rule or unconditional fact, i.e. ones whose conclusions are about the same "verb", `needs_to`. Here, the last fact is the only one. What this fact means, in effect, is "to prove someone needs to work, prove they have Finals". So the system would then pose itself the subtask of proving that John has finals. The first fact says something about this: "to prove someone has Finals, prove they're a third year". So the system now poses the sub-sub-task of proving John to be a third year. And we have an unconditional fact which says he is. So we can complete the proof.

4.3 Which does Prolog use?

In psychology, a common way of investigating human information processing is by response time experiments. Suppose with Prolog that you had some way of measuring the time between adding a fact and getting the prompt, and between asking a question and getting the answer. How could you determine which kind of inference is going on?

If you don't like thinking about the psychology of computers, put yourself in the place of a psychologist who has just discovered live Martians. They speak a kind of

English: to be more exact, they behave just as Prolog would if it worked on speech not text. Our psychologist has discovered that the Martians make the same kind of inferences as Prolog: but is this by forward or backward chaining?

5 English versus Prolog (PP)

5.1 The uses of “and”

Consider this knowledge base, which expresses some facts about various under- and post-graduates.

```
is_friends_with( paul, stan ).
is_friends_with( stan, paul ).
is_in_year( paul, 1 ).
is_in_year( stan, 2 ).

has_status( Student, post_graduate ) :-
    is_in_year( Student, N ),
    N > 3.

belongs_to( Student, gcr ) :-
    has_status( Student, post_graduate ).

has_post( paul, jcr_treasurer ).
has_post( stan, jcr_president ).

has_duty( Officer, attend_governing_body_meetings ) :-
    has_post( Officer, jcr_president ).
has_duty( Officer, attend_ousu_meetings ) :-
    has_post( Officer, jcr_president ).
has_duty( Officer, organise_elections ) :-
    has_post( Officer, jcr_president ).
has_duty( Officer, keep_the_books ) :-
    has_post( Officer, jcr_treasurer ).
```

Write, in Prolog, the sentences below. Explain the different meanings of “and” in each.

- Dave and Jason are third-years.
- Dave and Jason are friends.
- The JCR Warden’s duties are mending the pinball machine and the video-games and selling punt cards.
- First-years, second-years, and third-years are undergraduates and members of the JCR.

5.2 The uses of "if"

Suppose a friend of yours asks, having seen you do this practical: "I want to go skiing for fun. If I can do 45 pressups in one go, and I've taken 6 lessons so far, which resort should I go to?".

How does his use of the word "if" differ from that in a statement like "Someone is a beginner if they have taken no more than ten lessons"?

5.3 "And" versus "or"

It can seem that the connection between two facts for the same "verb" sometimes means "and" and sometimes "or". For example

```
has_moons( earth, 1 ).
has_moons( mars, 2 ).
```

mean "Earth has one moon *and* Mars has two moons", while

```
is_a( Skier, beginner ) :-
    had_lessons( Skier, L ),
    L < 10.

is_a( Skier, beginner ) :-
    has_fitness( Skier, poor ).
```

means "A skier is a beginner if they had fewer than ten lessons, *or* if their fitness is poor"? Why is this?

5.4 How rules are phrased

Go through the section on the ski advisor. Make a list of all the different English phrases there that you translated into conditional facts, and note the way pronouns are used in each. Can you think of any other phrasings?

6 Knowledge engineering (AI)

Much of the business of programming is converting vague everyday language into precise programs. In A.I., we often have to do this when putting the knowledge into expert systems, as we saw earlier.

It is the job of the *knowledge engineer* to interview experts and set them trial problems, then convert the results into something an expert system can handle. Experts usually are not able to talk systematically through a topic, giving neat syllogisms like those in the other exercises. Instead, they'll ramble, throwing in little snippets of information as something prompts its recall. Below, I imagine such a monologue from a G.P. who's being interviewed during the construction of a medical expert system.

Can you translate this into Prolog? What you need to do is to identify key concepts in the monologue: kinds of treatment, different types of illness, the various symptoms, and so on. Then tease out all the logical connections, throwing away any un-necessary ones. Finally, write down the rules, after choosing suitable names for your facts.

> You were asking what drugs I bought last month. Well, the main one's a new product from Hoffnung-Gavroche called brontomycin — if I buy more than 30 g each month, they throw in a free skiing holiday at the end of the year. They're very good with these perks. I must have gone on at least 50 skiing lessons since I started with Hoffnung. Of course, I only go for fun; to train seriously you need to be really fit, and with all these drug company bashes I've been to recently, I couldn't even do a pressup without collapsing, let alone the daily half-marathons those professional skiers seem to thrive on.
>
> It seems to be good stuff though... Treats what we call remediable infections. Fowlpox's one such. Very easy to prescribe, because it works the same for all remediables. Just give the patient a daily dose of 10 mg for four weeks.
>
> I treated the Mayor with it last week when he returned from a holiday in Enschede in Holland. He had Dutch measles. That's another remediable infection. You want to know the symptoms ... itchy red spots all over and high temperature. As with all remediable infections, you have to quarantine for three weeks, otherwise the kids get it. And that's where you must take care to halve the dose, if you don't then some children may develop eczma, hyperactivity and floating ribs.
>
> I remember that because the hairdresser caught fowlpox recently. He does pets as a sideline, and the local chicken farmer had called him in for a root-perm and auburn rinse on his prize Leghorns before they went up to the Royal Poultry Show. It's like Dutch measles, but the temperature crashes to 92 or so. Anyway, he refused my quarantine order ... wanted the money I suppose ... it was just before the school sports day, and all the parents were bringing in their brats to be smartened up. Whole school went down with it.
>
> Thought at first he might have an allergy. Some people are allergic to the plant extracts in deodorants, and get itchy red spots ... or sometimes hard black lumps in the armpits instead. No trouble to treat — like all allergies you remove the cause for a trial period, that's ten days, and then desensitise. He never uses them though — most unpleasant in there on a hot day. Smells like the Black Hole of Calcutta...
>
> There's another allergy. James the florist came in last month, he'd was shaking with cold, had severe itching and red spots all over. Not the chickens this time; I knew he works with exotic plants from the nursery up there, and so I asked him whether he'd handled any of those Rumanian "Devil's Lovebite" creepers. That causes a plant allergy too: "False Fowlpox" be-

cause it looks the same if you don't know the cause.

Someone else had that ... well actually it wasn't the allergy but something else. A senior chemist from the Grimbledon Down bio-warfare place. She had exactly the same symptoms. Nasty place, Grimbledon Down... they run recombinant-DNA analysis on plague strains. Suppose it's safe, but... standing instructions from the MoD if anyone shows up with high temperatures and black lumps or itchy red spots: plague's non-remediable, so there's no treatment, not even brontomycin. But you must take safety precautions. Not really quarantine, no ... evacuate the country for a radius of five miles, burn the corpses, and shoot all the foxes, rats, badgers, and voles. Oh, and the chickens.

Anyway, get back to the chemist ... she'd been refining batavium. They use it in cold-fusion research. Left the fume-cupboard open, and got a whiff ... came down with the same symptoms as "Devil's Lovebite". But if it's caused by batavium, then it's not a plant extract allergy, but heavy metal poisoning. You have to flush out the whole system with chelates, and place on dialysis for four weeks. That reminds me ... if she'd had a remediable infection, I'd have to replace the brontomycin by 30 mg daily of prestatine ... it seems that more than 500 gm daily of bronto causes birth defects in rats, so it's been forbidden for pregnant women...

Now translate the following question, and use Prolog to answer it: Kylie, having just returned from a holiday in Eindhoven, has itchy red spots all over and a high temperature. Do you recommend any safety precautions such as quarantine? What treatment is recommended? Would you change this if she were pregnant? What's the cause?

7 Some terminology

I want to finish with some terminology. So far, I've talked about the "verb" in a fact, such as `loves` or `suits` or `has_duty`. This is not a very good name, because the word often doesn't take the form of a verb. It may for example be a preposition or a preposition plus a verb, such as `february is_after january`. Programmers often write like this when describing spatial or temporal relationships. In future therefore, I shall borrow the terminology of formal logic, and call this part of a fact the *predicate*. All the Prolog literature uses this name.

The things in brackets are the predicate's *arguments*. So, in `john loves mary`, the arguments are `john` and `mary`. This is a specialised meaning of "argument", and it has nothing to do with disputes or disagreements; again, it comes from logic.

I shall use the word *rule* to mean a conditional fact from now on, that is one containing an `if`. Prolog is not the only type of programming system to use "if ... then ..." statements; many expert systems and other AI programs do too, and such statements are generally referred to as rules.

When I want to talk about an unconditional fact (no `if`) I shall say so specifically; I shall use the word *fact* on its own to mean either. Sometimes I may use the word *clause* instead of fact, following the custom of the Prolog literature.

I shall also talk about *defining a predicate*. In logic programming, the meaning of a predicate is defined solely by the clauses for it. There is no other authority Prolog can appeal to; it does not have an internal encyclopaedia telling it what a predicate "ought" to mean (apart from a very few "built-in" predicates, which you'll meet later).

Predicate definitions can be of two kinds. Some are what logicians call *intensional*: they consist of a general rule or two which try to capture the essence of (say) what it means for one person to be the ancestor of another, or what it means for one number to be the factorial of another. You will see some of these in Lesson 2. For example, if Bert is an ancestor of Charles, and Amanda of Caroline, there is something in common about the way both are related. Bert has a child who is Charles, or who has a child who is Charles, or who has a child who is Charles, ... Amanda has a child who is Caroline, or who has a child who is Caroline, or who has a child who is Caroline, ... To write intensional definitions, you have to extract these structural regularities and work out how to formalise them in logic.

Other predicates are *extensional*. Here, the definition is often just a long list of unconditional facts. For example, the facts you accumulated about `loves` in Lesson 1. Expert system rules for domains like medicine are often of this form; lacking a decent theory of the underlying principles, all you can do is to write down all the rules of thumb that experts have built up by experience. The knowledge bases of the skiing and disease exercises are of this kind.

In either case, the definition of a predicate will consist of one or more clauses. Each clause will express some part of the meaning, but you need them all in order for the definition to be complete.

7.1 Naming predicates

Finally, a note about names. When writing new facts, you may have to choose names for your predicates — names like `is_a`, or `loves`, or `has_part`. Please avoid the name `is`, which means something special to Prolog. If you try to define a fact whose predicate is `is`, you will get confusing messages about updating built-in facts.